User Service 1.2 Component Specification

1. Design

This component provides a web service interface to allow users to perform CRUD operations on user data. A user is used to group different multiple user informations such as address, user name, etc.

The design is very simple. The web interface UserService defines the methods that are exposed by the web interface. Two marker extensions of this interface provide local and remote interfaces for access through EJB containers.

In the version 1.2:

Methods are added to register/get user, add/remove user from/to group and add/remove user/use of term.

1.1 Design Patterns

Strategy Pattern - The strategy pattern is used by UserService to abstract the user management contract.

Data Transfer Object Pattern - The user related entities are all serializable business entities whose purpose is to act as data transfer objects.

Data Access Object Pattern - The UserService contract acts as the data access object for the Project entity.

1.2 Industry Standards

SOAP - This component provides a SOAP-based web service.

1.3 Required Algorithms

1.3.1 Logging

Only the service/bean methods of this component will log useful information. Logging should be performed only if a non-null Log object has been plugged into the bean.

- Error Level - Any exceptions that are raised or caught will be logged at this level.
- Info Level - All method entries and exits will be logged at this level. For method entry, the log must contain the method arguments and for method exit, the return value if any.

1.3.2 Dummy values for user and user_group_xref tables

When saving the user, developers can assume that:

For the time_zone_id in the user table, 1 is used, assuming that the time_zone_id=1 exists in the time_zone_type_lu table.

The phone of the User is inserted into the phone table, assuming that the phone_type_id=1 exists in the phone_type_lu table. The phone_id and user_id will be inserted into the phone table, the phone_id is generated by ID Generator.

The email of the User is stored in the email table. The email_id is generated by ID Generator. The status_id of the email is be configurable, it will be injected into the UserServiceBean.

The address of the User is inserted into the address table, assuming that the address_type_id=1 exists in the address_type_lu table. The address information will be stored in the address table. The address id will be generated by ID Generator, and the address id and user id association will be inserted into the user_address_xref table.

The user table is:

```
create table 'informix'.user (
        user_id DECIMAL(10,0) not null,
        first_name VARCHAR(64),
        last_name VARCHAR(64),
        create_date DATETIME YEAR TO FRACTION
                default CURRENT YEAR TO FRACTION,
        modify_date DATETIME YEAR TO FRACTION
                default CURRENT YEAR TO FRACTION,
        handle VARCHAR(50) not null,
        last_login DATETIME YEAR TO FRACTION,
        status VARCHAR(3) not null,
        password VARCHAR(30),
        activation_code VARCHAR(32),
        middle_name VARCHAR(64),
        handle_lower VARCHAR(50),
        timezone_id DECIMAL(5,0),
        last_site_hit_date DATETIME YEAR TO FRACTION
)
```

For the +addUserToGroups(handle:String,groupIds:long[]):void method, please note that the following is assumed: create_user_id=1,security_status_id=1 already exists, and are used as dummy values.

```
create table 'informix'.user_group_xref (
        user_group_id DECIMAL(12,0) not null,
        login_id DECIMAL(12,0),
        group_id DECIMAL(12,0),
        create_user_id DECIMAL(12,0),
        security_status_id DECIMAL(3,0),
        create_date DATETIME YEAR TO FRACTION
                default CURRENT YEAR TO FRACTION
)
```

1.3.3 Use ID Generator to generate ids

Algorithm 1.3.2 generates the entities' ids. Each entity has its separate ID Generator. For instance, for the address id, the id is generated this way:

```
long addressed =
```

IDGeneratorFactory.getIDGenerator(this.addressIdGeneratorName).getNextID();

1.4 Component Class Overview

UserServiceBean

It provides CRUD on user object.

Updated for Jira and Confluence User Sync Widget 1.0

The mock implementation of getEmailAddress and isAdmin has been moved from user_sync_service UserServiceBean class.

The mock implementation for getEmailAddress(..) returns <userHandle>@topcoder.com email address for any user handle

that starts with alphabets and just has allowed character sets [A-Z], [a-z], [0-9], _ (a underscore). Other wise it

returns null.

The mock implementation for isAdmin(..) returns true for 'user' handle or all those handles that has only Upper case

alphabets.

Annotations:

@RolesAllowed({ "Cockpit User", "Cockpit Administrator" })

@RolesAllowed({ "Cockpit User", "Cockpit Administrator" })

@DeclareRoles({ "Cockpit User", "Cockpit Administrator" })

@DeclareRoles({ "Cockpit User", "Cockpit Administrator" })

@TransactionManagement(TransactionManagementType.CONTAINER)

@TransactionAttribute(TransactionAttributeType.REQUIRED)

@Stateless

----------Modified in the version 1.2------

Methods are added to register/get user, add/remove user from/to group and add/remove user/use of term.

The thread safety requirement is not changed.

UserServiceRemote

This is the local interface for EJB 3.0 bean. It is a marker interface which simply extends the<code>{@link UserService}</code> contract. It allows the <code>{@link UserService}</code> to also be

accessed as a local bean.

Implementations will simply need to implement the base contract - <code>{@link UserService}</code>. The marker

interface itself requires no special implementation. Thread Safety: Implementations must be thread safe.

Annotations:

@Remote

UserServiceLocal

This is the local interface for EJB 3.0 bean. It is a marker interface which simply extends the<code>{@link UserService}</code> contract. It allows the <code>{@link UserService}</code> to also be

accessed as a local bean.

Implementations will simply need to implement the base contract - <code>{@link UserService}</code>. The marker

interface itself requires no special implementation. Thread Safety: Implementations must be thread safe.

Annotations:

@Local

UserService

It provides various getters on user object.

Updated for Jira and Confluence User Sync Widget 1.0

- Moved the methods that existed in user_sync_service component's UserService.

