Liquid Portal Service 1.0 Component Specification

1. Design

This component will provide web service related to user and competition provisioning. The web service will be deployed as part of TopCoder's platform software (as part of the TopCoder Direct, i.e. cockpit application) and will leverage the same web services that Cockpit uses. The client stubs will be integrated into the clients Liquid Portal, which is a program management and support application for running a strategic initiative with TopCoder. The module will also provide a strategy for securing system to system communications and allowing the client system to act on behalf of a remote user.

Conventions: When referring to bean properties in this component, the standard bean dot notation is used in lieu of quoting the actually get/set methods. So instead of having

```
createCompetitonResult.getCompetition().getId()
```

We will write

```
createCompetitonResult.competition.id
```

1.1 Design Patterns

1.1.1 Strategy

LiquidPortalServiceBean is an EJB 3.0 strategy implementation of the API represented by OrderManager.

It also uses strategy with the LiquidPortalService so it can be used transparently by web service clients.

1.1.2 DTO

All entities in this component are DTOs.

1.2 Industry Standards

- Web Services (JAX-WS 2.0)
- Inversion of Control (IoC)
- EJB 3.0

1.3 Required Algorithms

1.3.1 Logging standard for all bean business methods

This section will state the complete scenarios for logging in all business methods.

- Method entrance and exit will be logged with INFO level.
    - Entrance format: [Entering method {*className.methodName}*]

    - Exit format: [Exiting method {*className.methodName}]*. Only do this if there are no exceptions.
- All exceptions will be logged at ERROR level.
    - Format: Simply log the text of exception: [Error in method {*className.methodName}*: Details {*error details*}]

In general, the order of the logging in a method should be as follows:

1. Method entry
2. Log method entry
3. If error occurs, log it, and skip remaining steps
4. Log method exit
5. Method exit

1.3.2 Register User

1.3.2.1 Process

- The service will delegate to existing TopCoder API for creating handles.
- Add user to the Notus Observer Terms and Notus Eligibility groups.

1.3.2.2 Output

- An exception will be thrown if the handle was not created.
- A warning condition will be returned if the new user was not added to the Terms or Eligibility groups. This should also be logged to Jira for system admin support by sending an email to JIRA (see CS 1.3.9 and defer to sendJiraNotification method).
    - Send to jiraEmailAddress with subject jiraEmailSubject.
    - The message needs to mention that the user was not added
- If success, return the result with userId.

1.3.3 Validate User

1.3.3.1 Process

- The service will use existing TopCoder APIs to retrieve the handle and certain other user information. The service will check that the First Name, Last Name, and Email Address all match the database information.
- Add user to the Notus Observer Terms and Eligibility groups. In the following situations.
    - All demographic information matches the request
    - Force is set to true

1.3.3.2 Output

- An exception will be thrown if the handle does not exist.
- A warning condition will be returned if any demographic information doesn't match (Email, First or Last Name)
- A warning condition will be returned if the user failed to be added to the Terms or Eligibility group
- If success, return an empty result (with no warnings).

1.3.4 Provision User

1.3.4.1 Process

- Validate Handle and User
  - The service will confirm the Requestor and User Handle are valid
  - The service will confirm that both users are part of the Notus Eligibility group.
  - Requestor and User Handle may be the same.
- Add User to Cockpit Projects
  - For each cockpit project tied to the Notus Account – if the project is in the submitted list add the user, if not remove the user.
  - If "Has Account Access" is true then add user to all projects.
- Add User to Billing Projects
  - For each Billing Project associated to the Notus Account – if the project is in the submitted list add the user, if not remove the user.
  - If "Has Account Access" is true then add user to all projects.

1.3.4.2 Output

- An exception will be thrown if either handle didn't validate or the changes couldn't be completed
- A warning condition will be returned if any cockpit or billing project submitted was not found or not part of the Notus account.
- If success, return the result with the retrieved billing projects and cockpit projects.

1.3.5 Create Competition

1.3.5.1 Process

- Validate Permissions
  - Validate Requestor Handle exists
  - Validate Requestor is a member of Notus Eligibility Groups
  - Validate Requestor is a member of the Billing Project
  - Validate that if Cockpit Project exists, the requestor has permissions. To find the cockpit project lookup by project name and association with Notus Account
- Create Cockpit Project
  - If the cockpit project doesn't exist create the project using Cockpit Project as name and associating it with Notus Account
- Create Contest
  - Validate contest metadata (Type, Subtype, Name)
  - Create contest on requested date
  - If date is at capacity and Automatically Reschedule is true, then use the next available date
- Add Support Handles
  - For each support handle
    - Check handle exists
    - Check user is part of Notus Eligibility groups

        - Add handle to OR as observer (including forum permissions, file upload, and forum watch)

1.3.5.2 Output

- An exception will be thrown if
  - the permissions are invalid
  - Cockpit project creation failed
  - The contest cannot be created (including because date was unavailable and reschedule was not allowed)
- Warning should be returned if
  - Some or all of the support handles could not be added as observers in OR
  - Project had to be created
- If successful, return the result with created contest.

