[TopCoder]

Client Project Lookup Services 1.0 Component Specification

1. Design

This component provides a wrapper to the Client and Project Management component, exposing the main management functionality as web services. This component also provides client classes that will allow for easy remote access to the service methods. This component addresses the Lookup and Status management services, using authorization implemented in the Client and Project Management Services component. The client, project, and company management services are part of a separate component.

The web service operations perform logging using Logging Wrapper component and authentications and authorizations are performed using JBoss Login Module component.

In addition, a super user (administrator) is added that can perform all the operations without any interactions with UserMappingRetriever.

1.1 Design Patterns

DAO Pattern – this component provided a series of services interfaces and implementations, which follows this pattern.

Strategy pattern – the bean classes created in this design use the managers injected in a strategic manner.

DTO pattern – all the entities used in this design are Data Transfer Objects.

Proxy Pattern – is used by the clients of the implemented web services. The returned interfaces of the services provide the service operations on the remote machine. `XXXServiceClient` client classes actually acts as a Proxy Pattern.

Delegation pattern – the service beans delegates their operations to the configured managers.

1.2 Industry Standards

- EJB 3.0
- JAX-WS 2.0

1.3 Required Algorithms

There are no complicated algorithms in this design.

The sections below are just design considerations used in this design.

1.3.1 Transactions

All access to the database that updates creates, or retrieves from multiple tables happen in a transaction. If an error occurs, the transaction is rolled back gracefully.

Exceptions are configured to handle this:

@ApplicationException(rollback=true) - to notify the EJB container to roll back the transaction.

1.3.2 Logging

1.3.2.1 The Log field can be initialized using:

If a logName is provided (not null or empty String)

```
log = LogManager.getLog( logName )
```

or:

```
log = LogManager.getLog()
```
if no logName is provided

Logging is used in this way:

```
logger.log( Level.DEBUG, "Some internal details" );
```

1.3.2.2 Logging can be turned on or off using:

`setLog(newLog)` method. If `newLog` is null, logging is disabled. To enable logging, a valid value should be provided.

1.3.2.3 Determine the type of the logging:

```
//Determine if logging should be performed:
if( !log ==null)
      // determine the type of logging:
      if (verboseLogging== true)
      //detailed logging actions should be performed
      else
      //standard logging should be performed
else.
//if false: no logging should be performed.
```

1.3.2.4 *There are two types of logging that can be set using:*

setVerboseLogging(verboseLogging:boolean) method:

- ***Standard log* that logs only the following:**

  `if(getVerboseLogging()==false):`
  - Method entry and parameter information (`Level.DEBUG` level);
  - Any exception `(Level.WARNING level)` before the exception is re thrown. The exception name and stack trace should be logged.
- ***Detailed log*** that logs exceptions and other details:

  `if(getVerboseLogging()==true):`
  - Entering the method and timestamp (`Level.DEBUG level`);
  - Method arguments (`Level.DEBUG level`);
  - Time spent in the method (`Level.DEBUG level`);
  - Return value (`Level.DEBUG level`). Any exception `(Level.WARNING level)` before the exception is re thrown. The exception name and stack trace should be logged.
  - Any exception `(Level.WARNING level)` before the exception is re thrown. The exception name and stack trace should be logged.

1.3.3 Initialize the needed dependencies

This section shows the initialization of the needed dependencies for the stateless session's beans. The dependencies should be initialized using the configured resources and expected tokens:

dependency	fileResource	namespaceResource	token
projectManager	projectManagerFile	projectManagerNamespace	project_manager_token
clientManager	clientManagerFile	clientManagerNamespace	client_manager_token
companyManager	company ManagerFile	company ManagerNamespace	company_manager_token
userMapping Retriever	userMappingRetriever File	userMappingRetriever Namespace	user_mapping_retriever_token

