Time Tracker Company v3.2 Component Specification

1. Design

Overview

The Time Tracker Company custom component is part of the Time Tracker application. It provides an abstraction of company accounts in the system. This component handles the persistence and other business logic required by the application.

This component was part of Time Tracker User v2.0. The new version breaks the existing User component into many smaller components.

Changes in this version:

- This component only handles the persistence and business logic of company accounts.
- Time Tracker Contact is to be used to retrieve and persist contact and address information of a company
- There is no requirement on encryption, thus the password will be stored in persistence as plain text

New feature in version 3.1:

- Audit requirement. All methods that perform modification of data will have a boolean parameter to indicate whether the action is to be audited. Time Tracker Audit component is used to store and retrieve the audit data.

New features in version 3.2:

- Introduces a session bean to facilitate transactional control. This was a major issue with the 3.1 version. As a result, the DAO implementation documentation as well as the relevant SDs are modified to remove all manual transaction control.
- The CompanyManager will become a delegate to this bean. As such, the manager will be split into an interface with this name, and a delegate implementation for the EJB integration. This interface thus removes the getCompanyDAO method is it loses its relevance and appropriateness here

The main persistence logic of company accounts is provided by the class DbCompanyDAO which provides Create, Retrieve, Update, Delete, Enumerate and Search methods for company accounts.

The actual Time Tracker company entity is modeled as data beans that follow the JavaBean naming convention. They are also Serializable for easy persistence and network transfer.

Approach

Time Tracker User v2.0 has provided a relatively good design to accommodate its requirement. Since the requirements of Time Tracker Company v3.1 overlaps very much with the requirements of Time Tracker User v2.0, the new design is made based on the old one and makes minimal amount of changes to the existing design to accommodate the changes in the new requirements.

Changes introduces in existing design

- The public API of the new design is very similar to the existing one with some minor changes:
    - Method create, update, delete (and the batch version) of CompanyDAO interface now has additional doAudit parameter to indicate if the action is to be audited. For delete operation, another parameter is added which username to indicate the user who performed the deletion
    - A new class CompanyManager is added to be used by other sub components of Time Tracker. This class allows the implementation of CompanyDAO to be pluggable. Version 3.2 changes this to an interface as to accommodate the local delegate.
- Major changes occurred in class DbCompanyDAO in which it now utilizes ContactManager and AddressManager to store and retrieve contact and address information of a company. In addition, it also utilizes AuditManager to store the audit information.
- Search Builder v1.3 is now used, instead of Search Builder v1.2. SearchBundle is used in DbCompanyDAO to perform actual search. The context of the SearchBundle spans multiple tables: company, contact, contact_relation, address, address_relation, and state_name. This breaks the boundary of Time Tracker Contact component. However, in the current architecture, it seems that this is the best and simplest way to perform the search.
- Atomicity of Transaction is no longer guaranteed by DbCompanyDAO. This is due to ContactManager, AddressManager and AuditManager create their own connections (and thus transactions) to perform their operations. Thus, each operation, eg: create, spans multiple transactions. Therefore, no atomicity is guaranteed. In Version 3.2, transactions are now guaranteed thanks to the use of session beans.
- Manual rollback. Since some operations such as create, update, delete spans multiple transactions, then manual rollback needs to be performed. See Section 1.3 for the details of implementation of each operation augmented with manual rollback. Version 3.2 now replaces this with rollbacks occurring in the session bean using the session context.

Bug fixes

The new design fixes bugs that are found in existing design, including (but not limited to):

- In Company bean, passcode should be passCode
- Add IllegalArgumentException if id<=0 in retrieveCompany and retrieveCompanies
- CompanySearchBuilder should use SEARCH_CONTACT_PHONE instead of SEARCH_CONTACT_PHONE_NUMBER
- In CompanySearchBuilder, since the date is inclusive, the use of GreaterThanFilter and LessThanFilter which should be GreaterThanOrEqualToFilter and LessThanOrEqualToFilter respectively. EqualsFilter should be EqualToFilter.

Design Issues

Id Generation and separate DAOs:

Since in this component, the company and address persistence are handled by other component (Time Tracker Contact), thus there is no need that the same ID Generator is used for Time Tracker Contact and Time Tracker Company.

