Time Tracker User v2.0 Component Specification

1. Design

The Time Tracker User custom component is part of the Time Tracker application. It provides an abstraction of user accounts and company accounts in the system. This component handles the persistence and other business logic required by the application.

The new version introduces the concept of "companies", where each user will be associated with a company. User account information will now be stored directly in the database.

The component consists of a set of Data Access Objects (DAOs) which are used to manipulate the various entities in the Time Tracker system. A DAO exists for the Company, User, RejectReason and RejectEmail. Each DAO provides Create, Retrieve, Update, Delete, Enumerate and Search methods for their respective Time Tracker entity.

In addition, since User management is a bit more complex, a UserManager is also included that acts as a facade and facilitates the usage of the Authentication and Authorization components within the scope of the Time Tracker system.

The actual Time Tracker entities are modeled as data beans that follow the Javabean naming convention. They are also Serializable for easy persistence and network transfer.

Design Issues

Id Generation and separate DAOs:

The component user and developers should note that in the current database schema, certain tables are shared between different entities. For example, both User and Company share the Address table. Since each Address entity needs to maintain id uniqueness, it is important that **both Company and User DAOs need to share the same ID Generator**. It is simply a matter of specifying the same id generator name for both DAOs, but the developer should document this for the end-user.

Updating the Modification Date and User:

Some Time Tracker Entities like User and Company are composed of different entities. For example, User contains both a Contact and an Address. Each entity has its own Modification Date and User. When a modification is made to either the containing entity (the User), or one of the composing entities (Contact or Address), the question remains as to which table's modification_date and modification_user columns need to be updated. This is resolved by adding an additional class variable *changed* to the TimeTrackerBean. This allows the updating DAO to detect where changes have been made and update the columns according to a set of rules. Those rules are outlined in the Algorithm section.

1.1 Design Patterns

Facade is used by the UserManager by aggregating and simplifying all the User-related functionality into a single class.

Strategy is used by the UserManager by utilizing a pluggable system of delegating to the UserDAO, AbstractAuthenticator and AuthorizationPersistence.

DataAccessObject pattern is used by UserDAO, CompanyDAO, RejectReasonDAO and RejectEmailDAO.

1.2 Industry Standards

JDBC

1.3 Required Algorithms

1.3.1 SQL Statements

The implementation notes for the DAO methods contain details on the SQL statements that need to be executed, as well as general notes on how the Transactions should be processed in atomic batch mode, or non-atomic batch mode. The developer is expected to have enough basic knowledge of JDBC to be able to implement the needed functionality.

1.3.2 Modification User and Date

When a DAO is performing an update on a Time Tracker entity, the following procedure may be followed:

- If the entity is simple (it does not contain any other entities) like RejectEmail or RejectReason, then perform the update only if the *changed* variable is *true*. After performing the update successfully, update the *changed* variable to *false*. The modification user and date are the provided username and current date respectively.

  Example:

```
RejectReason rejectReason = rejectReasonDAO.retrieve(id);

// Setting the description should trigger the changed to become true.
rejectReason.setDescription("A different description");

// This should result in an update, since setDescription was changed.
// The changed variable should be updated to false after the update.
rejectReasonDAO.update(rejectReason, "adminUser");

// Setting the description to the same String will not trigger changed
// to become true.
rejectReason.setDescription("A different description");


// This will not update the table.
rejectReasonDAO.update(rejectReason, "adminUser");
```

- If the entity is composite (it contains other entieis) like User or Company, then:
  - Update the table corresponding to the main entity if its *changed* variable is *true* OR any of its composing entities *changed* variable is *true.*
  - The composing entities do not need to be updated unless their own *changed* variable is true also.

