# Time Tracker Project version 3.2 Component Specification

Updates new in version 3.2 are in red.

## 1.Design

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

The design is separated into 3 separate types of beans - Project, ProjectManager, and ProjectWorker. For each type of bean, there are 2 layers of management: The topmost layer is composed of the Utility classes, which provide the functionality needed to update, retrieve and search from the data store. Batch operations are also available in the Utility classes. The next layer is the DAOs, which interact directly with the data store. There is also a sublayer, called the FilterFactory layer, which are responsible for building search filters which can be used to search the filter.

There is an additional J2EE layer on top of the component. This layer is composed of a SessionBean and a BusinessDelegate. The SessionBean is stateless, and supports local interfaces. The BusinessDelegate doubles as a service locator, and looks up the Session Bean upon construction, delegating all similar method calls to the SessionBean. This layer is responsible for transactions, allowing a J2EE container to manage transactions for the component.

*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. The deployment descriptor will need to set the transaction level for ProjectUtilitySessionBean, ProjectManagerUtilitySessionBean, and ProjectWorkerUtilitySessionBean to REQUIRED.ConfigurationManagerSpecificationFactory for ObjectFactory.
2. Since the transaction is now handled by the EJB container, transaction management must be removed from the DAO classes.

### 1.1Design Patterns

***The Strategy Pattern.*** *The DAO classes such as ProjectDAO, ProjectManagerDAO, ProjectEntryDAO and ProjectWorkerDAO are all strategies for persisting and retrieving information from a data store.*

***The Facade Pattern.*** *The ProjectUtility class encapsulates some subsystem DAOs to provide a unified API that makes it easier to manage the projects.*

***The Factory Pattern.*** *The Factory pattern is used in the FilterFactory layer. The ProjectFilterFactory, ProjectWorkerFilterFactory and ProjectManagerFilterFactory employ this pattern.*

***The Business Delegate Pattern.*** *The Business Delegate pattern is used with the ProjectUtilityDelegate, ProjectManagerUtilityDelegate and ProjectWorkerUtilityDelegate* *classes.*

***DAO Pattern.** Used to abstract persistence from the rest of the application.*

## 1.2 Industry Standards

*None*

## 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.

#### 1.3.1.1 Project Class and Project Table (Used by DbProjectDAO)

Column project_id maps to the id property in the TimeTrackerBean superclass of the Project class.
Column company_id maps to the companyId property in the Project class.
Column description maps to the description property in the Project class.
Column name maps to the name property in the Project class.
Column start_date maps to the startDate property in the Project class.
Column end_date maps to the endDate property in the Project class.
Column creation_date maps to the creationDate property in the TimeTrackerBean superclass of the Project class.
Column creation_user maps to the creationUser property in the TimeTrackerBean superclass of the Project class.
Column modification_date maps to the modificationDate property in the TimeTrackerBean superclass of the Project class.
Column modification_user maps to the modificationUser property in the TimeTrackerBean superclass of the Project class.
Column sales_tax maps to the salesTax property in the Project class.
Column active maps to the active property in the Project class.

#### 1.3.1.1 ProjectManager Class and project_manager Table (Used by DbProjectManagerDAO)

Column project_id maps to the projectId property in the ProjectManager class.
Column creation_date maps to the creationDate property in the TimeTrackerBean superclass of the ProjectManager class.
Column creation_user maps to the creationUser property in the TimeTrackerBean superclass of the ProjectManager class.
Column modification_date maps to the modificationDate property in the TimeTrackerBean superclass of the ProjectManager class.
Column modification_user maps to the modificationUser property in the TimeTrackerBean superclass of the ProjectManager class.
Column start_date maps to the startDate property in the ProjectManager class.
Column end_date maps to the endDate property in the ProjectManager class.

#### 1.3.1.2 ProjectWorker Class and project_manager Table (Used by DbProjectWorkerDAO)

Column project_id maps to the projectId property in the ProjectWorker class.
Column creation_date maps to the creationDate property in the TimeTrackerBean superclass of the ProjectWorker class.

