Client Project Management 1.0 Component Specification

1. Design

This component provides a framework for managing TopCoder clients, projects, and companies. This component provides simple API calls to manipulate and retrieve clients, projects, statuses, and companies, using the Client and Project Entities and DAO component to provide the entities and persistence implementation.

This component provides several convenient methods to manage Client, Project and Company entities as well as their status entities; it acts more like a wrapper upon the underlying DAO interfaces (different DAO implementation can be plugged into this component). Besides regular CRUD operations on entities, entity searching functionalities are implemented too, user can construct a general filter (defined in Search Builder component) to search client/project/company entities with any proper criteria.

1.1 Design Patterns

Strategy Pattern – Managers hold handlers of DAO interfaces, different DAO implementations can be plugged in. This component only plays part of this pattern; other parts are completed by entity DAO component.

Facade pattern – This pattern is used in DAOXXXManager classes to provide access to the component functions. They utilize the DAO implementations.

1.2 Industry Standards

None.

1.3 Required Algorithms

There is no special algorithms used in this component, all dirty work are done by underlying DAO implementations. The steps of how to implement each method is detailed in the method doc. Here, we discuss some points of this component.

1.3.1 Validation of entities

In create and update operations, the entity to be persisted should be validated first, if validation fails, the operation is terminated and exceptions are thrown. Some rules are defined to do validation, in this version, only some of the properties of the entities have valid values ranges imposed on them. They are described below.

Company:

Property name	Valid value
passcode	Non-null and non-empty string
isDeleted	False

Project:

Property name	Valid value
salesTax	Should be greater than zero
description	Non-null string
name	Non-null and non-empty string
isDeleted	False

Client:

Property name	Valid value
startDate, endDate	startDate < endDate
name	Non-null and non-empty string
isDeleted	False

ClientStatus and ProjectStatus:

Property name	Valid value
name	Non-null and non-empty string
description	Non-null and non-empty string
isDeleted	False

1.3.2 *Logging*

Logging should be done in all methods of three managers, method call, method exit, and errors should be logged appropriately. Logging is not required for constructors.

Logging should be performed like this:

All method call and method exit should be logged at DEBUG level. Logging for method call includes the details of input parameters; logging for method exit includes the details of returned values. The message templates are: [Calling and entering method <*className.methodName*>; Parameters: {ParaName: <*parameterName*>; ParaValue: <*parameterValue*>}].
[Exiting method <*className.methodName*>; Return value: <*returnValue*>]

All error should be logged at WARN level, including the stack trace. The message template is: [Error occurred in method <*className.methodName*>, caused by <*stacktrace*>].

Note: Don't put exit logging into a "finally" block.

Any other logging is at the developer's discretion. The developer is also free to improve on the above template. Such logging should be at the DEBUG level.

1.4 Component Class Overview

com.topcoder.clients.manager
ClientManager<interface>:
This interface defines functionality for managing client information, including CRUD functionality for clients and statuses, searching for clients by name and a provided search filter, and getting clients by status, as well as directly updating a client's status and code name.

ProjectManager<interface>:
This interface defines functionality for managing project information, including CRUD functionality for projects and statuses, searching for projects by name and a provided search filter, and getting projects by status, as well as getting all projects for a provided client.

CompanyManager<interface>:
This interface defines functionality for managing company information, including CRUD functionality for companies, as well as retrieving clients and projects for a provided company.

com.topcoder.clients.manager.dao
AbstractDAOManager:
This is the base class for all manager classes in this component; it provides an IDGenerator used by subclasses to generate ID for new entities, and a logger that can be used to log activities and error in children. It stores a ConfigurationObject parsed by Configuration Persistence Manager if configuration file is used, so subclasses can use it to do further configuration. An ObjectFactory is created and can be used by subclasses to create DAO instances.