1.3.6 Provision Project

1.3.6.1 Process

- Validate Permissions
  - Validate Requestor Handle exists
  - Validate Requestor is a member of Notus Eligibility Groups
  - Validate that if Cockpit Project exists, the requestor has permissions. To find the cockpit project lookup by project name and association with Notus Account
  - Validate that the Requestor had permissions to the cockpit project
- Add Handles
  - For each handle
    - Check handle exists
    - Check user is part of Notus Eligibility groups
    - Add handle to cockpit project with full control

1.3.6.2 Output

- An exception will be thrown if
  - If the permissions are invalid
  - Cockpit project did not exist
- Warning should be returned if
  - Some or all of the handles could not be added to project with full control
- If successful, return an empty result (with no warnings).

1.3.7 Delete Competition

1.3.7.1 Process

- Validate Permissions
  - Validate Requestor Handle exists
  - Validate Requestor is a member of Notus Eligibility Groups
  - Validate that contest exist

- Validate that the user has permissions for the billing account associated to the competition
- Validate that the user has permissions for the cockpit project associated to the competition

- Delete Contest
  - Set the contest OR status to deleted, and use <Requestor Handle>: <Reason> as the explanation.

1.3.7.2 Output

- An exception will be thrown if any validation fails or contest delete fails

1.3.8 Decommission User

- Validate Permissions
  - Validate Requestor Handle exists
  - Validate Requestor is a member of Notus Eligibility Groups
  - Validate User Handle exists
  - Validate User Handle is a member of Notus Eligibility Groups
- Remove User from Cockpit Projects
  - For each cockpit project that is associated with Notus Account remove user if the user has permissions
- Remove User from Billing Accounts
  - For each billing account that is associated with Notus Account remove user if the user has permissions
- Remove User from Eligibility Groups and Terms
  - Remove user from Notus Eligibility groups
  - Remove user from Notus Observer Terms

1.3.8.1 Output

- An exception will be thrown if any validation fails or any removal fails

1.3.9 Email notification message

The email notification message will be assembled from the handle. It will have the following structure:

```
<DATA>
    <HANDLE>handle</HANDLE>
<DATA>
```

1.4 Component Class Overview

LiquidPortalService

This represents the API to manage portal users, create and delete contests, and provision projects.

LiquidPortalServiceLocal

The local EJB interface that simply extends the LiquidPortalService interface, with no additional facilities.

LiquidPortalServiceRemote
The remote EJB interface that simply extends the LiquidPortalService interface, with no additional facilities.

LiquidPortalServiceBean
This class provides an implementation of the LiquidPortalService via its Local and Remote interfaces. It uses various services to process liquid service requests. It uses Logging Wrapper to log events.

CompetitionData
Represents the data for creating a competition. Provides data about the names of the project, contest, types and sub types, the requested start date, the associated billing project, and flags whether to auto-start the competition and whether a CCA is required for it. It is passed in the createCompetition method.

Result
Represents the processing result. The absence of warnings signifies a flawless execution. Otherwise, any warning will detail when something non-fatal occurred. Returned in the validateUser and provisionProject methods of the service.

Warning
Represents a non-fatal warning regarding something that went wrong during processing. It contains the message.

RegisterUserResult
A type of Result that also provides the registered user ID. Returned in the registerUser method of the service.

CreateCompetitonResult
A type of Result that also provides the competition just created. Returned in the createCompetition method of the service.

ProvisionUserResult
A type of Result that also provides the cockpit and billing projects associated with the user. Returned in the provisionUser method of the service.

1.5 Component Exception Definitions

This component defines new exceptions.

LiquidPortalServicingException
This exception is the top-level exception thrown by the liquid portal service. It extends Exception.

HandleCreationException

This exception is thrown by the service if it cannot create the given handle. It extends LiquidPortalServicingException.

HandleNotFoundException
This exception is thrown by the service if it cannot find the given handle. It extends LiquidPortalServicingException.