Updating the Modification Date and User:

Time Tracker Company is composed of different entities. It contains both a Contact and an Address. Each entity has its own Modification Date and User. When a modification is made to either the containing entity (which is Company), or one of the composing entities (Contact or Address), the question remains as to which table's modification_date and modification_user columns need to be updated. This is resolved by adding an additional class variable *changed* to the TimeTrackerBean. This allows the updating DAO to detect where changes have been made and update the columns according to a set of rules. Those rules are outlined in the Algorithm section.

Transactional Control:

The DbCompanyDAO utilizes other managers to perform tasks, such as auditing. To ensure transactional control, this version uses EJBs to enforce them. Since there is no need for remote computing, only a local session bean is used.

EJBs and ConfigManager:

The EJB specification stipulates that File I/O is not allowed during the execution of the bean. At first glance this might mean that the use of the Config Manager, and by extension Object Factory, DB Connection Factory, and ID generator, could not be allowedbecause it performs property retrieval and storing using files. However, the restriction is only placed on the bean's lifetime, not the ConfigManager's. Therefore, as long as the Config Manager does not perform I/O itself during the execution of the bean, or to be more accurately, that the thread performing a bean operation does not cause the ConfigManager to perform I/O, the use of Config Manager to hold an in-memory library of properties is acceptable. Therefore, this design makes extensive use of Config Manager, Object Factory, DB Connection Factory, and ID generator, again, with the stipulation that the Config Manager implementation does not perform I/O when this component is retrieving properties.

Serialization and Filter:

The Filter hierarchy in Search Builder is currently not serializable, which means it currently cannot be used. The PM is currently researching solutions to this.

Non-atomic operations and transactional control

The requirement calls for keeping the non-atomic batch operations while incorporating them into the container-managed transaction demarcation. Because non-atomic operations in their v3.1 version require that they be able to roll back individual entities, this was simply not possible. The suggestion was to use with them a transaction level of ""NotSupported" and simply let them control themselves manually. This is done by providing the EJBs and DAOs with batch methods that are either atomic or not, and the delegate will pick the correct one based on the atomic flag. The non-atomic batch operations will simply perform their operations manually, by commencing a transaction for each entity to process (including its audit, if so desired), then either committing or rolling back each one separately.

As such, in the Deployment Descriptor, the EJB will declare all non-atomic batch methods as being of transaction level "NotSupported".

1.1 Design Patterns

Strategy is used by the CompanyManager by utilizing a pluggable system of delegating to the CompanyDAO

DataAccessObject pattern is used by CompanyDAO

Proxy pattern is used by CompanyDAOSynchronizedWrapper to provide synchronized access to CompanyDAO implementation.

Business Delegate pattern is used by manager implementation so the user is decoupled from the intricacies of obtaining and calling the session EJBs.

1.2 Industry Standards

JDBC, SQL, EJB 2.1

1.3 Required Algorithms

1.3.1 Modification User and Date

The following procedure is equivalent to the one applied in Time Tracker User 2.0. When a DAO is performing an update on a Time Tracker entity, the following procedure may be followed:

The entity managed in this component is Company entity which is a composite entity (it contains other entities), then:

- Update the table corresponding to the main entity if its *changed* variable is *true* OR any of its composing entities *changed* variable is *true.*
- The composing entities do not need to be updated unless their own *changed* variable is true also.

Example:

```
Company company = dbCompanyDAO.retrieveCompany(id);

// We will simply change a direct attribute of the user
company.setPassword("newPassword");


// This should result in an update to only the company table.
// The address or contact tables are not updated.  Because those entities
// were not modified.
dbCompanyDAO.updateCompany(company, "adminUser", true);
```

Example #2:

```
Company company = dbCompanyDAO.retrieveCompany(id);

// If we modify only one of the composing entities of the Company.
Address addy = company.getAddress();
addy.setLine1("10737 SandBox Ave.");
addy.setLine2("Palmer Hills");

// attempt to update the company will result in an update both to the
// company table and the address table.  The contact table is not
// modified because it was not modified.  Note that company is
// still modified because it is the containing entity of the address.
```

```
dbCompanyDAO.updateCompany(company, “adminUser”, true);
```

Example #3:

```
Company company = dbCompanyDAO.retrieveCompany(id);

// If we modify both of the composing entities of the Company.
Address addy = company.getAddress();
addy.setLine1(“10737 SandBox Ave.”);
addy.setLine2(“Palmer Hills”);

Contact contact = company.getContact();
contact.setPhoneNumber(“123-3456”);

// attempt to update the company will result in an update both to the
// company table, the address table and the contact table, because
// all entities are changed.
dbCompanyDAO.updateCompany(company, “adminUser”, true);
```