DAOClientManager:
This class is an implementation of ClientManager interface. It provides convenient methods to manage Client and ClientStatus entities, all operations will eventually delegate to the underlying ClientDAO and ClientStatusDAO appropriately. It uses IDGenerator to generate new IDs for entities, and has a logger used to log method entry/exit and all exceptions.
It can be configured via ConfigurationObject directly, as well as configuration file compatible with the format defined in Configuration Persistence component.

DAOProjectManager:
This class is an implementation of ProjectManager interface. It provides convenient methods to manage Project and ProjectStatus entities, all operations will eventually delegate to the underlying ProjectDAO, ClientDAO and ProjectStatusDAO appropriately. It uses IDGenerator to generate new IDs for entities, and has a logger used to log method entry/exit and all exceptions.
It can be configured via ConfigurationObject directly, as well as configuration file compatible with the format defined in Configuration Persistence component.

DAOCompanyManager:

This class is an implementation of CompanyManager interface. It provides convenient methods to manage Company entities, all operations will eventually delegate to the underlying CompanyDAO appropriately. It uses IDGenerator to generate new IDs for entities, and has a logger used to log method entry/exit and all exceptions.

It can be configured via ConfigurationObject directly, as well as configuration file compatible with the format defined in Configuration Persistence component.

1.5 Component Exception Definitions

com.topcoder.clients.manager
ClientEntityNotFoundException:
This exception is used to indicate the Client and ClientStatus entity does not exist in persistence when updating/deleting it. It extends from ManagerException.

ClientManagerException:
This exception is used to indicate any error that occurs in ClientManager's business methods, for example, an error thrown by underlying ClientDAO implementation will be caught and wrapped in this exception and re-thrown. It extends from ManagerException.

It has a field that can be used to carry a Client instance when the exception is constructed, user can retrieve the client by related getter, it helps user diagnose the exception.

ManagerConfigurationException:
This exception is used to indicate there is an error when configuring the managers, for example, required configuration value is missing, error occurred in Object Factory. It extends from ManagerException.

ProjectEntityNotFoundException:
This exception is used to indicate the Project and ProjectStatus entity does not exist in persistence when updating/deleting it. It extends from ManagerException.

ProjectManagerException:
This exception is used to indicate any error that occurs in ProjectManager's business methods, for example, an error thrown by underlying ProjectDAO implementation will be caught and wrapped in this exception and re-thrown. It extends from ManagerException.

It has a field that can be used to carry a Project instance when the exception is constructed, user can retrieve the project by related getter, it helps user diagnose the exception.

ManagerException:
This exception is the base exception of all other exceptions in this component, it's not thrown directly by this component, it extends from BaseCriticalException in Base Exception.

CompanyManagerException:
This exception is used to indicate any error that occurs in CompanyManager's business methods, for example, an error thrown by underlying CompanyDAO implementation will be caught and wrapped in this exception and re-thrown. It extends from ManagerException.

It has a field that can be used to carry a Company instance when the exception is constructed, user can retrieve the company by related getter, it helps user diagnose the exception.

CompanyEntityNotFoundException:
This exception is used to indicate the Company entity does not exist in persistence when updating/deleting it. It extends from ManagerException.

1.6 Thread Safety

This component is thread-safe. Three managers are thread safe, all fields of them do not change after construction, and implementation of ClientDAO, ClientStatusDAO, ProjectDAO, ProjectStatusDAO and CompanyDAO are expected to be thread-safe, the logger used in logging and the ID generator used to generate ID are also thread safe. So this component is effectively thread-safe.