Column creation_user maps to the creationUser property in the TimeTrackerBean superclass of the ProjectWorker class.
Column modification_date maps to the modificationDate property in the TimeTrackerBean superclass of the ProjectWorker class.
Column modification_user maps to the modificationUser property in the TimeTrackerBean superclass of the ProjectWorker class.
Column start_date maps to the startDate property in the ProjectWorker class.
Column end_date maps to the endDate property in the ProjectWorker class.
Column pay_rate maps to the payRate property in the ProjectWorker 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 Projects (we reduce the retrieved data to only the id, but developer may modify it to retrieve all relevant fields):

```
SELECT
      project_id
FROM
      project
INNER JOIN
      project_manager
      ON
      project_manager.project_id = project.project_id
INNER JOIN
      project_worker
      ON
      project_worker.project_id = project.project_id
INNER JOIN
      project_expense
      ON
      project_expense.project_id = project.project_id
INNER JOIN
      project_time
```

```
        ON
        project_time.project_id = project.project_id
INNER JOIN
        project_fix_bill
        ON
        project_fix_bill.project_id = project.project_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 description was created using createContainsProjectManagerFilter, then the Search Builder would add:

```
WHERE (continued from above)
        project_manager.user_account_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 PROJECT_MANAGER_ID_COLUMN_NAME should be "project_manager.user_account_id".

Finally, note that this may be optimized by adding a GROUP BY clause to group the results by ids (this is especially important if a project manager, project worker, or enty filter is not specified, since it would return multiple rows for each entity associated with the project). The GROUP BY clause may be manually added by overriding buildSearchContext method, and adding the GROUP BY clause after the superclass' buildSearchContext has been completed. This will need to be done in a private inner subclass of SearchStrategy.

### *1.3.3 Container managed transactions*

Transactions are managed container managed. This component doesn't need to commit any transactions. However, if an operation fails, the NotificationSessionBean should call sessionContext.setRollbackOnly(). It will be necessary to set the transaction level of the various Session Beans to REQUIRED in the deployment descriptor.

## 1.4Component Class Overview

**Package com.topcoder.time.tracker.project**

**Project**

This is the main data class of the component, and includes getters and setters to access the various properties of a Time Tracker Project, such as the project name, timeline, contact information, and address.

**EntryType (Enumeration)**

This is an enumeration that is used to distinguish between the different types of Entries that may be added to the project. At the moment, it supports Time, Expense, and Fixed Billing entries.

**ProjectWorker**

This is a bean class representing a user that is working on the project. It contains various properties, such as the id of the user who is working on the project, and the id of the project which the user is working on. The pay rate and work timeline of the worker is also available.

**ProjectManager**

This is a bean class that represents a Project Manager. A ProjectManager is a user that is responsible for managing the project. Various properties are available, such as the ids of the project and user involved. The timeline when the user is expected to manage the project is also provided.

**ProjectUtility (interface)**

This interface represents the API that may be used in order to manipulate the various details involving a Time Tracker Project. CRUDE and search methods are provided to manage the Projects inside a persistent store. There are also methods that allow various Time, Expense, and FixedBilling entries to be associated with a Project. It is also possible to search the persistent store for Projects based on different search criteria.

**ProjectWorkerUtility (interface)**

This interface represents the API that may be used in order to manipulate the various details involving a Project Worker. CRUDE and search methods are provided to manage the ProjectWorkers inside a persistent store. It is also possible to search the persistent store for ProjectWorkers based on different search criteria.

**ProjectManagerUtility (interface)**

This interface represents the API that may be used in order to manipulate the various details involving a Project Manager. CRUDE and search methods are provided to manage the ProjectManagers inside a persistent store. It is also possible to search the persistent store for ProjectManagers based on different search criteria.