1.3.2. Audit Header creation

To create a AuditHeader object, do the following:

- Create new instance of AuditHeader through the non-argument constructor
- Update its properties
  The short descriptions of each field in AuditHeader:
  entityId : company.getId()
  tableName : company
  companyId : company.getId()
  actionType : either INSERT, DELETE or UPDATE depending of the action type
  applicationArea : TT_COMPANY
  resourceId : company.getId()
  creationUser : username method argument

- Create AuditDetail object for each column involved in SQL Statement
- Create an array containing list of AuditDetail
- Update the detail of AuditHeader

1.3.3. Creation of new company

Following SQL can be used to create new tuple:

```
INSERT INTO company (company_id, name, passcode, creation_date, creation_user,
modification_date, modification_user) VALUES (?,?,?,?,?,?,?)
```

The mapping is as follow:

company_id: company.getId()
creation_date and modification date: the current date time
modification_user and creation_user: the username method argument

Implementation:

```
Execute the SQL statement above
Get contact and address from Company
contactManager.create(contact, doAudit);
addressManager.create(address, doAudit);
contactManager.associate(contact, ContactType.TT_COMPANY, company.getId(),
doAudit);
```

```
addressManager.associate(address, AddressType.TT_COMPANY, company.getId(),
doAudit);

If audit is true then
       Create appropriate auditHeader
       auditHeader=auditManager.createAuditRecord(auditHeader);
```

If error occurs, throw the exception wrapped in CompanyDAOException

1.3.4. Retrieve Company

Following SQL can be used:

```
SELECT name, passcode, creation_date, creation_user, modification_date,
modification_user FROM company WHERE company_id = ?
```

Set the ? to companyId method argument

Impl:
Execute the SQL statement
Create a contact search criterion based on companyId and entity type as ContactType.TT_COMPANY.
Use contactManager.search(criterion) to retrieve the corresponding Contact object
Create an address search criterion based on companyId and entity type as AddressType.TT_COMPANY
Use addressManager.search(criterion) to retrieve the corresponding Address object
Create a Company bean and set the properties correspondingly.
Assign the Contact and Address to Company bean.

If error occurs, throw the exception wrapped in CompanyDAOException

1.3.5. Update Company

Following SQL can be used:

```
UPDATE company SET name=?, passcode=?, modification_date=?, modification_user=?
```

```
Set:
```

modification_user: the username method argument
modification_date: the current date time

Impl:

```
Save the old value of contact and address:
       oldCompany=retrieveCompany(company.getId());
Execute the SQL statement above
Get contact and address from Company
Call contactManager.update(contact,user,doAudit);
Call addressManager.update(address,user,doAudit);
If audit is true then
Create appropriate auditHeader
auditHeader=auditManager.createAuditRecord(auditHeader);

If error occurs on above operations, throw the exception wrapped in
CompanyDAOException
```

1.3.6. Delete company

Following SQL can be used:

```
DELETE FROM company WHERE company_id=?
```

```
Impl:
```

```
Get contact and address from Company
Call contactManager.deassociate(contact, ContactType.TT_COMPANY, company.getId(),
doAudit);

Call addressManager.deassociate(address, AddressType.TT_COMPANY, company.getId(),
doAudit);

Execute the SQL statement above
Call contactManager.delete(contact, doAudit,user);
Call addressManager.delete(address, doAudit,user);
If audit is true
Create appropriate auditHeader
auditHeader=auditManager.createAuditRecord(auditHeader);

If exception occurs, throw the exception wrapped in CompanyDAOException
```

1.3.7. List companies

Following SQL can be used:

```
SELECT company_id, name, passcode, creation_date, creation_user, modification_date,
modification_user FROM company
```

If error occurs throw the exception wrapped in CompanyDAOException

1.3.8 Batch company creation

```
If using non atomic batch mode methods, wrap each entity processing in a manual
transaction. Any single failure will result in a non-BatchCompanyDAOException
CompanyDAOException.
       Process each company one by one in the same manner as createCompany
       For audit, save the auditHeader for each company(this is needed for
rollback)

Else If atomicBatchMode is false then
       Process each company one by one in the same manner as createCompany, each
company will be processed independently
       The Company where an error occurred is recorded in a list for constructing
the BatchCompanyDAOException that is thrown.
```