Depending of the needed **dependency**, replace the **fileResource**, **namespaceResource** and **token** with the values from the table from above.

```
// create an ConfigurationFileManager from the given file:
ConfigurationFileManager configurationFileManager = new
ConfigurationFileManager(fileResource);

// create an ConfigurationObject using the give namespace:
ConfigurationObject configurationObject =
configurationFileManager.getConfiguration(namespaceResource);
```

```
// create an ConfigurationObjectSpecificationFactory from the created
// configurationObject:
ConfigurationObjectSpecificationFactory configurationObjectSpecificationFactory =
new
ConfigurationObjectSpecificationFactory(configurationObject);

// create an ObjectFactory from the created
// configurationObjectSpecificationFactory:
ObjectFactory objectFactory = new
ObjectFactory(configurationObjectSpecificationFactory);

// get the needed token value from the created configurationObject:
String tokenValue = configurationObject.getPropertyValue(token);

// create the needed dependency using the created objectFactory and the
// retrieved token value:
dependency = (DependencyClass)objectFactory.createObject(tokenValue);
```

1.3.4 *Authorization and authentication*

The services defined in this component relies on JAAS authentication of users to be performed under control of the EJB container, and it relies on the security roles configured for the user in making authorization decisions.
Two security roles will be used: "User" and "Admin".
These need to be declared either in the bean's deployment descriptor or via annotation of the bean class. The annotation to use is:

```
@DeclareRoles({ Roles.USER, Roles.ADMIN })
@RolesAllowed({ Roles.USER, Roles.ADMIN })
```

- *Case 1: user is administrator:*

Means that the user has full privileges and no other authentication should be used and the user is authorized to perform the given operation:

```
// check whether the caller is an administrator
if(sessionContext.isCallerInRole(administratorRole))
// do the rest of the operation
```
else check if the user is client and project user (case 2)

- *Case 2: user is client and project user:*

In some cases, the bean implementation must obtain the user's ID in order to make fine-grained authorization decisions. For this purpose the bean must obtain the Principal object identifying the caller, which it assumes to be of type UserProfilePrincipal (on account of the TopCoder JBoss Login Module being in use). The principal is available from the bean's session context. The complete procedure is as follows:

```
// check whether the caller is an clientAndProjectUserRole role
if(sessionContext.isCallerInRole(clientAndProjectUserRole))
// do the rest of the operation
else throw an exception (case 3)
```

If the user operate on a **company** then precede to a call the namesake method from the configured companyManager.

```
// obtain the user profile principal:
UserProfilePrincipal principal = (UserProfilePrincipal)
sessionContext.getCallerPrincipal();

// obtain the user ID
long userId = principal.getUserId();
```

Now if the user operates on some **project,** check if the user is associated to the given **project**:

```
List<Project> projectsForUserId =
userMappingRetriever.getProjectsForUserId(userId);

// call the namesake method from the configured projectManager:
```

- projectManager return a List<Project>

```
List<Project> projects = projectManager.XXX;
List<Project> validProjects = new ArrayList<Project>();

FOR EACH project in projects
if (projectsForUserId.contains(project)
validProjects.add(project);
END FOR EACH;
return validProjects;
```

- projectManager return a Project

```
Project project = projectManager.XXX;
if (projectsForUserId.contains(project)
return project;
else throw an exception (case 3)
```

Now if the user operates on some **client,** check if the user is associated to the given **client**:

```
List<Client> clientsForUserId =
userMappingRetriever.getClientsForUserId(userId);

// call the namesake method from the configured clientManager:
```

- clientManager return a List<Client>

```
List<Client> clients= clientManager.XXX;
List<Client> validClients = new ArrayList<Client>();

FOR EACH client in clients
if (clientsForUserId.contains(client)
validClients.add(client);
END FOR EACH;
return validClients;
```

- clientManager return a Client

```
Client client = clientManager.XXX;
if (clientsForUserId.contains(client)
return client;
else throw an exception (case 3)
```

- *Case 3: user authentication failed:*

If authentication failed means the user is not authorized to perform this operation and:

```
throw new AuthorizationFailedException(..).
```

1.3.5 Update the user and date properties for the entities

In this design, entity means a subclass of AuditableEntity: ClientStatus and ProjectStatus.