Example:

```
User user = userDAO.retrieve(id);

// We will simply change a direct attribute of the user
user.setPassword(“newPassword”);


// This should result in an update to only the user_account table.
// The address or contact tables are not updated.  Because those entities
// were not modified.
userDAO.update(user, “adminUser”);
```

Example #2:

```
User user = userDAO.retrieve(id);

// If we modify only one of the composing entities of the User.
Address addy = user.getAddress();
addy.setLine1(“10737 SandBox Ave.”);
addy.setLine2(“Palmer Hills”);

// attempt to update the user will result in an update both to the
// user_account table and the address table.  The contact table is not
// modified because it was not modified.  Note that user_account is
// still modified because it is the containing entity of the address.
userDAO.update(user, “adminUser”);
```

Example #3:

```
User user = userDAO.retrieve(id);

// If we modify both of the composing entities of the User.
Address addy = user.getAddress();
addy.setLine1(“10737 SandBox Ave.”);
addy.setLine2(“Palmer Hills”);

Contact contact = user.getContact();
contact.setPhoneNumber(“123-3456”);

// attempt to update the user will result in an update both to the
// user_account table, the address table and the contact table, because
// all entities are changed.
userDAO.update(user, “adminUser”);
```

1.4 Component Class Overview

[com.cronos.timetracker.common]

TimeTrackerBean

This is a Base class to represent all entities of the Time Tracker User v2.0 component. It contains all the properties which are common to these entities. It is also Serializable to support network transfer, and state persistence.

Address

This bean represents a possible Address where a different entity may be situated in. It may represent the address of a User or Company.

State

This bean represents a State where an Address could be located. Some examples could be the state of Texas, Illinois, etc.

Contact

This bean represents the contact details of a specific entity. It may include the name of the contact person, or the name of the person itself. Other details on how to get in touch like phone number and email are also included.

RejectEmail

This bean represents an email template that is sent to the contractor when a time or expense entry has been rejected by the Project Manager. Each company has a different email template.

RejectEmailDAO [interface]

This interface defines the necessary methods that a RejectEmail DAO should support. Create, Retrieve, Update, Delete and Enumerate (CRUDE) methods are provided. There is also a search method that utilizes Filter classes from the Search Builder 1.2 component.

DbRejectEmailDAO

This is a Database implementation of the RejectEmailDAO interface. It is capable of persisting and retrieving RejectReason information from the database. Create, Retrieve, Update, Delete and Enumerate (CRUDE) methods are provided. There is also a search method that utilizes Filter classes from the Search Builder 1.2 component.

RejectReason

This bean represents a possible reason why a time or expense entry may be rejected by the Project Manager. Each company has a different set of policies on which entries will be rejected, so each RejectReason is associated with a company.

RejectReasonDAO [interface]

This interface defines the necessary methods that a RejectReason DAO should support. Create, Retrieve, Update, Delete and Enumerate (CRUDE) methods are provided. There is also a search method that utilizes Filter classes from the Search Builder 1.2 component.

DbRejectReasonDAO

This is a Database implementation of the RejectReasonDAO interface. It is capable of persisting and retrieving RejectReason information from the database. Create, Retrieve, Update, Delete and Enumerate (CRUDE) methods are provided. There is also a search method that utilizes Filter classes from the Search Builder 1.2 component.

RejectReasonSearchBuilder

This is a convenience class that may be used to build filters for performing searches in the RejectReasonDAO. Users may call the different methods to set the criteria for the filter and finally retrieve the filter for use via the buildFilter() method.

RejectEmailSearchBuilder

This is a convenience class that may be used to build filters for performing searches in the RejectEmailDAO. Users may call the different methods to set the criteria for the filter and finally retrieve the filter for use via the buildFilter() method.

[com.cronos.timetracker.user]

User

This bean contains the account information of a Time Tracker user. The User password is stored within the bean in encrypted form.

UserDAO [interface]

This interface defines the necessary methods that a User DAO should support. Create, Retrieve, Update, Delete and Enumerate (CRUDE) methods, and their respective batch-mode equivalents are specified. There is also a search method that utilizes Filter classes from the Search Builder 1.2 component.

DbUserDAO

This is a Database implementation of the UserDAO interface. It is capable of persisting and retrieving Time Tracker User information from the database. The DAO is capable of performing the operations on multiple Users simultaneously (batch mode). While in batch mode, the operation can be done atomically, or separately. If done atomically, then a failure at any one of the specified entries will mean that the entire batch will be rolled back. Otherwise, only the user where a failure occurred will be rolled back.

UserManager

This class is a facade that provides access to all functions related to a Time Tracker User. It provides CRUDE methods to manage the users within the persistent store. It works with the Authorization Manager component to manage the Authorization Roles for a user. And finally, it is also capable of authenticating a User through a username/password combination.