2. Environment Requirements

2.1 Environment

- Development language: Java 1.5
- Compile target: Java 1.5 and Java 1.6

2.2 TopCoder Software Components

- **Configuration API 1.0**
  - Provides ConfigurationObject that is used by this component.
- **Configuration Persistence 1.0.1**
  - Provides ConfigurationFileManager that can convert configuration in a file to a ConfigurationObject.
- **Base Exception 2.0**
  - Provides BaseCriticalException that is the base exception of all custom exceptions in this component.
- **Logging Wrapper 2.0**
  - Provides the Log that can be used to log activities and errors.
- **Search Builder 1.4**

  - Provides Filter that is used to search entities with in this component
- **ID Generator 3.0**
  - Provides IDGenerator that is used to generate ID for new entities.
- **Client and Project Entities and DAO 1.0**
  - Provides the definitions of all entities used by this component, it also defines DAO contracts and provides default DAO implementations that are used here.
- **Object Factory 2.1**
  - Used to configure managers in this component, provides the capability to create new instances of specified class via configuration.
- **Object Factory Configuration API Plugin 1.0**
  - An adapter between Object Factory and Configuration API components, upgrades Object Factory by adding the ability of creating objects with configuration stored in ConfigurationObject.

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

- None.

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.clients.manager

Contains all exceptions and manager contracts.

com.topcoder.clients.manager.dao

Contains manager implementations

3.2 Configuration Parameters

DAOXXXManager of this component can be constructed via configuration. Configuration parameter name and valid values are described below.

AbstractDAOManager:

Parameter	Description	Required
id_generator_name	Name of the IDGenerator, used by	Yes

	IDGeneratorFactory. Should be non-null/non-empty string.	
logger_name	Name of the logger, used by LogManager. Should be non-empty string. If null is provided, then default logger name(full class name) will be used..	No
object_factory_configruation	A child ConfigurationObject containing the configuration used to create ObjectFactory.	Yes

DAOClientManager

Parameter	Description	Required
client_dao	The key used to create ClientDAO with ObjectFactory. Should be non-null/non-empty string.	Yes
client_status_dao	The key used to create ClientStatusDAO with ObjectFactory. Should be non-null/non-empty string.	Yes

DAOProjectManager

Parameter	Description	Required
client_dao	The key used to create ClientDAO with ObjectFactory. Should be non-null/non-empty string.	Yes
project_dao	The key used to create ProjectDAO with ObjectFactory. Should be non-null/non-empty string.	Yes
project_status_dao	The key used to create ProjectStatusDAO with ObjectFactory. Should be non-null/non-empty string.	Yes

DAOCompanyManager

Parameter	Description	Required
company_dao	The key used to create CompanyDAO with ObjectFactory. Should be non-null/non-empty string.	Yes

3.3 Dependencies Configuration

All dependent components should be properly configured before this component is used. Please consult the documents of the dependent components for configuration details.

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

- Configure all dependencies as described in their respective docs.
- Create DAOXXXManager instances and use them.
- Sample configuration file for DAOClientManager, configuration for other managers are similar

```
<CMConfig>
      <Config
name="com.topcoder.clients.manager.dao.DAOClientManager">

            <!--Child configuration for object factory-->
            <Property name="object_factory_configruation">
                  <!--Configuration for ClientDAO-->
                  <Property name="Default_Client_DAO">
                        <Property name="type">
                        <Value><!--full name of ClientDAO
implementation--></Value>
                        </Property>
                        <Property name="params">
                              <!--Parameter configuration-->
                        </Property>
                  </Property>
                  <!--Configuration for ClientStatusDAO-->
                  <Property name="Default_Client_Status_DAO">
                        <Property name="type">
<Value><!--full name of ClientStatusDAO implementation--></Value>
                        </Property>
                        <Property name="params">
                              <!--Parameter configuration-->
                        </Property>
                  </Property>
            </Property>

<!--Name of IDGenerator, used to create IDGenerator with
IDGeneratorFactory-->
            <Property name="id_generator_name">
                  <Value>client_id_generator</Value>
            </Property>

            <!--Name of the logger, used to create Log with
LogManager-->
```

```
            <Property name="logger_name">
                  <Value>client_logger</Value>
          </Property>

            <!--Key used to create ClientDAO instance with
ObjectFactory-->
            <Property name="client_dao">
                  <Value>Default_Client_DAO</Value>
            </Property>

            <!--Key used to create ClientStatusDAO instance with
ObjectFactory-->
            <Property name="client_status_dao">
                  <Value>Default_Client_Status_DAO</Value>
            </Property>
      </Config>
</CMConfig>
```