1.3.9. Batch company retrieval

The companies are processed in similar manner to retrieveCompany. Developers might want to create SQL statement to retrieve multiple records:

```
SELECT company_id, name, passcode, creation_date, creation_user, modification_date,
modification_user FROM company WHERE


company_id = ? OR
company_id= ? OR ….
```

1.3.10. Batch company update

```
If atomicBatchMode is true then
If using non atomic batch mode methods, wrap each entity processing in a manual
transaction. Any single failure will result in a non-BatchCompanyDAOException
CompanyDAOException.
       Process each company one by one in the same manner as updateCompany
       For audit, save the auditHeader for each company (this is needed for
rollback)
       Save the oldCompany for each company (this is needed for rollback)

Else If atomicBatchMode is false then
```

```
        Process each company one by one in the same manner as updateCompany, each
company will be processed in a single transaction
        The Company where an error occurred is recorded in a list for constructing
the BatchCompanyDAOException that is thrown.
```

1.3.11. Batch company delete

```
If using non atomic batch mode methods, wrap each entity processing in a manual
transaction. Any single failure will result in a non-BatchCompanyDAOException
CompanyDAOException.
        Process each company one by one in the same manner as deleteCompany
        For audit, save the auditHeader for each company (this is needed for
rollback)

Else If atomicBatchMode is false then
        Process each company one by one in the same manner as deleteCompany, each
company will be processed in a single transaction
        The Company where an error occurred is recorded in a list for constructing
the BatchCompanyDAOException that is thrown.
```

1.4 Component Class Overview

[com.topcoder.timetracker.company]

Company

This bean represents a Company within the context of the Time Tracker component. It holds the different attributes of the company such as the company name, address and contact information. The Company passcode is also stored within in plain text form.

CompanyDAO

This interface defines the necessary methods that a Company DAO should support. Create, Retrieve, Update, Delete and Enumerate (CRUDE) methods, and their respective batch-mode equivalents are specified. There is also a search method that utilizes Filter classes from the Search Builder 1.3 component.

CompanySearchBuilder

This is a convenience class that may be used to build filters for performing searches in the CompanyDAO. Users may call the different methods to set the criteria for the filter and finally retrieve the filter for use via the buildFilter() method.

CompanyManager

This interface defines the contract for the complete management of a company. It provides single and batch CRUD operations. It has one implementation in this design: LocalCompanyManagerDelegate.

[com.topcoder.timetracker.company.implementation]

DbCompanyDAO

Database implementation of the CompanyDAO interface. It is capable of persisting and retrieving Time Tracker Company information from the database. If used in the context of a session bean, the DBConnectionFactory should be configured with a JNDI connection provider, so the Datasource is obtained from the application server.

CompanyDAOSynchronizedWrapper

This class is a synchronized wrapper for an implementation of CompanyDAO interface. Please note that this class marks all of the methods as synchronized.

[com.topcoder.timetracker.company.ejb]

CompanyHomeLocal

The local interface of the Company EJB

CompanyLocal

The local component interface of the Company EJB, which provides access to the persistent store for companies managed by the application.

LocalCompanyManagerDelegate

Implements the CompanyManager interface to provide management of the Company objects through the use of a local session EBJ. It will obtain the handle to the bean's local interface and will simply delegate all calls to it. It implements all methods.

CompanyBean

The session EJB that handles the actual manager requests. It simply delegates all operations to the CompanyDAO it obtains from the ObjectFactory.

1.5 Component Exception Definitions

[com.topcoder.timetracker.company]

CompanyDAOException

This exception is thrown by the CompanyDAO when a problem occurs while accessing the data store. All layers throw this exception: CompanyManager and implementations, EJBs, and the CompanyDAO and implementations.

BatchCompanyDAOException

This exception is thrown by the CompanyDAO when a problem occurs while accessing the data store in non-atomic batch mode methods. It contains an array of causes and problemCompanies, so that the calling application can keep track of which beans a problem occurred in and their respective causes. All layers throw this exception: CompanyManager and implementations, EJBs, and the CompanyDAO and implementations.

CompanyNotFoundException

This exception is thrown if the involved Company was not found in the datastore during a DAO operation that required it to be present. All layers throw this exception: CompanyManager and implementations, EJBs, and the CompanyDAO and implementations.