**ProjectDAO (interface)**

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

**ProjectEntryDAO (interface)**

This is the interface definition for the DAO that is responsible handling the association and dissassociation of an Entry identifier with a project. It is also capable of retrieving the entries that have been associated with a particular project. The DAO is also responsible for generating any identifiers (if necessary) for the associations.

**ProjectWorkerDAO (interface)**

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

**ProjectManagerDAO (interface)**

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

**ProjectManagerFilterFactory (interface)**

This interface defines a factory that is capable of creating search filters used for searching through Project Managers. 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 ProjectManagerUtility that supports the given schema.

**ProjectWorkerFilterFactory (interface)**
This interface defines a factory that is capable of creating search filters used for searching through Project Workers. 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 ProjectWorkerUtility that supports the given schema.

**ProjectFilterFactory (interface)**
This interface defines a factory that is capable of creating search filters used for searching through Time Tracker Projects. 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 ProjectUtility 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.

**ProjectUtilityImpl**
This is a default implementation of the ProjectUtility interface. it utilizes instances of the ProjectDAO and ProjectEntryDAO in order to fulfill the necessary CRUDE and search operations defined.

**ProjectWorkerUtilityImpl**
This is a default implementation of the ProjectWorkerUtility interface. it utilizes instances of the ProjectWorkerDAO in order to fulfill the necessary CRUDE and search operations defined.

**ProjectManagerUtilityImpl**
This is a default implementation of the ProjectManagerUtility interface. it utilizes instances of the ProjectManagerDAO in order to fulfill the necessary CRUDE and search operations defined.

**Package com.topcoder.time.tracker.project.db**

**BaseDAO (abstract class)**
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.

**DbProjectEntryDAO**
This is the default implementation for the DAO that is responsible handling the association and dissassociation of an Entry identifier with a project. It is also capable of retrieving the entries that have been associated with a particular project. The DAO is also responsible for generating any identifiers (if necessary) for the associations.
Transaction management is removed from this class in version 3.2.

**DbProjectDAO**
This is default implementation of the ProjectDAO that is for the database schema that was defined in the Requirements for Time Tracker Project 3.1. It supports all the methods needed to manipulate and retrieve a Project in the data store.
Transaction management is removed from this class in version 3.2.

**DbProjectManagerDAO**

This is default implementation of the ProjectManagerDAO that is for the database schema that was defined in the Requirements for Time Tracker Project 3.1  It supports all the methods needed to manipulate and retrieve a ProjectManager in the data store.
Transaction management is removed from this class in version 3.2.

**DbProjectWorkerDAO**

This is default implementation of the ProjectWorkerDAO that is for the database schema that was defined in the Requirements for Time Tracker Project 3.1  It supports all the methods needed to manipulate and retrieve a ProjectWorker in the data store.
Transaction management is removed from this class in version 3.2.

**DbBaseFilterFactory**

This is an implementation of the BaseFilterFactory that may be used for building searches in the database. It maintains a set of column names that are necessary for the filter criterion that it supports, and builds filters according to the specified column names.

**DbProjectFilterFactory**

This is an implementation of the ProjectFilterFactory that may be used for building searches in the database. It maintains a set of column names that are necessary for the filter criterion that it supports, and builds filters according to the specified column names.

**DbProjectManagerFilterFactory**

This is an implementation of the ProjectManagerFilterFactory that may be used for building searches in the database. It maintains a set of column names that are necessary for the filter criterion that it supports, and builds filters according to the specified column names.

**DbProjectWorkerFilterFactory**

This is an implementation of the ProjectWorkerFilterFactory that may be used for building searches in the database. It maintains a set of column names that are necessary for the filter criterion that it supports, and builds filters according to the specified column names.

**Package com.topcoder.time.tracker.project.ejb**

**ProjectUtilityLocal**

Local interface for ProjectUtility.  It contains exactly the same methods as ProjectUtility interface.

**ProjectUtilityLocalHome**

LocalHome interface for the ProjectUtility.  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 Session Bean.

**ProjectUtilityDelegate**

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 ProjectUtility, and delegating any calls to the bean.