4.3 Demo

The usage of this component is very intuitive. We demonstrate the typical APIs below.

```
 //Assume following entities are stored in persistence currently

 //ProjectStatus-1: id=1, name="running", desc="it's running"
 //ProjectStatus-2: id=2, name="stopped", desc="it's stopped"

 //Project-1: id=1, name="project-1", company=Company-1,
                         status=ProjectStatus-2,
                         children={Project-1, Project-3}
                         client=Client-1
//Project-2: id=2, name="project-2", company=Company-1,
                                     status=ProjectStatus-1
                                     client=Client-1
 //Project-3: id=3, name="project-3", company=Company-2,
                                     status=ProjectStatus-1
                                   client=Client-2
 //ClientStatus-1: id=1, name="online", desc="it's online"
 //ClientStatus-2: id=3, name="offline" desc="it's offline"

//Client-1:  id=1, name="client-1",  company=Company-1,
                                        status=ClientStatus-2
//Client-2:  id=2, name="client-2",  company=Company-1,
                                        status=ClientStatus-2
 //Client-3:  id=3, name="client-3",  company=Company=2,
                                        status=ClientStatus-1
 //Company-1: id=1, name="company-1"

//Company-2: id=2, name="company-2"
```

Demonstrates the usage of DAOClientManager, all CRUD operations and searching methods would be shown.

```
  // create an instance of DAOClientManager by default
  DAOClientManager manager = new DAOClientManager();
  // create an instance of DAOClientManager with
```

```
ConfigurationObject
        manager = new DAOClientManager(configObj);
        // create an instance of DAOClientManager with configuration
file
        manager = new DAOClientManager(configFile, "DAOClientManager");

        // retrieve a client with ID=1, Client-1 will be returned.
        client = manager.retrieveClient(1);
        // retrieve all clients
        List<Client> clients = manager.retrieveAll();
        //a list containing {Client-1, Client-2, Client-3} should be
returned.

        // search client for specified name
        clients = manager.searchClientsByName("clent-1");
        //a list containing {Client-1} should be returned

        // search clients with Company-1, create a filter for this
        clients = manager.searchClients(filter);
        //a list containing {Client-1, Client-2} should be returned.

        // set new code name to Client-1
        manager.setClientCodeName(client, "new code name");

        //get client status with ID=1, ClientStatus-1 should be
returned
        ClientStatus status = manager.retrieveClientStatus(1);

        //get clients with status=Status-2, Client-1 and Client-2
should be returned
        clients = manager.getClientsForStatus(status);

        // set new status to Client-1
        manager.setClientStatus(client, status);
        // Client-1 should have ClientStatus-1 now.

        // update the client
        client.setSalesTax(100.121);
        manager.updateClient(client);
        //sales tax of Client-1 is updated to 100.121

        //deletes the client
        manager.deleteClient(client);
        //Client-1 should be deleted

        //create a new client
        manager.createClient(newClient);

        //updates the status
        status.setName("pending");
        //name of ClientStatus-1 is changed to "pending"

        //deletes the status
        manager.deleteClientStatus(status);
        //ClientStatus-1 is deleted

        //create a new Client status
        manager.createClientStatus(newStatus);
```