[com.topcoder.timetracker.company.ejb]

InstantiationException

This exception signals an error during the instantiation of the LocalCompanyManagerDelegate. This error could be due to the provided namespace being not recognized by ConfigManager, or the required jndi property not being available.

1.6 Thread Safety

This component is almost completely thread-safe. Each thread is expected to work on a separate instance of the bean. A single instance of a bean should not be read/modified concurrently by multiple threads. When dealing with the EJB layer, this is ensured since there is only one thread per session bean. The other implementations are immutable and thread-safe, and rely on thread-safe components.

The EJB container also provides transactional control, which was not present properly in 3.1 version.

Note that the CompanyDAO's thread-safety is really dependent on the underlying CompanyDAO implementation.

2. Environment Requirements

2.1 Environment

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

2.2 TopCoder Software Components

- **Configuration Manager 2.1.5** is used to configure the component.
- **ID Generator 3.0** is used to generate unique long ids for the Time Tracker data objects that are created in the persistence layer.
- **DB Connection Factory 1.0** is used to configure and create database connections, which includes the ability to provide Datasources via JNDI.
- **Base Exception 1.0** is used as base class for component's exceptions
- **Search Builder 1.3** is used to provide Search filtering functionality to the database DAO implementation.
- **Time Tracker Common v3.1** is used to provide the TimeTrackerBean class that is the superclass of Company bean
- **Time Tracker Audit v3.1** is used to store the audit information
- **Time Tracker Contact v3.1** is used to store and retrieve the contact and address information
- **Object Factory v2.0.1** is used to obtain object instances

NOTE: The default location for TopCoder Software component jars is../lib/tcs/COMPONENT_NAME/COMPONENT_VERSION relative to the component installation. Setting the tcs_libdir property in topcoder_global.properties will overwrite this default location.

2.3 Third Party Components

- There are no direct 3rd party dependencies. See the 3rd party component dependencies of the components that this component is based on for indirect dependencies.

NOTE: The default location for 3rd party packages is ../lib relative to this component installation. Setting the ext_libdir property in topcoder_global.properties will overwrite this default location.

3. Installation and Configuration

3.1 Package Name

com.topcoder.timetracker.company
com.topcoder.timetracker.company.implementation
com.topcoder.timetracker.company.ejb

3.2 Configuration Parameters

3.2.1 LocalCompanyManagerDelegate

Parameter	Description	Details
JndiReference	JNDI reference to the local company EJB. Required	Example: "java:comp/env/ejb/CompanyLocal"

3.3 Dependencies Configuration

The dependency components should be configured according to their documentation.

The EJB deployment descriptor must have the Datasource configured for the DBConnection Factory. Also, it must include the following two environment entries for the CompanyBean:

Parameter	Description	Details
SpecificationNamespace	Namespace to use with the ConfigManagerSpecificationFactory Required	Example: "com.topcoder.specification"
CompanyDAOKey	Key to the CompanyDAO instance to pass to object factory Required.	"companyDAOKey"

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

Follow the "Demo".

4.3 Demo

```
<property name="TimeTrackerCompanyDAO">
```

```
    <property name="type">
        <value>com.topcoder.timetracker.company.DbCompanyDAO</value>
    </property>
    <property name="params">

       <!-- Parameter corresponds to connection factory.  Should be specified
             as another Object Factory specification. See TimeTrackerConnectionFactory. -->
        <property name="param1">
             <property name="type">
                 <value>com.topcoder.db.connectionfactory.DBConnectionFactory</value>
             </property>
             <property name="name">
                 <value>TimeTrackerConnectionFactory</value>
             </property>
        </property>


        <!-- Parameter corresponds to Connection Name. -->
        <property name="param2">
             <property name="type">
                  <value>java.lang.String</value>
             </property>
            <property name="value">
                  <value>mySQL Connection</value>
             </property>
         </property>



        <!-- Parameter corresponds to Id Generator Name. -->
         <property name="param3">
             <property name="type">
                  <value>java.lang.String</value>
             </property>
            <property name="value">
                  <value>idgeneratorname</value>
             </property>
         </property>
    </property>
</property>

<!-- Property for specifying the Connection Factory that is used by the Company DAO. -->
<property name="TimeTrackerConnectionFactory">
     <property name="type">
         <value>com.topcoder.db.connectionfactory.DBConnectionFactoryImpl</value>
     </property>
     <property name="params">
         <property name="param1">
             <property name="type">
                  <value>java.lang.String</value>
             </property>
            <property name="value">
                  <value>com.topcoder.timetracker.confignamespace</value>
             </property>
         </property>
     </property>
</property>
```