- *Create entity:*

Update Dates: when creating an entity, the entity.createDate and entity.modifyDate should be set to the current date using appropriate setCreateDate() and setModifyDate()methods.

Update Users: when creating an entity, the entity.createUsername and entity.modifyUsername should be set to the current user using appropriate setCreateUsername(**principal**.getName()) and setModifyUsername(**principal**.getName())methods.

- *Other operations over the entity like delete or update:*

Update Dates: the entity.modifyDate should be set to the current date using appropriate setModifyDate() method.

Update Users: the entity.modifyUsername should be set to the current user using appropriate setModifyUsername(**principal**.getName())method.

Note: please note the fact that the entities are not really deleted from the database, only marked as deleted using the is_deleted flag. So the modifyDate and modifyUsername should be updated too.

1.4 Component Class Overview

1.4.1 com.topcoder.clients.webservices

LookupService
This interface represents the LookupService web service endpoint interface.
This interface defines the method available for the LookupService web service: retrieve client, project, company, client status and project status by id, retrieve all clients, projects, companies, client statuses and project statuses, search all clients, projects, companies by name, search all clients, projects, companies by filter, retrieve clients for status, retrieve projects for client, retrieve projects for status, retrieve projects for company, retrieve clients for company.

ClientStatusService
This interface represents the ClientStatusService web service endpoint interface.
This interface defines the method available for the ClientStatusService web service: create, update and delete client status.

ProjectStatusService
This interface represents the ProjectStatusService web service endpoint interface.
This interface defines the method available for the ProjectStatusService web service: create, update and delete project status.

LookupServiceLocal
This interface represents the LookupService local interface of the session bean.
Also it defines a static String variable containing the JNDI name of the local interface.

ClientStatusServiceLocal
This interface represents the ClientStatusService local interface of the session bean.
Also it defines a static String variable containing the JNDI name of the local interface.

ProjectStatusServiceLocal
This interface represents the ProjectStatusService local interface of the session bean.
Also it defines a static String variable containing the JNDI name of the local interface.

LookupServiceRemote
This interface represents the LookupService remote interface of the session bean.
Also it defines a static String variable containing the JNDI name of the remote interface.

ClientStatusServiceRemote
This interface represents the ClientStatusService remote interface of the session bean.
Also it defines a static String variable containing the JNDI name of the remote interface.

ProjectStatusServiceRemote
This interface represents the ProjectStatusService remote interface of the session bean.

Also it defines a static String variable containing the JNDI name of the remote interface.

1.4.2 com.topcoder.clients.webservices.beans

LookupServiceBean
This class is a Stateless Session Bean endpoint realization of LookupService web service interface.
This class has a default no-arg constructor.
It uses CompanyManager, ClientManager and ProjectManager from Client Project Management component by delegating to namesake methods to perform the operations.
This class implements the method available for the LookupService web service: retrieve client, project, company, client status and project status by id, retrieve all clients, projects, companies, client statuses and project statuses, search all clients, projects, companies by name, search all clients, projects, companies by filter, retrieve clients for status, retrieve projects for client, retrieve projects for status, retrieve projects for company, retrieve clients for company.
All the available web service operations perform logging using the Logging Wrapper component.
All the available web service operations performs the authentication and authorization of the users that performs the operations using JBoss Login Module component and then delegates to the Client Project Management component which performs the needed operations.

ClientStatusServiceBean
This class is a Stateless Session Bean endpoint realization of ClientStatusService web service interface.
This class has a default no-arg constructor.
It uses ClientManager from Client Project Management component by delegating to namesake methods to perform the operations.
This class implements the method available for the ClientStatusService web service: create, update and delete client status.
All the available web service operations perform logging using the Logging Wrapper component.
All the available web service operations are under the "User" and "Admin" security roles.