DbUserAuthenticator

An authenticator implementation that is capable of authenticating a username/password combination against the Time Tracker datastore. It will connect to the specified data store and verify that the username/password combination is valid. It will automatically perform any necessary encryption/decryption that is needed when interacting with the data store.

UserSearchBuilder

This is a convenience class that may be used to build filters for performing searches in the UserDAO. Users may call the different methods to set the criteria for the filter and finally retrieve the filter for use via the buildFilter() method.

[com.cronos.timetracker.company]

Company

This bean represents a Company within the context of the Time Tracker component. It holds the different attributes of the company such as the company name, address and contact information. The Company passcode is also stored within in encrypted form.

CompanyDAO [interface]

This interface defines the necessary methods that a Company DAO should support. Create, Retrieve, Update, Deleteand Enumerate (CRUDE) methods, and their respective batch-mode equivalents are specified. There is also a search method that utilizes Filter classes from the Search Builder 1.2 component.

DbCompanyDAO

This is a Database implementation of the CompanyDAO interface. It is capable ofpersisting and retrieving company information from the database. The DAO is capable of performing the operations on multiple Companies simultaneously (batch mode). While in batch mode, the operation can be done atomically, or separately. If done atomically, then a failure at any one of the specified entries will mean that the entire batch will be rolled back. Otherwise, only the company where a failure occurred will be rolled back.

CompanySearchBuilder

This is a convenience class that may be used to build filters for performing searches in the CompanyDAO. Users may call the different methods to set the criteria for the filter and finally retrieve the filter for use via the buildFilter() method.

1.5 Component Exception Definitions

New Custom Exceptions

[com.cronos.timetracker.common]

RejectEmailDAOException

This exception is thrown by the RejectEmailDAOs when a problem occurs while accessing the data store.

RejectEmailNotFoundException

This exception is thrown if the involved RejectEmail was not found in the datastore during a DAO operation that required it to be present.

RejectFilterDAOException

This exception is thrown by the RejectFilterDAOs when a problem occurs while accessing the data store.

RejectFilterNotFoundException

This exception is thrown if the involved RejectFilter was not found in the datastore during a DAO operation that required it to be present.

[com.cronos.timetracker.company]

CompanyDAOException

This exception is thrown by the CompanyDAOs when a problem occurs while accessing the data store.

BatchCompanyDAOException

This exception is thrown by the CompanyDAOs when a problem occurs while accessing the data store in non-atomic batch mode. It contains an array of causes and problemCompanys, so that the calling application can keep track of which beans a problem occurred in and their respective causes.

CompanyNotFoundException

This exception is thrown if the involved Company was not found in the datastore during a DAO operation that required it to be present.

[com.cronos.timetracker.user]

UserDAOException

This exception is thrown by the UserDAOs when a problem occurs while accessing the data store.

BatchUserDAOException

This exception is thrown by the UserDAOs when a problem occurs while accessing the data store in non-atomic batch mode. It contains an array of causes and problemUsers, so that the calling application can keep track of which beans a problem occurred in and their respective causes.

UserNotFoundException

This exception is thrown if the involved User was not found in the datastore during a DAO operation that required it to be present.

1.6 Thread Safety

This component is not entirely thread safe, but may be used safely in a thread safe environment.

The DAO implementations are immutable, and their thread safety lies in maintaining a thread-safe access to the data store. This can be done by the use of database transactions for the database DAO implementations.

The UserManager's thread-safety is dependent on the underlying AuthorizationPersistence, AbstractAuthenticator and UserDAO implementations, all of which should generally be thread-safe.

The beans themselves are not thread-safe, but each thread is expected to work on a separate instance of the bean. A single instance of a bean should not be read/modified concurrently by multiple threads.

2. Environment Requirements

2.1 Environment

- Java 1.4

2.2 TopCoder Software Components

- **Configuration Manager 2.1.4** is used to configure the component.
- **ID Generator 3.0** is used to generate unique long ids for the Time Tracker data objects that are created in the persistence layer.
- **DB Connection Factory 1.0** is used to configure and create database connections.
- **Base Exception 1.0** is used as base class for component's exceptions
- **Search Builder 1.2** is used to provide Search filtering functionality to the database DAO implementations.
- **Encryption 1.0** is used to encrypt sensitive information such as the Company passcode and User password.

