Time Tracker User 1.0 Component Specification

1. Design

The Time Tracker User custom component is part of the Time Tracker application. It provides an abstraction of a user in the system which will be imported from a variety of sources. This component handles the persistence and other business logic required by the application.

The component logic is divided into four separate aspects:

- Management of imported users (requirement 1.2.1) – **UserManager** class provides all required functionality and allows import of users from user stores, removing of imported users, enumeration of all imported users and retrieving of imported user.
- API for User Stores (requirement 1.2.2) - the design provides a pluggable framework (**UserStoreManager** and **UserStore** interfaces)for the purpose of importing users from different user stores into the Time Tracker application. Also, the relationship between the imported users and the original user store must be maintained.
- Default User Store (requirement 1.2.2.3) - **DbUserStore** class provides default user store based on database implementation for clients that do not have an existing user store. Data schema is provided within docs/Time_Tracker_User.sql
- User Roles (requirement 1.2.3) – the component utilizes the **Authorization** component to provide user roles management and persistence.
- User Persistence (requirement 1.2.4) – the component utilizes the **Authorization** component to provide user role persistence and defines own **UserPersistence** interface to provide imported user persistence (**UserPersistenceImpl** class is default database implementation that utilizes **DB Connection Factory** and supports any pluggable database systems, data schema is provided within docs/Time_Tracker_User.sql)

The component can be used in two ways – simplified, when no parameters are needed for UserManager (all needed data will be retrieved from configuration file) and advanced when user can pass **UserPeristence**, **AuthorizationPersistence** and **UserRoleManager** instances. This way the component is very flexible (because it allows custom persistence implementations), but still allows simple usage (when default persistence implementation are used).

UserManager class defines set of following required user roles as static final variables:

- Super Administrator – this user can perform all functions and view all data
- Human Resource – this user is from the Human Resource Department
- Account Manager – this user manages multiple clients
- Project Manager – this user manages multiple projects
- Internal Employee – this user is a regular employee
- Contractor – this user works on a per contract basis

Note, a user should add them to Time Tracker Application only once, during deployment, using

UserManager.getAuthPersistence().AddRole method.

In the future a user can add more roles or change existing per application requirements using RemoveRole, AddRole methods UserManager.getAuthPersistence() (Authorization component)

To enumerate all available user roles user will use UserManager.getAuthPersistence().getSecurityRoles() method

1.1 Design Patterns

- **Delegate** pattern is utilized by UserManager to provide user role management (getUserRole, setUserRole methods just reuse appropriated AuthorizationPersistence methods)

- **Strategy** pattern is utilized by UserStore, UserStoreManager and UserPersistence interface and their implementations. UserStoreManager is used to support UserStore implementations (user store “plugins”). UserStoreManagerImpl provides default UserStoreImplementation and provides adding/removing/enumeration of UserStore instances.

1.2 Industry Standards

None

1.3 Required Algorithms

None

1.4 Component Class Overview

UserManager

The central class of the component provides importing of user from a variety of sources. An internal ID is generated for imported user (using ID Generator component) and the original user name is replicated locally for performance (User instance). Also the class provides authentication of user with given password using UserStore.authenticate method.

The class makes of use of the Authorization component to support user roles and permissions. Since there are no defined permissions for this component, the class provides access to used authorization persistence mechanism (getAuthorization method) and principal (getUser), so Time Tracker application will be able to define permissions and apply them for imported users and assigned roles.

For this application user has only one assigned role, so the class utilizes Facade pattern and provides easier API for managing of roles than Authorization component (setUserRole, getUserRole methods)

UserManager uses Configuration Manager instance to retrieve all used instances from configuration file and hide them from end user (UserManager() constructor).

Database logic is separated from business logic of the component - UserPersistence and AuthorizationPersistence instances are used to store/load user roles and imported users. It can be any database system or other data source.

UserStoreManager

The interface represents general framework for pluggable user sources. The interface provides mapping between String representation of user store (unique user store name) and UserStore instance.

The interface allows adding/removing user store, get all supported user store names and retrieve UserStore instance by given unique name.

UserStoreImpl

UserStoreManager implementation provides basic support for pluggable user stores.

The class allows adding/removing of user store, get all supported user store names and retrieve UserStore instance by given unique name.

The manager allows definition of predefined set of user stores within configuration file. Constructor reads all user store names and classes, instantiate them to UserStore instances and add to the manager.

UserStore

This interface provides API for pluggable user store - generic interface for importing users by Time Tracker application. To facilitate requirements, this interface provides the following operations:

- Connection - a way to connect to the user store (setConnectionString method)
- Enumeration - a way to enumerate all existing users in the user store (getUsernames method)
- Search - a way to search for specific users by name in the user store (contains, search methods)
- Authentication - a way to authenticate a user against a given password

The component contains one implementation of this interface - DbUserStore class which provides database backend persistence for user store.

DbUserStore

For clients that do not have an existing user store, a default database implementation is provided. Clients will maintain the database externally. The Time Tracker application will be able to import users from the default implementation just like any other user stores.

This class provides default UserStore implementation which will be used by clients that do not have an existing user store. For the initial version, the Informix database system will be used as persistence storage for this component and the Time Tracker application. Note, that DbUserStore supports any database registered by DB Connection Factory component, so other database systems are easy pluggable into the framework.

Used database table is "DefaultUsers" and described by Requirement Specification.

User

The class represents imported user entity for Time Tracker application. It extends GeneralPrincipal from Authorization Component and adds one more parameter - name of user store where this user imported from.