ProjectStatusServiceBean
This class is a Stateless Session Bean endpoint realization of ProjectStatusService web service interface.
This class has a default no-arg constructor.
It uses ProjectManager from Client Project Management component by delegating to namesake methods to perform the operations.
This class implements the method available for the ProjectStatusService web service: create, update and delete project status.
All the available web service operations perform logging using the Logging Wrapper component.
All the available web service operations are under the "User" and "Admin" security roles.

Roles
This enum represents the Roles enum and is a holder for all of the available roles.
Also it defines two variables containing the role names: USER and ADMIN.

1.4.3 com.topcoder.clients.webservices.webserviceclients

LookupServiceClient
This class is a client of LookupService service.
This class has tree constructors that use the wsdlDocumentLocation and service name to create this class instance.
This class has only one method getLookupServicePort() that return the service endpoint interface.

ClientStatusServiceClient
This class is a client of ClientStatusService service.
This class has tree constructors that use the wsdlDocumentLocation and service name to create this class instance.
This class has only one method getClientStatusServicePort() that return the service endpoint interface.

ProjectStatusServiceClient
This class is a client of ProjectStatusService service.
This class has tree constructors that use the wsdlDocumentLocation and service name to create this class instance.
This class has only one method getProjectStatusServicePort() that return the service endpoint interface.

1.5 Component Exception Definitions

1.5.1 com.topcoder.clients.webservices

LookupServiceClientCreationException
This runtime exception signals an issue when the creation of the lookup web service client failed (wraps the MalformedURLException when creating the URL given a String wsdl document location).

ClientStatusServiceClientException
This runtime exception signals an issue when the creation of the client status web service client failed (wraps the MalformedURLException when creating the URL given a String wsdl document location).

ProjectStatusServiceCreationException
This runtime exception signals an issue when the creation of the project status web service client failed (wraps the MalformedURLException when creating the URL given a String wsdl document location).

1.5.2 com.topcoder.clients.webservices

LookupServiceException
This exception is the base exception for all exceptions raised from operations from LookupService (excerpt loading of the configurations).
This exception wraps exceptions raised from persistence or from usage of the J2EE utilities or from the used TopCoder components.

ProjectStatusServiceException
This exception is the base exception for all exceptions raised from operations from ProjectStatusService (excerpt loading of the configurations).
This exception wraps exceptions raised from persistence or from usage of the J2EE utilities or from the used TopCoder components.

ClientStatusServiceException
This exception is the base exception for all exceptions raised from operations from ClientStatusService (excerpt loading of the configurations).
This exception wraps exceptions raised from persistence or from usage of the J2EE utilities or from the used TopCoder components.

1.5.3 com.topcoder.clients.webservices.beans

LookupServiceBeanConfigurationException
This runtime exception signals an issue if the configured value is invalid in LookupServiceBean (in this design, when managers are null, user mapping retriever is null or required resources are null or empty String).
Also it wraps the underlying exceptions when using the configured values.

ProjectStatusServiceBeanConfigurationException
This runtime exception signals an issue if the configured value is invalid in ProjectStatusServiceBean (in this design, when managers are null, user mapping retriever is null or required resources are null or empty String).
Also it wraps the underlying exceptions when using the configured values.

ClientStatusServiceBeanConfigurationException

This runtime exception signals an issue if the configured value is invalid in ClientStatusServiceBean (in this design, when managers are null, user mapping retriever is null or required resources are null or empty String).
Also it wraps the underlying exceptions when using the configured values.

1.5.4 java.lang.

IllegalArgumentException
This system exception is thrown if an argument is invalid (null or long ids <= 0), empty string etc.

IllegalStateException
This system exception is thrown if a needed instance is needed and it has not been initialized yet or it has a null value.

1.5.5 javax.xml.ws

WebServiceException
This web service exception is thrown if appears any problem during retrieving the endpoint interface.