ActionNotPermittedException
This exception is thrown by the service if the user does not have permission to perform the action. It extends LiquidPortalServicingException.

ContestNotFoundException
This exception is thrown by the service if it cannot find the given contest. It extends LiquidPortalServicingException.

LiquidPortalServiceConfigurationException
This exception signals an issue if the configuration fails for any reason during initialization. It extends BaseRuntimeException. It is used by all classes that perform initialization via configuration.

LiquidPortalIllegalArgumentException
This exception is thrown by LiquidPortalServiceBean if the argument of public methods is invalid. An error code is provided to indicate which argument is invalid.

InvalidHandleException
This exception is thrown by the service if the handle is invalid. It extends LiquidPortalServicingException

1.6 Thread Safety

The service implementations need to be effectively thread-safe, and the provided implementations are, as they are immutable. The EJB will have its state set during initialization, and not modified afterwards, which makes it effectively thread-safe. All components used by this component are also effectively thread-safe.

Overall, the lack of thread-safety comes from the non-thread-safe entities. But as they are POJOs and are not expected to he handled by more than one thread at a time. To handle scenarios where they would not be used in a thread-safe manner would require these entities to be changed to be thread-safe themselves.

1.7 Transaction Management

As per requirements, all transactions will be container managed and required for all modification methods.

2. Environment Requirements

2.1 Environment

- Development language: Java 1.5, J2EE 1.5
- Compile target: Java 1.5, Java 1.6, J2EE 1.5
- Application server: JBoss 4.2

2.2 TopCoder Software Components

- Logging Wrapper 2.0
    - Used for logging operations.
- Configuration API 1.0
    - The substitution to the ConfigManager. It provides the configuration parameters for all classes.
- Configuration Persistence 1.0
    - Provides file-based configuration that provides configuration as a ConfigurationObject.
- Client Project Entities DAO 1.1
    - Provides the project entities
- Project Service 1.1
    - Provides the ProjectService and the ProjectData
- Contest Service Façade 1.0
    - Provides the ContestServiceFacade.
- Pipeline Service Façade 1.0
    - Provides the PipelineServiceFacade and CapacityData.
- Contest and Submission Entities 1.0
    - Provides the contest and submission entities.
- User Service 1.0
    - Provides the UserService and entities.
- Email Engine 3.1
    - Used to send messages to JIRA.
- Document Generator 3.0
    - Provides the facility to generate messages.

2.3 Third Party Components

None

3. Installation and Configuration

3.1 Package Names

com.liquid.portal.service
com.liquid.portal.service.bean

3.2 Configuration Parameters

3.2.1 LiquidPortalServiceBean configuration in initialize method

A configuration named "document_generator_configuration" should be configured, this configuration object is used for document generator component. So the properties in the configuration object should be set properly. Refer to 3.2.2 for an example.

Properties in root

property name	value	required
notusObserverTermsId	Represents the notus observer of terms ID	Yes
notusEligibilityGroupIds	Represents the IDs of notus eligible groups	Yes
fullControlPermissionTypeId	Represents the full control permission ID	Yes
projectCategories	Represents the mapping of project categories names to their IDs. It will have a list of the following entries: <name>::<id>	Yes
studioContestTypes	Represents the mapping of studio contest type names to their IDs. It will have a list of the following entries: <name>::<id>	Yes
inactiveStudioContestStatusId	Represents the ID of the inactive studio contest status	Yes
inactiveSoftwareContestStatusId	Represents the ID of the inactive software contest status	Yes
deletedSoftwareContestStatusId	Represents the ID of the deleted software contest status	Yes
notusClientId	Represents the notus client ID	Yes
billingAccountPropertyKey	Represents the billing account property key	Yes
deleteReasonPropertyKey	Represents the reason account property key	Yes
logName	The name of the log to use	Yes
jiraEmailAddress	Represents the email address for JIRA	Yes
jiraEmailSubject	Represents the subject of the email message to send to JIRA	Yes
jiraEmailSender	Represents the email address for the sender of the email to JIRA	Yes
jiraEmailMessageTemplateName	Represents the template name of the message to use to send to JIRA	Yes