**ProjectUtilitySessionBean**

This is a Stateless SessionBean that is used to provide business services to manage Projects within the Time Tracker Application. It contains the same methods as ProjectUtility, and delegates to an instance of ProjectUtility.
In version 3.2, the internal impl field is instantiated in the ejbCreate() method, which will

read configuration as necessary. If an operation throws an exception, sessionContext.setRollbackOnly() will be called before rethrowing the exception.

**ProjectManagerUtilityLocal**

Local interface for ProjectManagerUtility. It contains exactly the same methods as ProjectManagerUtility interface.

**ProjectManagerUtilityLocalHome**

LocalHome interface for the ProjectManagerUtility. 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 Session Bean.

**ProjectManagerUtilityDelegate**

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 ProjectManagerUtility, and delegating any calls to the bean.

**ProjectManagerUtilitySessionBean**

This is a Stateless SessionBean that is used to provide business services to manage Projects within the Time Tracker Application. It contains the same methods as ProjectManagerUtility, and delegates to an instance of ProjectManagerUtility.
In version 3.2, the internal impl field is instantiated in the ejbCreate() method, which will read configuration as necessary. If an operation throws an exception, sessionContext.setRollbackOnly() will be called before rethrowing the exception.

**ProjectWorkerUtilityLocal**

Local interface for ProjectWorkerUtility. It contains exactly the same methods as ProjectWorkerUtility interface.

**ProjectWorkerUtilityLocalHome**

LocalHome interface for the ProjectWorkerUtility. 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 Session Bean.

**ProjectWorkerUtilityDelegate**

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 ProjectWorkerUtility, and delegating any calls to the bean.

**ProjectWorkerUtilitySessionBean**

This is a Stateless SessionBean that is used to provide business services to manage Projects within the Time Tracker Application. It contains the same methods as ProjectWorkerUtility, and delegates to an instance of ProjectWorkerUtility.
In version 3.2, the internal impl field is instantiated in the ejbCreate() method, which will read configuration as necessary. If an operation throws an exception, sessionContext.setRollbackOnly() will be called before rethrowing the exception.

## 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 Utility 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 Utility and DAO interfaces (and their implementations).

**DuplicateEntityException**
This exception is thrown when there is an attempt to associate an entry with a Project, and it is found that the entry was already associated with the Project.

**InvalidCompanyException**
This exception is thrown when there is an attempt to associate an entity with a Project, and the Company IDs of the Project and the Entity do not match.

**ConfigurationException**
Thrown by xxxDelegate classes if there's a problem in configuration.

### 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 Utility classes (ProjectUtlity, ProjectManagerUtility, and ProjectWorkerUtility) are required to be thread safe, and achieves this via the thread safety of the implementations of the DAO Layer, and the FilterFactory Layer.

The DAOLayer (DbProjectDao, DbProjectEntryDao, DbProjectManagerDao, DbProjectWorkerDao) is made thread safe through the use of transactions, and is achieved with the transaction level of READ_COMMITTED.

There are no new thread safety considerations in version 3.2.
ManualSpecificationFactory is not thread safe, but it is used in an inherently thread safe manner by the Session Beans and it does not form part of the public interface of the component, so it has no bearing on the overall thread safety of the component.

## 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

**TypeSafe Enum 1.0** – is used for the EntryType enum.

**JNDI Context Utility 1.0 -** is used by the business delegates to look up the home interface of the session bean.

**Config Manager** – is used by the Delegate classes.

**Object Factory 2.0** – is indirectly used to configure the component.

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

**Search Builder 1.3** – is used to perform the searches.

**Time Tracker Audit 3.1** – is used to perform the optional audit.

**Time Tracker Contact 3.1** – is used to persist and retrieve Project Contact information and Address

**Time Tracker Common 3.1** – is used to keep track of the payment terms, and has the base TimeTrackerBean class.

### 2.3Third Party Components

None

## 3.Installation and Configuration

### 3.1Package Names

com.topcoder.time.tracker.project
com.topcoder.time.tracker.project.db
com.topcoder.time.tracker.project.ejb

### 3.2Configuration Parameters

~~For the non-j2ee portion, the component relies on Object Factory 2.0 for any configuration that is based from a file.~~
Since the Session Beans will rely on the ability to instantiate these classes at run time, this is no longer possible. All configuration of these classes will be handled through the Session Beans; see the configuration parameters below for details.