1.6 Thread Safety

This component is thread-safe due to the following reasons:

The EJB WebService implementations are technically mutable since the configured managers and resources are set after construction, but the container will not initialize the properties and resources more than once for the session beans and the container ensure the thread safety in this case.

All modifying methods are transactional managed by the container so EJB implementations can be safely accessed from multiple threads in EJB container.

The ***web service clients*** are stateless and there are no explicit data store concurrency issues that are in the scope of this component so they are thread safe too.

2. Environment Requirements

2.1 Environment

- Sun Java 1.5 is required for development and Java 1.5 and Java 1.6 for compilation
- RedHat Linux 7
- Solaris 7
- Informix 10
- EJB3
- J2EE
- Windows 200, 2003

2.2 TopCoder Software Components

- **Client Project Entities DAO 1.0:** its entities are used in this design.
- **Client Project Management 1.0:** its managers are used in this design.
- **Client Project Management Services 1.0:** its UserMappingRetriever and base exceptions are used in this design.
- **Search Builder 1.4** – Filter from this component is used to perform the needed search operations.
- **Configuration API 1.0:** This is used as the main abstraction for configuration.
- **Configuration Persistence 1.0.1:** This will be used as the actual specific configuration reader for this component.
- **Object Factory Configuration API Plugin 1.0 –** is used in the creation of ObjectFactory.
- **Object Factory 2.1:** This is used to instantiate configured objects as needed by this component.
- **JBoss Login Module 2.0** – is used to perform user's authentications.
- **Logging Wrapper 2.0** – is used to perform the logging.

- ***Base Exception 2.0*** *– it is not used because of some known issues with JBoss.*

2.3 Third Party Components

- None

3. Installation and Configuration

3.1 Package Name

- **com.topcoder.clients.webservices –** contains the web services interfaces, their local and remote interfaces and their corresponding exceptions;
- **com.topcoder.clients.webservices.beans -** contains the web services implementations, their corresponding exceptions and Roles interface;
- **com.topcoder.clients.webservices.webservicesclients -** contains the web services clients and their corresponding exceptions.

3.2 Configuration Parameters

3.2.1 Resources

Configuration of resources needed in beans is realized using @Resource(name = resourceName). Here **xxx** means **company**, **client** and **project**.

Parameter	Description	Values
xxxManagerFile	Represents the manager configuration file name. ***Required***.	**String**. Can not be null or empty.
xxxManagerNamespace	Represents the manager configuration namespace. ***Required***.	**String**. Can not be null or empty.
userMappingRetrieverFile	Represents the user mapping retriever configuration file name. ***Required***.	**String**. Can not be null or empty.
userMappingRetrieverNamespace	Represents the user mapping retriever configuration namespace. ***Required***.	**String**. Can not be null or empty.
logName	Represents the configuration log name. ***Optional***.	**String**. Can be null or empty.
aAdministratorRole	Represents the configuration administrator role. ***Optional***.	**String**. Can be null or empty.
clientAndProjectUserRole	Represents the configuration client and project user role. ***Optional***.	**String**. Can be null or empty.