- **Authentication Factory 2.0** is used as the authentication framework for authenticating the Time Tracker Users.
- **Authorization 2.1.1** is used as the Authorization framework for managing a Time Tracker User's possible Roles and authorized Actions.

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

- There are no direct 3rd party dependencies. See the 3rd party component dependencies of the components that this component is based on for indirect dependencies.

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.cronos.timetracker.user
com.cronos.timetracker.company
com.cronos.timetracker.common

3.2 Configuration Parameters

Note that this design has been made with the intent of having a majority of the classes configured via the **TC Object Factory 2.0** component. All classes which may need configuration have been defined with an appropriate constructor that the Object Factory may utilize.

To maintain consistency with Authentication Factory, the DbUserAuthenticator still supports direct configuration via ConfigManager. The following table lists the configuration parameters:

Configuration for **DbUserAuthenticator**:

Parameter	Description	Value
principal_key_converter	Contains class/namespace sub properties to create the converter. (Property is supported via inheritance to AbstractAuthenticator) **Optional**.	none (will contain subproperties)
principal_key_converter .class	Full qualified class name for the converter. (Property is supported via inheritance to AbstractAuthenticator) **Required**.	DefaultKeyConverter
principal_key_converter.namespace	The namespace for converter's constructor.	com.DefaultConverter

	Required.	
cache_factory	Contains class/namespace sub properties to create the converter. (Property is supported via inheritance to AbstractAuthenticator) **Optional**.	none (will contain subproperties)
cache_factory.class	Full qualified class name for the cache factory. (Property is supported via inheritance to AbstractAuthenticator) **Required**.	TimeoutCacheFactory
cache_factory.namespace	The namespace for cache factory's constructor. (Property is supported via inheritance to AbstractAuthenticator) **Required**.	com.TimeoutFactory
connection_factory	The namespace for the DbConnectionFactoryImpl. **Required**.	none (will contain subproperties)
connection_factory.classname	This property represents the class name for the connection factory implementation. **Required**.	com.topcoder.db.connectionfactory
connection_factory.namespace	This property represents the namespace for the application complete handler. **Required**.	com.topcoder.db.connectionfactory
connection_factory.connection_name	This property contains the connection name used in acquiring a connection to the factory. If not specified, then the default connection is used. **Optional.**	OracleConnection
encryption_algorithm	The property contains the encryption algorithm name to be used when processing the password. **Required.**	DES

3.3 Dependencies Configuration

The dependency components should be configured according to their documentation.

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

Follow the “Demo”.