Demonstrates the usage of DAOProjectManager, all CRUD operations and searching methods would be shown. Persistence is set to its initial state.

```
        // create an instance of DAOProjectManager by default
        DAOProjectManager manager = new DAOProjectManager();
        // create an instance of DAOProjectManager with
ConfigurationObject
        manager = new DAOProjectManager(configObj);
        // create an instance of DAOProjectManager with configuration
file
        manager = new DAOProjectManager(configFile, "DAOClientManager");


        // retrieve project with ID=1, Project-1 is returned, all
children are retrieved too
        Project project = manager.retrieveProject(1);
        // retrieve project with ID=1, Project-1 is returned, children
is not retrieved
        project = manager.retrieveProject(1, false);
        // retrieve projects for Client-1, {Project-1, Project-2}
should be returned, all
        // children are retrieved
        List<Project> projects =
manager.retrieveProjectsForClient(Client1);
        // retrieve projects for Client-1, {Project-1, Project-2}
should be returned
        // children are not retrieved
        projects = manager.retrieveProjectsForClient(Client1, false);
        // retrieve all projects, {Project-1, Project-2, Project-3}
should be returned,
        // children are retrieved too
        projects = manager.retrieveAllProjects();
        // retrieve all projects, {Project-1, Project-2, Project-3}
should be returned,
        // children are not retrieved
        projects = manager.retrieveAllProjects(false);
        // search projects with name="project-2", Project-2 should be
returned
        projects = manager.searchProjectsByName("project-2");
        // search projects with company=Company-1, prepare proper
filter.
        // {Project-1, Project-2} should be returned.
        projects = manager.searchProjects(filter);

        // retrieve project status wit ID-1, ProjectStatus-1 should be
returned.
        ProjectStatus status = manager.getProjectStatus(1);
        // get all project status, {ProjectStatus-1, ProjectStatus-2}
should be returned
        statuses = statuses = manager.getAllProjectStatus();
        // get projects with status=ProjectStatus-1, {Project-2,
Project-3} should be returned
        projects = manager.getProjectsForStatus(status);
        //set status of Project-1 to ProjectStatus-1
        project.setStatus(status);

        //update Project-1
        project.setName("nasa");
        manager.updateProject(project);
        //name of Project-1 is changed to "nasa"

        //delete project-1
        manager.deleteProject(project);
        //Project-1 is deleted

        //create a new project
        manager.createProject(newProject);

        //update ProjectStatus-1, set its name to "new status"
        status.setName("new status");
        manager.updateProjectStatus(status);
        //name of ProjectStatus-1 is changed to "new status"

        //delete ProjectStatus-1
        manager.deleteProjectStatus(status);
        //ProjectStatus-1 is deleted

        //create a new ProjectStatus
         manager.createProjectStatus(newStatus);
```

Demonstrates the usage of DAOCompanyManager, all CRUD operations and searching methods would be shown. Persistence is set to its initial state.

```
        // create an instance of DAOCompanyManager by default
        DAOCompanyManager manager = new DAOCompanyManager();
        // create an instance of DAOProjectManager with
ConfigurationObject
        manager = new DAOCompanyManager(configObj);
        // create an instance of DAOProjectManager with configuration
file
        manager = new DAOCompanyManager(configFile, "DAOClientManager");


        // get Company with ID=1, Company-1 should be returned
        Company company = manager.retrieveCompany(1);
        // get all companies, {Company-1, Company-2} should be returned
        List<Company> companies = manager.retrieveAllCompanies();
        // search companies with name="company-1", Company-1 should be
returned
        companies = manager.searchCompaniesByName("company-1");
        // search companies with filter (criteria: name="company-2"),
Company-2 should be returned.
        companies = manager.searchCompanies(filter);

        //get clients for Company-1, {Client-1, Client-2} should be
returned
        List<Client> clients = manager.getClientsForCompany(company);
        //get projects for Company-1, {Project-1, Project-2} should be
```

```
returned
        List<Project> projects = manager.getProjectsForCompany(company);

        //update Company-1, set its name to "dsa"
        company.setName("dsa");
        manager.updateCompany(company);
        //name of Company-1 is changed to "dsa"

        //deletes Company-1
        manager.deleteCompany(company);
        //Company-1 is deleted

        //create a new Company
         manager.createCompany(newCompany);
```

5. Future Enhancements

- Further manager implementations can be added.