Below is a sample configuration for the resources:

```
<env-entry>
       <description>The name of the log to use</description>
       <env-name>logName</env-name>
       <env-type>java.lang.String</env-type>
       <env-value>log_Name</env-value>
</env-entry>
```

```
<env-entry>
       <description>The client manager file to use</description>
       <env-name>clientManagerFile</env-name>
       <env-type>java.lang.String</env-type>
       <env-value>clientManagerFile</env-value>
</env-entry>

<env-entry>
       <description>The project manager file to use</description>
       <env-name>projectManagerFile</env-name>
       <env-type>java.lang.String</env-type>
       <env-value>projectManagerFile</env-value>
</env-entry>

<env-entry>
       <description>The company manager file to use</description>
       <env-name>companyManagerFile</env-name>
       <env-type>java.lang.String</env-type>
       <env-value>companyManagerFile</env-value>
</env-entry>

<env-entry>
       <description>The client manager namespace to use</description>
       <env-name>clientManagerNamespace</env-name>
       <env-type>java.lang.String</env-type>
       <env-value>clientManagerNamespace</env-value>
</env-entry>

<env-entry>
       <description>The project manager namespace to use</description>
       <env-name>projectManagerNamespace</env-name>
       <env-type>java.lang.String</env-type>
       <env-value>projectManagerNamespace</env-value>
</env-entry>

<env-entry>
       <description>The company manager namespace to use</description>
       <env-name>companyManagerNamespace</env-name>
       <env-type>java.lang.String</env-type>
       <env-value>companyManagerNamespace</env-value>
</env-entry>

<env-entry>
       <description>The user mapping retriever file to use</description>
       <env-name>userMappingRetrieverFile</env-name>
       <env-type>java.lang.String</env-type>
       <env-value>userMappingRetrieverFile</env-value>
</env-entry>

<env-entry>
       <description>The user mapping retriever namespace to use</description>
       <env-name>userMappingRetrieverNamespace</env-name>
       <env-type>java.lang.String</env-type>
       <env-value>userMappingRetrieverNamespace</env-value>
</env-entry>


<env-entry>
       <description>The name of the administrator role</description>
       <env-name>administratorRole</env-name>
       <env-type>java.lang.String</env-type>
       <env-value>Admin</env-value>
</env-entry>

<env-entry>
       <description>The name of the client and project user role</description>
```

```
        <env-name>clientAndProjectUserRole</env-name>
        <env-type>java.lang.String</env-type>
        <env-value>Client and Projects User</env-value>
</env-entry>
```

3.2.2 *Configuration properties inside the ConfigurationObject*

Here **xxx** means **company_manager**, **client_manager, project_manager and user_mapping_retriever.**

Parameter	Description	Values
xxx_token	Represents the token used to get the needed Manager instance via object factory. ***Required***.	**String**. Can not be null or empty.

Below is a sample configuration:

- For **LookupServiceBean**:

```
<CMConfig>

    <Config name="LookupServiceBean">

        <Property name="client_manager_token">
            <Value>DAOClientManager</Value>
        </Property>

       <Property name="company_manager_token">
            <Value>DAOCompanyManager</Value>
        </Property>

       <Property name="project_manager_token">
            <Value>DAOProjectManager</Value>
        </Property>

       <Property name="user_mapping_retriever_token">
            <Value>JPAUserMappingRetriever</Value>
        </Property>

    </Config>
</CMConfig>
```

- For **ProjectStatusServiceBean**:

```
<CMConfig>

    <Config name="ProjectStatusServiceBean">

        <Property name="project_manager_token">
            <Value>DAOProjectManager</Value>
        </Property>

    </Config>
</CMConfig>
```

- For **ClientStatusServiceBean**:

```
<CMConfig>

    <Config name="ClientStatusServiceBean">

        <Property name="client_manager_token">
            <Value>DAOClientManager</Value>
        </Property>
```

```
    </Config>
</CMConfig>
```

3.3 Dependencies Configuration

- EJB 3.0 and TopCoder dependencies should be properly configured.

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

- The entities, interfaces and beans should be deployed in an EJB 3.0 container.

4.3 Demo

The following demo shows the usage of the operations available over the provided web services.

4.3.1 Demo for ProjectStatusService

```
// we have the following WSDL document location:
String wsdlLocation =
"http://topcoder.com:8080/jaxws/clientProject/ProjectStatusService/?wsdl";

// create a ProjectStatusServiceClient client:
ProjectStatusServiceClient projectStatusServiceClient = new ProjectStatusServiceClie
(wsdlLocation);

// get ProjectStatusService endpoint:
ProjectStatusService projectStatusService=
projectStatusServiceClient.getProjectStatusServicePort();

ProjectStatus projectStatus;
projectStatus = new ProjectStatus();
projectStatus.setName("project status name");
projectStatus.setDescription("project status description");

// create a projectStatus:
projectStatus = projectStatusService.createProjectStatus(projectStatus);
// if the user is in "User" or "Admin" roles then the operation has been performed
otherwise an AuthorizationFailedException has been thrown.

// update a projectStatus:
projectStatus.setDescription("project status another description");
projectStatus.setName("project status another name ");
projectStatus.updateProjectStatus (projectStatus);
// if the user is in "User" or "Admin" roles then the operation has been performed
otherwise an AuthorizationFailedException has been thrown.


// delete a projectStatus:
projectStatusService.deleteProjectStatus (projectStatus);
// if the user is in "User" or "Admin" roles then the operation has been performed
otherwise an AuthorizationFailedException has been thrown.
```