4.3 Demo

The Following Configuration File May be used for the Object Factory 2.0 Component:

```
<!-- Property defining a specification for constructing a Time Tracker User DAO -->
<property name="TimeTrackerUserDAO">
    <property name="type">
        <value>com.cronos.timetracker.user.DbUserDAO</value>
    </property>
    <property name="params">

        <!-- Parameter corresponds to connection factory.  Should be specified
             as another Object Factory specification.See TimeTrackerConnectionFactory. -->
        <property name="param1">
             <property name="type">
                 <value>com.topcoder.db.connectionfactory.DBConnectionFactory</value>
             </property>
             <property name="name">
                 <value>TimeTrackerConnectionFactory</value>
             </property>
        </property>


        <!-- Parameter corresponds to Connection Name. -->
        <property name="param2">
             <property name="type">
                  <value>java.lang.String</value>
             </property>
            <property name="value">
                  <value>mySQL Connection</value>
             </property>
         </property>



        <!-- Parameter corresponds to Algorithm Name. -->
         <property name="param3">
             <property name="type">
                  <value>java.lang.String</value>
             </property>
            <property name="value">
                  <value>DES</value>
             </property>
         </property>


        <!-- Parameter corresponds to Id Generator Name. -->
         <property name="param4">
             <property name="type">
                  <value>java.lang.String</value>
             </property>
            <property name="value">
                  <value>idgeneratorname</value>
             </property>
         </property>
    </property>
</property>



<property name="TimeTrackerCompanyDAO">
    <property name="type">
        <value>com.cronos.timetracker.company.DbCompanyDAO</value>
    </property>
    <property name="params">

       <!-- Parameter corresponds to connection factory.  Should be specified
             as another Object Factory specification. See TimeTrackerConnectionFactory. -->
        <property name="param1">
             <property name="type">
                 <value>com.topcoder.db.connectionfactory.DBConnectionFactory</value>
```

```
            </property>
            <property name="name">
                <value>TimeTrackerConnectionFactory</value>
            </property>
        </property>


        <!-- Parameter corresponds to Connection Name. -->
        <property name="param2">
            <property name="type">
                <value>java.lang.String</value>
            </property>
           <property name="value">
                <value>mySQL Connection</value>
            </property>
         </property>



        <!-- Parameter corresponds to Algorithm Name. -->
         <property name="param3">
            <property name="type">
                <value>java.lang.String</value>
            </property>
           <property name="value">
                <value>DES</value>
            </property>
         </property>


        <!-- Parameter corresponds to Id Generator Name. -->
         <property name="param4">
            <property name="type">
                <value>java.lang.String</value>
            </property>
           <property name="value">
                <value>idgeneratorname</value>
            </property>
         </property>
    </property>
</property>



<property name="TimeTrackerRejectReasonDAO">
    <property name="type">
        <value>com.cronos.timetracker.common.DbRejectReasonDAO</value>
    </property>
    <property name="params">

        <!-- Parameter corresponds to connection factory. Should be specified
             as another Object Factory specification.See TimeTrackerConnectionFactory. -->

        <property name="param1">
            <property name="type">
                <value>com.topcoder.db.connectionfactory.DBConnectionFactory</value>
            </property>
            <property name="name">
                <value>TimeTrackerConnectionFactory</value>
            </property>
        </property>


        <!-- Parameter corresponds to Connection Name. -->
        <property name="param2">
            <property name="type">
                <value>java.lang.String</value>
            </property>
           <property name="value">
                <value>mySQL Connection</value>
            </property>
         </property>


        <!-- Parameter corresponds to Id Generator Name. -->
         <property name="param3">
            <property name="type">
                <value>java.lang.String</value>
            </property>
           <property name="value">
```

```
                <value>idgeneratorname</value>
            </property>
        </property>
    </property>
</property>


<property name="TimeTrackerRejectEmailDAO">
    <property name="type">
        <value>com.cronos.timetracker.common.DbRejectEmailDAO</value>
    </property>
    <property name="params">

        <!-- Parameter corresponds to connection factory.  Should be specified
             as another Object Factory specification. See
TimeTrackerConnectionFactory. -->
        <property name="param1">
             <property name="type">
<value>com.topcoder.db.connectionfactory.DBConnectionFactory</value>
             </property>
             <property name="name">
                 <value>TimeTrackerConnectionFactory</value>