The following properties may be specified to the Delegate classes 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 |

For SessionBeans:

| | | | |
|---|---|---|---|
| of_namespace | The namespace used for ConfigManagerSpecificationFactory to use with ObjectFactory | Required | A valid object factory namespace |
| impl_key | The key for a object to get from ObjectFactory; should refer to an appropriate type for the impl field of a particular SessionBean | Required | An object factory key referring to a proper xxxUtility object |

### 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.

### 4.3Demo

We will assume here that everything is configured properly. Try/catch clauses have been removed to enhance clarity. Since this component already has the required Delegate - Session Bean pattern in place, the only changes in usage are in configuration, as covered above.

```
// Create a ProjectUtility Delegate
ProjectUtility projectUtil = new ProjectUtilityDelegate("applicationNamespace.project");

// Create a new Project.
Project project = new Project();
project.setCreationUser("TCUSER");
project.setModificationUser("TCUSER");
project.setName("Time Tracker Project");
project.setDescription("Topcoder Component Competition");
project.setCompanyId(1);
project.setSalesTax("6.5");
project.setContact(projectManagerContact);
project.setAddress(projectManagerAddress);
project.setTerms(paymentTerms);

// Store it in persistence, with auditing
projectUtil.addProject(project, true);

// Search for a Project that started in a specified date range, and whose manager has a userId of
50.
ProjectFilterFactory factory = projectUtil.getFilterFactory();
Filter filter1 = factory.createContainsProjectManagerFilter(50);
Filter filter2 = factory.createStartDateFilter(startDate, endDate);
Filter andFilter = new AndFilter(filter1, filter2);

Project[] searchResults = projectUtil.searchProjects(andFilter);

// Associate a Time Entry with the Project (no auditing)
projectUtil.addEntryToProject(project.getId(), entry.getId(), entryType.TIME_ENTRY, false);

// Add a new ProjectManager to a project.

// First we need a ProjectManagerUtility Delegate
ProjectManagerUtility projectManagerUtil = new
ProjectManagerUtilityDelegate("applicationNamespace.project.manager");

// Create and define a new ProjectManager
ProjectManager projectManager = new ProjectManager();
projectManager.setCreationUser("TCUSER");
projectManager.setModificationuser("TCUSER");
projectManager.setUserId(osokId);
projectManager.setProjectId(project.getId());

// Store the defined ProjectManager
projectManagerUtil.create(projectManager);

// Double the pay rate of all ProjectWorkers that are working on Time Tracker Project

// First we need a ProjectWorkerUtilityDelegate
ProjectWorkerUtility projectWorkerUtil = new
ProjectWorkerUtilityDelegate("applicationNamespace.project.worker");

// Retrieve the filter factory and search for all workers assocaited with time tracker project.
ProjectWorkerFilterFactory filterFactory = projectWorkerUtil.getProjectWorkerFilterFactory();
Filter criterion = filterFactory.createProjectIdFilter(timeTrackerProjectId);

// Perform the search
ProjectWorker[] timeTrackerWorkers = projectWorkerUtil.search(criterion);

// Double the pay rate of the workers
for (int x = 0; x < timeTrackerWorkers.length; x++) {
	timeTrackerWorkers[x].setPayRate(timeTrackerWorkers[x].getPayRate() * 2);
}

// Perform a batch update on the workers, with audit.
projectWorkerUtil.update(timeTrackerWorkers);
```

## 5.Future Enhancements

Provide implementations for different RDBMS.