4.3.2 Demo for ClientStatusService

```
// we have the following WSDL document location:
String wsdlLocation =
"http://topcoder.com:8080/jaxws/clientProject/ClientStatusService/?wsdl";
```

```
// create a ClientStatusServiceClient client:
ClientStatusServiceClient clientStatusServiceClient = new ClientStatusServiceClient
(wsdlLocation);

// get ClientStatusService endpoint:
ClientStatusService clientStatusService=
clientStatusServiceClient.getClientStatusServicePort();

ClientStatus clientStatus;
clientStatus = new ClientStatus();
clientStatus.setName("client status name");
clientStatus.setDescription("client status description");

// create a clientStatus:
clientStatus = clientStatusService.createClientStatus(clientStatus);
// if the user is in "User" or "Admin" roles then the operation has been performed
otherwise an AuthorizationFailedException has been thrown.

// update a clientStatus:
clientStatus.setName("client status another name ");
clientStatus.setDescription("client status another description");
clientStatus = clientStatusService.updateClientStatus(clientStatus);
// if the user is in "User" or "Admin" roles then the operation has been performed
otherwise an AuthorizationFailedException has been thrown.

// delete a clientStatus:
clientStatusService.deleteClientStatus(clientStatus);
// if the user is in "User" or "Admin" roles then the operation has been performed
otherwise an AuthorizationFailedException has been thrown.
```

4.3.3 Demo for LookupService

```
// we have the following WSDL document location:
String wsdlLocation =
"http://topcoder.com:8080/jaxws/clientProject/LookupService/?wsdl";

// create a LookupServiceClient client:
LookupServiceClient lookupServiceClient =
       new LookupServiceClient (wsdlLocation);

// get LookupService endpoint:
LookupService lookupService=
lookupServiceClient.getLookupServicePort();

// we assume that there is already a filter provided by the application:
Filter filter=…;

Client client;
List<Client> clients;

ClientStatus clientStatus;
clientStatus = new ClientStatus();
clientStatus.setName("client status name");
clientStatus.setDescription("client status description");

// retrieve client with the id 111222333:
client = lookupService.retrieveClient(111222333);
// if the user is in "User" or "Admin" roles then the operation has been performed
otherwise an AuthorizationFailedException has been thrown.

// retrieve all clients:
clients = lookupService.retrieveAllClients();
// if the user is in "User" or "Admin" roles then the operation has been performed
otherwise an AuthorizationFailedException has been thrown.
```

```
// search clients with the name 'TopCoder':
clients = lookupService.searchClientsByName("TopCoder");
// if the user is in "User" or "Admin" roles then the operation has been performed
otherwise an AuthorizationFailedException has been thrown.

// search clients with the given filter:
clients = lookupService.searchClients (filter);
// if the user is in "User" or "Admin" roles then the operation has been performed
otherwise an AuthorizationFailedException has been thrown.

// get clients with the given client status:
clients = lookupService.getClientsForStatus(clientStatus);
// if the user is in "User" or "Admin" roles then the operation has been performed
otherwise an AuthorizationFailedException has been thrown.

Note: the rest of the lookup operations are performed in a similar fashion so they a
omitted from the rest of this demo.
```

5. Future Enhancements

- If the Client and Projects Management component is updated, this component will be updated to reflect the changes.