             </property>
        </property>


        <!-- Parameter corresponds to Connection Name. -->
        <property name="param2">
             <property name="type">
                  <value>java.lang.String</value>
             </property>
            <property name="value">
                  <value>mySQL Connection</value>
             </property>
         </property>


        <!-- Parameter corresponds to Id Generator Name. -->
         <property name="param3">
             <property name="type">
                  <value>java.lang.String</value>
             </property>
            <property name="value">
                  <value>idgeneratorname</value>
             </property>
         </property>
    </property>
</property>



<!-- Property for specifying the Connection Factory that is used by the Time Tracker DAOs. -->
<property name="TimeTrackerConnectionFactory">
     <property name="type">
         <value>com.topcoder.db.connectionfactory.DBConnectionFactoryImpl</value>
     </property>
     <property name="params">
         <property name="param1">
             <property name="type">
                  <value>java.lang.String</value>
             </property>
            <property name="value">
                  <value>com.cronos.timetracker.configNamespace</value>
             </property>
         </property>
     </property>
</property>


// Try-catch clauses have been removed for clarity.

// 1. A simple demonstration of Time Tracker User management and searching.
UserDAO userDAO = new DbUserDAO(connFactory, connName, algoName, true, idGenName);
DbUserAuthenticator authenticator = new DbUserAuthenticator(demoNamespace);
AuthorizationPersistence authPersistence = authorizationManager.getPersistence();

UserManager manager = new UserManager(userDAO, authenticator, authPersistence);

User userToCreate = new User();
```

```
// algorithm name needs to be set for proper encoding of password within the bean.
userToCreate.setAlgorithmName(“DES”);

userToCreate.setUserName(“newUser”);
userToCreate.setPassword(“pwd”);


Address addy = new Address();
addy.setLine1(“Palm Street”);
addy.setLine2(“Maple Drive”);
addy.setCity(“Florida City”);
addy.setState(floridaState);


Contact contact = new Contact();
contact.setFirstName(“Mr.”);
contact.setLastName(“User”);
contact.setPhoneNumber(“555-5555”);
contact.setEmail(“user@user.com”);



userToCreate.setContact(contact);
userToCreate.setAddress(addy);
userToCreate.setAccountStatus(activeStatus);


// Authorization should now have a Principal with the given user.
manager.createUser(userToCreate, “adminUser”);

//userToCreate should now have an assigned id
userToCreate.getId();

//Give the newly created user manager privileges.
manager.addRoleForUser(userToCreate, managerRole);

//List all roles for given user.
SecurityRole[] userRoles = manager.getRolesForUser(userToCreate);

// User wishes to log in.  Authenticate the user.
manager.authenticateUser(username, password);

// Search for all users with whose first name starts with “J”

// First construct a filter for it:
Filter searchFilter = new LikeFilter(“contact.first_name”, “J%”);

// Now the manager can be used to search:
User[] results = manager.search(searchFilter);

// Another way to do it would be to use a Search Builder:
UserSearchBuilder builder = new UserSearchBuilder();
builder.hasFirstName(“J%”);
builder.hasLastName(“S%”);
builder.hasCity(“New York”);


searchFilter = builder.builderFilter();

// Now the manager can be used to search:
User[] results = manager.search(searchFilter);


// All users in New York have relocated to Old York
for (int x = 0; x < results.length; x++) {
     results[x].setCity(“Old York”);
}

// Perform a batch update on those users.
manager.updateUsers(results, true);
```

```
// The builder can be reset to perform additional searches.
builder.reset();


// 2. Manipulating Time Tracker Companies
Company company = new Company();
company.setAlgorithmName(“DES”);

Address compAddy = new Address();
addy.setLine1(“Palm Street”);
addy.setLine2(“Maple Drive”);
addy.setCity(“Florida City”);
addy.setState(floridaState);

company.setAddress(compAddy);
company.setPasscode(“44tkdi882sz”);
company.setContact(contact);


// The Object Factory 2.0 may be used to create the DAOs to have the advantages of
// a factory for the DAOs.
ObjectFactory daoFactory = new ObjectFactory(new
ConfigManagerSpecificationFactory(“com.cronos.timetracker.user”);
CompanyDAO compDAO = (CompanyDAO)
     daoFactory.createObject(“TimeTrackerCompanyDAO”);

compDAO.createCompany(company);

compDAO.deleteCompany(company);


// 3. Manipulating Reject Reasons
RejectReason reason = new RejectReason();
reason.setDescription(“Insufficient Funds”);
reason.setCompanyId(company.getId());
reason.setActive(true);


RejectReasonDAO reasonDAO = (RejectReasonDAO)
     daoFacotry.createObject(“TimeTrackerRejectReasonDAO”);
reasonDAO.createRejectReason(reason);

RejectReasonSearchBuilder reasonBuilder = new RejectReasonSearchBuilder();

reasonBuilder.hasDescription(“Verbose”);
Filter reasonFilter = reasonBuilder.buildFilter();

RejectReason[] reasonResults = reasonDAO.search(reasonFilter);


reasonDAO.deleteRejectReasons(reasonResults, true);


// 4. Manipulating Reject Emails

RejectEmail email = new RejectEmail();
email.setBody(“Your entry has been rejected because...”);
email.setCompanyId(company.getId());

RejectEmailDAO emailDAO = (RejectEmailDAO)
     daoFactory.createObject(“TimeTrackerRejectEmailDAO”);

emailDAO.createRejectEmail(email);
```

5. Future Enhancements

None.