3.2.2 Sample Configuration File

```
<?xml version="1.0"?>
<CMConfig>
        <Config name="com.liquid.portal.service">
                <property name="notusObserverTermsId">
                        <value>1</value>
                </property>
```

```
        <property name="projectCategories">
                <value>development::1</value>
                <value>design::2</value>
        </property>

        <property name="studioContestTypes">
                <value>icon::3</value>
                <value>flash::4</value>
        </property>

        <property name="notusEligibilityGroupIds">
                <value>1</value>
                <value>2</value>
                <value>3</value>
        </property>

        <property name="fullControlPermissionTypeId">
                <value>11</value>
        </property>

        <property name="inactiveStudioContestStatusId">
                <value>12</value>
        </property>

        <property name="inactiveSoftwareContestStatusId">
                <value>13</value>
        </property>

        <property name="deletedSoftwareContestStatusId">
                <value>14</value>
        </property>

        <property name="notusClientId">
                <value>15</value>
        </property>

        <property name="billingAccountPropertyKey">
                <value>account</value>
        </property>

        <property name="deleteReasonPropertyKey">
                <value>reason</value>
        </property>

        <property name="logName">
                <value>portal</value>
        </property>

        <property name="jiraEmailAddress">
                <value>topcoder@topcoder.com</value>
        </property>

        <property name="jiraEmailSubject">
                <value>hello</value>
        </property>

        <property name="jiraEmailSender">
                <value>topcoder@topcoder.com</value>
        </property>

        <property name="jiraEmailMessageTemplateName">
                <value>hellotemplate</value>
        </property>
</Config>
<Config name="document_generator_configuration">
        <Property name="sources">
     <Value>file</Value>
 </Property>

 <!-- the identifier of the template source to be used as default -->
```

```
        <Property name="default_source">
            <Value>file</Value>
        </Property>

        <!-- here follow custom properties for each template source -->
        <!-- the <sourceidentifier> class property is mandatory -->
        <!-- the other <sourceidentifier> <property name> properties are specific for
each source -->

        <!-- file template source properties -->
        <Property name="file_class">
            <Value>com.topcoder.util.file.templatesource.FileTemplateSource</Value>
        </Property>
       </Config>
</CMConfig>
```

3.3 Dependencies Configuration

3.3.1 Dependency configuration

The developer should read the specifications for all components specified in section 2.2 to see how they are configured.

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

To deploy this component in Jboss, We need ejb-jar.xml and jboss.xml.
Ejb-jar.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<ejb-jar xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/ejb-jar_3_0.xsd"
         version="3.0">
  <enterprise-beans>
    <session>
        <ejb-name>LiquidPortalServiceBean</ejb-name>
        <remote>com.liquid.portal.service.bean.LiquidPortalServiceRemote</remote>
        <local>com.liquid.portal.service.bean.LiquidPortalServiceLocal</local>
              <service-endpoint>com.liquid.portal.service.LiquidPortalService</service-
endpoint>
        <ejb-class>com.liquid.portal.service.bean.LiquidPortalServiceBean</ejb-class>
        <session-type>Stateless</session-type>
        <transaction-type>container</transaction-type>
              <env-entry>
                     <description>the configuration file path</description>
                     <env-entry-name>configurationFileName</env-entry-name>
                     <env-entry-type>java.lang.String</env-entry-type>
                     <env-entry-value>liquidPortalService.properties</env-entry-value>
              </env-entry>
              <env-entry>
                     <description>the configuration namespace</description>
                     <env-entry-name>configurationNamespace</env-entry-name>
                     <env-entry-type>java.lang.String</env-entry-type>
                     <env-entry-value>com.liquid.portal.service</env-entry-value>
              </env-entry>
        <ejb-ref>
               <ejb-ref-name>ejb/pipelineServiceFacade</ejb-ref-name>
               <ejb-ref-type>Session</ejb-ref-type>
```