Annotations:

@Remote

----------Modified in the version 1.2------

Methods are added to register/get user, add/remove user from/to group and add/remove user/use of term.

The thread safety requirement is not changed.

UserServiceFault

This class represents the "user_service_fault" element in WSDL. It's contained in the related exception-message

class. It was generated by JBoss tools and I changed the name and other little changes.

Updated for Jira and Confluence User Sync Widget 1.0

- Correct few of JavaDoc comments.

This class is not thread safe because it's highly mutable

Annotations:

@XmlAccessorType(XmlAccessType.FIELD)

@XmlType(name = "", propOrder = { "faultMessage" })

@XmlType(name = "", propOrder = { "faultMessage" })

@XmlRootElement(name = "user_service_fault")

@XmlRootElement(name = "user_service_fault")

Address

Represents the Address entity.

It holds the attributes address id, address type, address1, address2, etc.

It's mutable and not thread safe.

User

Represents the User entity.

It holds the attributes user id, first name, last name, handle, etc.

It's mutable and not thread safe.

UserInfo

Represents the UserInfo entity.

It holds the attributes user id, first name, last name, handle, etc.

It's mutable and not thread safe.

1.5 Component Exception Definitions

UserServiceException

A common exception class for the user services.

Updated for Jira and Confluence User Sync Widget 1.0

- Specified the @WebFault mapping

- Initialize fault message too.

- For proper exception serialization to webservice faultInfo is also initialized now

Annotations:

@WebFault(name = "user_service_fault", faultBean = "com.topcoder.service.user.UserServiceFault")

@WebFault(name = "user_service_fault", faultBean = "com.topcoder.service.user.UserServiceFault")

1.6 Thread Safety

The thread safety requirement is not changed in the version 1.2

This component is, strictly speaking, not thread safe since the data objects provided are mutable and thus not thread safe. The bean class implementation are thread safe. This allows for two ways in which the component may be used in a thread safe manner.

- If the bean is accessed as a remote interface or as a web service, thread safety is

automatically achieved. This is because all data objects used during communication will be serialized/deserialized at either end, thus ensuring that there are no shared data object references.

- If the bean is accessed as a local interface, then all data objects used to communicate must be used from a single thread. This can be easily done by only creating and using data objects within method scopes and not having data objects as shared instance members. If shared data objects are a must, some locking mechanism must be used so that the shared instances are not modified concurrently.

2. Environment Requirements

2.1 Environment

- Java 1.5, EJB 3.0 and JPA 1.0 are required for compilation and execution.

2.2 TopCoder Software Components

- **Base Exception 2.0** - This component provides the base exception for the exceptions of our component.
- **JBoss Login Module 2.0 -** This component provides us with the UserProfilePrinicipal class which is used for authentication.
- **Logging Wrapper 2.0 -** This component allows our component to use logging functionality.

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

- Hibernate Core - The core allows us to map Java classes to database tables using Hibernate and also allows us to use Hibernate persistence.
- Hibernate Entity Manager - The entity manager allows us to use Hibernate as if it were just another JPA persistence provider.

Both of these components may be downloaded from http://www.hibernate.org/ The site also contains useful information on how to setup and use the components. The Hibernate version used must be 3.2 or higher.

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.service.user

com.topcoder.service.user.ejb

3.2 Configuration Parameters

Note the all configuration properties should be set as environment entries in the EJB container.

3.2.1 *Configuration for ProjectServiceBean*

Name	Description
unitName	This property gives the name of the persistence unit to use to create the EntityManager. **Required.**
log_name	This property gives the name of the log to create using the LogManager class. **Optional.**
roles_key	This property gives the name of the key used to fetch the roles from the attributes of the user profile. **Required.**
administrator_role	This property gives the name of the administrator role. **Required.**
user_role	This property gives the name of the user role. **Required.**
emailStatusId	The email status id, required
addressIdGeneratorName	The id generator name used to get the separate id generator.
userIdGeneratorName	The id generator name used to get the separate id generator.
emailIdGeneratorName	The id generator name used to get the separate id generator.
phoneIdGeneratorName	The id generator name used to get the separate id generator.
userGroupIdGeneratorName	The if generator name used to get the separate id generator.

3.2.2 *Configuration for JPAProjectPersistence*

Name	Description
persistence_unit_name	This property gives the name of the persistence unit to use to create the EntityManagerFactory. **Required.**

3.3 Dependencies Configuration

To use named logs, the LoggingWrapper component must be configured with a Log under the appropriate name.

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

Please follow the section 3.2

4.3 Demo

The service of this component is likely to be used by users and administrators to CRUD

```
Properties env = new Properties();
env.setProperty(Context.INITIAL_CONTEXT_FACTORY,
"org.jnp.interfaces.NamingContextFactory");
env.setProperty(Context.URL_PKG_PREFIXES, "org.jboss.naming:org.jnp.interfaces");

Context context = new InitialContext(env);
```

```
UserService userServiceBean = (UserService)
       context.lookup("UserServiceBean/Remote");

// get email address
String emailAddress = userServiceBean.getEmailAddress(userId);

// get user id
userId = userServiceBean.getUserId(userHandle);

// create the User instance
User user = new User();

// set first name
user.setFirstName(firstName);

// set last name
user.setLastName(lastName);

// set handle
user.setHandle(handle);

// register user
userId = userServiceBean.registerUser(user);

// get user
UserInfo userInfo = userServiceBean.getUser(handle);

// add user to group
userServiceBean.addUserToGroups(handle, groupIds);

// remove the user from group
userServiceBean. removeUserFromGroups(handle, groupIds);

// add user term
userServiceBean.addUserTerm(handle, useOfTermId, new Date());

// remove user term
userServiceBean.removeUserTerm(handle, useOfTermId);
```

5. Future Enhancements

None.