This way relationship between the imported users the original user store will be maintained and stored within data source (using UserPersistence implementation)

UserPersistence

The interface represents general persistence API for loading/saving imported user entities. Note, that User Roles are persisted by Authorization Component, so this component is responsible only to persist imported users. Also, the relationship between the imported users and the original user store is maintained. Following basic operations are supported:

- addUser
- removeUser
- getUsers

This interface is utilized by UserManager class. All usernames are cached for better performance, so getUsers operation is called only once within constructor.

The component contains one implementation of this interface - UserPersistenceImpl which provides database backend as persistence storage.

UserPersistenceImpl

UserPersistence implementations provide database backend for storing and retrieving imported users. The class utilizes DB Connection Factory to support various database systems.

Used database table is "Users" and described by Requirement Specification. Note, that CreationDate, CreationUser, ModificationDate, ModificationUser fields are out of scope of this component (as noted in Developer Forums).

Following basic operations are supported:

- addUser
- removeUser
- getUsers

1.5 Component Exception Definitions

ConfigurationException

An exception is thrown by UserManager , UserStoreManagerImpl , DbUserStore , UserPersistenceImpl being initialized from a parameters provided by a configuration namespace if any error occurs while reading the configuration parameters, instantiating and initializing the inner

fioelds. Usually this exception will wrap the original exception which caused the operation to fail. Such a cause may be obtained through method.

PersistenceException

An exception is thrown by all classes of this component if some error occurs during persistence (loading or saving data to the data source). Usually this exception will wrap the original exception which caused the operation to fail. Such a cause may be obtained through method.

UnknownUserException

An exception is thrown by UserManager if given username doesn't exist within internal storage or user store (UserStore implementation)

UnknownUserStoreException

An exception is thrown by UserManager if user contains unknown user store name (which doesn't present in UserStoreManager)

1.6 Thread Safety

This component is not thread-safe because it’s not required. The user should provide own UserStore, UserPersistence, AuthorizationPersistence implementations and UserManager wrapper to provide thread-safe functionality.

2. Environment Requirements

2.1 Environment

- Development language: Java 1.4
- Compile target: Java 1.3, Java 1.4

2.2 TopCoder Software Components

Configuration Manager latest version

It is used in the component to load configuration values.

DB Connection Factory 1.0

Manages db connections within the component

Authentication Factory 2.0

API is used to provide authentication within the component.

Authorization 2.0

Used to manage and persist user roles

ID Generator 3.0

Used to generate unique ID for imported user

Base Exception 1.0

Used as the base class for custom exceptions

2.3 Third Party Components

None

3. Installation and Configuration

3.1 Package Name

com.topcoder.timetracker.user

3.2 Configuration Parameters

All used components should be properly configured within “com.topcoder.timetracker.user” namespace. ID Generator should contain sequence for “TimeTrackerUser” name.

Own configuration parameters are following

Parameter	Description	Values
connection_name	Connection name within DB Connection Factory that will be used for user persistence	String
UserStoreNames	Names of user stores to register within UserStoreManagerImpl	String[]
ClassName.XX	Class name of user store instance with name XX	com.topcoder.timetracker.user.DbUserStore
ConnectionString.XX	Represents specific connection information used to create UserStore instance via reflection	String

3.3 Dependencies Configuration

Informix database system should be installed and configured within DB Connection Factory component

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

Preload the configuration file into Configuration Manager. Follow demo.

4.3 Demo

```
package com.topcoder.timetracker.user;

import java.util.Collection;
import java.util.Iterator;

import com.topcoder.security.authenticationfactory.Response;
import com.topcoder.security.authorization.Action;
import com.topcoder.security.authorization.ActionContext;
import com.topcoder.security.authorization.persistence.GeneralAction;
import com.topcoder.security.authorization.persistence.GeneralActionContext;
import com.topcoder.security.authorization.persistence.GeneralSecurityRole;

/**
 * Demo class demonstrates usage of the Time Tracker User component
 */
public class Demo {

        public static void main(String[] args) {
                try {
                        // create user manager
                        UserManager userManager = new UserManager();

                        // connect and add varius database user stores
                       // (db1, db2 are connection names from DB Connection Factory component)
                        userManager.getUserStores().add("db1", new DbUserStore("db1") );

                        userManager.getUserStores().add("db2",  new DbUserStore("db2") );
```

```
                        // get db2 user store and search records
                        Iterator users =
userManager.getUserStores().getUserStore("db2").search("test%").iterator();
                        while (users.hasNext()){
                                userManager.importUser(users.next().toString(), "db2");
                        }
                        // get all imported user names
                        Collection usernames = userManager.getNames();

                        // assign SUPER ADMIN role for test1
                        userManager.setUserRole("test1", UserManager.SUPER_ADMIN);

                        // assign EMPLOYEE role for test2
                        userManager.setUserRole("test2", UserManager.EMPLOYEE);

                        //      add new role
                        userManager.getAuthPersistence().addRole(new GeneralSecurityRole(7,
"CustomRole"));

                        // get all roles
                       Collection roles = userManager.getAuthPersistence().getSecurityRoles();

                        // authenticate "test1"
                        Response response = userManager.authenticate("test1", "password1");
                        System.out.println("Result : "+response.isSuccessful());

                        // get user instance
                        User user = userManager.getUser("test1");

                        // use authorization component againt this user (future usage of this
component)
                        Action action = new GeneralAction(1, "base action");
                        ActionContext actionContext = new GeneralActionContext(1, "base action
context", null);
                        userManager.getAuthPersistence().addAction(action);
                        userManager.getAuthPersistence().addActionContext(actionContext);

                        // check permissions
                        int result =
userManager.getAuthPersistence().checkPermissionForPrincipal(user, action, actionContext);

                } catch (Exception e){
                        e.printStackTrace();
                }
        }
}
```

5. Future Enhancements

None