```
            <remote>com.topcoder.service.pipeline.PipelineServiceFacade</remote>
        </ejb-ref>
            <ejb-ref>
             <ejb-ref-name>ejb/contestServiceFacade</ejb-ref-name>
             <ejb-ref-type>Session</ejb-ref-type>
             <remote>com.topcoder.service.facade.contest.ContestServiceFacade</remote>
        </ejb-ref>
            <ejb-ref>
             <ejb-ref-name>ejb/userService</ejb-ref-name>
             <ejb-ref-type>Session</ejb-ref-type>
             <remote>com.topcoder.service.user.UserService</remote>
        </ejb-ref>
            <ejb-ref>
             <ejb-ref-name>ejb/projectService</ejb-ref-name>
             <ejb-ref-type>Session</ejb-ref-type>
             <remote>com.topcoder.service.project.ProjectService</remote>
        </ejb-ref>
            <ejb-ref>
             <ejb-ref-name>ejb/billingProjectDAO</ejb-ref-name>
             <ejb-ref-type>Session</ejb-ref-type>
             <remote>com.topcoder.clients.dao.ProjectDAO</remote>
        </ejb-ref>
    </session>
  </enterprise-beans>
</ejb-jar>
```

Jboss.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<jboss>
  <enterprise-beans>
    <session>
        <ejb-name>LiquidPortalServiceBean</ejb-name>
        <jndi-name>remote/LiquidPortalServiceBean</jndi-name>

        <ejb-ref>
            <ejb-ref-name>ejb/pipelineServiceFacade</ejb-ref-name>
            <jndi-name>remote/PipelineServiceFacadeBean</jndi-name>
        </ejb-ref>
            <ejb-ref>
            <ejb-ref-name>ejb/contestServiceFacade</ejb-ref-name>
            <jndi-name>remote/ContestServiceFacadeBean</jndi-name>
        </ejb-ref>
            <ejb-ref>
            <ejb-ref-name>ejb/userService</ejb-ref-name>
            <jndi-name>UserServiceBean/remote</jndi-name>
        </ejb-ref>
            <ejb-ref>
            <ejb-ref-name>ejb/projectService</ejb-ref-name>
            <jndi-name>remote/ProjectServiceBean</jndi-name>
        </ejb-ref>
            <ejb-ref>
            <ejb-ref-name>ejb/billingProjectDAO</ejb-ref-name>
            <jndi-name>ProjectDAOBean/remote</jndi-name>
        </ejb-ref>
    </session>
  </enterprise-beans>
</jboss>
```

Since we use EJB 3.0, deployment of the service is simple. Depending on the application server used, all necessary artifacts (such as the deployment descriptor) can be automatically generated. The class files and any generated artifacts need to be packaged in an EJB JAR file, which should then be deployed in an EJB container.

4.2.1 Deploying the web service

- Extract the component distribution

- Follow Dependencies Configuration.
- Build the EAR file containing the web service
- Copy the EAR file to the JBOSS_INSTALL_PATH/server/default/deploy directory
- Start JBoss

4.2.2 *Generating the WSDL and web services artifacts*

The WSDL description and all other artifacts for the web service will automatically be generated by JBoss at deploy time. The URL to the WSDL file may be found on the following page provided by JBoss, which lists all the deployed web services:

```
http://[SERVER_HOST]:[SERVER_PORT]/jbossws/services
```

4.3 Demo

4.3.1 *API demo*

This will show how the API is used.

```
// In a client, the EJB would be injected directly, so we assume in our
testing client we have access to it
LiquidPortalService service = // EJB instance

// We begin by registering some users
User user = new User();
user.setFirstName("Lucian");
user.setLastName("Illiuescu");
user.setEmailAddress("hammer@topcoder.com");
user.setHandle("hammer");
user.setPassword("hammerfall");
user.setPhone("456-234-5678");
RegisterUserResult registerUserResult = service.registerUser(user, new
Date());
// registerUserResult contains the user ID assigned to user, so we add it
user.setUserId(registerUserResult.getUserId());

// We next validate the user
long[] groupIds = // some valid IDs
user.setGroupIds(groupIds);
Result validationResult = service.validateUser(user, true);
// the validation would be successful

// next we provision the user
String[] cockpitProjectNames = // some valid names
long[] projectIds = // some valid projects IDs
ProvisionUserResult provisionUserResult = service.provisionUser("ivern",
"hammerfall", true, cockpitProjectNames, billingProjectIds);
// provisionUserResult would return with the instances of the cockpit
projects and billing projects

// administrator ivern can also take the time to decomission some users
service.decommissionUser("ivern", "drunkenlad");
// this removes the user "drunkenlad"


// next we create a competition
String[] supportHandles = {"GregEldridge"};
CompetitionData competitionData = new CompetitionData();
competitionData.setContestTypeName("software");
competitionData.setContestName("World Domination");
competitionData.setCockpitProjectName("Excalibur");
competitionData.setRequestedStartDate(// Feb 3rd, 2010);
competitionData.setAutoReschedule(false);
```

```
competitionData.setBillingProjectId(3);
competitionData.setCca(false);
CreateCompetitonResult createCompetitonResult =
service.createCompetition("hammerfall", competitionData, supportHandles);
// the result provides us the instance of the created competition. we
assume ID 4569

// next we provision the project
String[] handles = {"CParish"};
Result projectProvisionResult = service.provisionProject("hammerfall",
"Excalibur", handles);

// After consideration, the user does not want the competition, so he
deletes it
service.deleteCompetition("hammerfall", 4569, false, "out of cash");
// this contest is now deleted
```

5. Future Enhancements

None