Other configuration items are at developer's discretion, but will include the configuration of Object Factory, and the insertion of the environmental entries in the deployment description described in CS 3.3. Note that the DB Connection Factory must provide connections from a Datasource from JNDI that must be available in the deployment descriptor.

```
// Try-catch clauses have been removed for clarity.

// 1. A simple demonstration of Time Tracker Company management and searching.
CompanyManager manager = new LocalCompanyManagerDelegate(namespace);

Company companyToCreate = new Company();

companyToCreate.setCompanyName("new Company");
companyToCreate.setPassCode("pwd");


Address addy = new Address();
addy.setLine1("Palm Street");
addy.setLine2("Maple Drive");
```

```
addy.setCity(“Florida City”);
addy.setState(floridaState);
addy.setZipCode(“612554”);


Contact contact = new Contact();
contact.setFirstName(“Mr.”);
contact.setLastName(“User”);
contact.setPhoneNumber(“555-5555”);
contact.setEmail(“user@user.com”);



companyToCreate.setContact(contact);
companyToCreate.setAddress(addy);


// Create new company and perform audit
manager.createCompany(companyToCreate, “adminUser”,true);

//companyToCreate should now have an assigned id and modification date,user &
creation date,user assigned.
companyToCreate.getId();


// Get all companies in the datastore
Company[] companyList=manager.listCompanies()

// Delete a company from the datastore
// Perform audit, the deletion is done by a user with username “admin”
manager.deleteCompany(companyToDelete, true, “admin”);

// Update a company, no audit
companyToCreate.setPassword(“newPassword”);
manager.updateCompany(companyToCreate, “admin”, false);

// Retrieve a company with id = 10
Company company10 = manager.retrieveCompany(10);

// Delete companies: companyToCreate, secondCompany, ThirdCompany
// Use batch mode deletion. Perform audit. The deletion is done by a user
// with username “admin”. The deletion will be atomic.
Company[] companiesToDelete = {companyToCreate, secondCompany, ThirdCompany};
manager.deleteCompanies(companiesToDelete,true, true,“admin”);

// Retrieve companies with id = 1,2, 5
Long[] ids = {1,2,5};
Company[] users=manager.retrieveCompanies(ids);
companies[0].getId() → equal to 1
companies[1].getId() → equal to 2
companies[2].getId() → equal to 5

// Create 2 new companies
Company c1 = new Company();
Company c2 = new Company();
// Set the properties of c1 and c2
// Add them to datastore with batch mode and no audit, the batch mode is not
// atomic
Company[] companiesToAdd = {c1,c2};
companiesToAdd=manager.createCompanies(companiesToAdd,”admin”,false,false);



// Search for all companies with whose name contains “Jo” that is in New York.
// Similar approach can be used to search for other criteria

CompanySearchBuilder builder = new CompanySearchBuilder();
builder.hasCompanyName(“Jo”);
builder.hasCity(“New York”);
```

```
Filter searchFilter = builder.buildFilter();

// Now the manager can be used to search:
Company[] results = manager.search(searchFilter);

// Search companies who either satisfy the existing criteria or was created by a
// user with username = “babut”

CompanySearchBuilder builder2 = new CompanySearchBuilder();
builder2.createdByCompany(“babut”);
searchFilter2 = builder2.buildFilter();

Filter combinedFilter = new OrFilter(searchFilter, searchFilter2);

// Search the manager with the combined criteria
Company[] results2 = manager.search(combinedFilter);



// All companies in New York have relocated to Old York
for (int x = 0; x < results.length; x++) {
     results[x].setCity(“Old York”);
}

// Perform a batch update on those companies.
manager.updateCompanies(results, “admin”,true, false); // no audit

// The builder can be reset to perform additional searches.
builder.reset();


// The Object Factory 2.0 may be used to create DbCompanyDAO
ObjectFactory daoFactory = new ObjectFactory(new
ConfigManagerSpecificationFactory(“com.topcoder.timetracker.company”);
CompanyDAO companyDAO = (CompanyDAO)
     daoFactory.createObject(“TimeTrackerCompanyDAO”);

companyDAO.createCompany(company,”admin”,true);

companyDAO.deleteCompany(company,”admin”,true);
```

5. Future Enhancements

None.