[TopCoder]

Hera Proxied Email Manager 1.0 Component Specification

1. Design

The proxied email service is a basic email server facade that will allow for the actual email addresses of consumers to be anonymous when Hera partners wish to send emails. Consumers come to the site and fill out qualification forms to connect with Hera partners for a number of services. The by-product of this interaction is called a Lead. The client wishes to protect consumer privacy by masking consumer email addresses to partners but without disrupting normal email usage paradigms. Partners will need to communicate with consumers and vice versa. However the client wants to centrally route the email through a common gateway. Long term this central routing would allow features such as: charge partners per communication or to communicate at all and detect partners that violate privacy agreements by selling contact information to 3rd parties. This component provides a manager class with methods for generation of proxy addresses, CRUD methods for access of real/proxy addresses and database logging service. Also this component encapsulates all access to the database.

1.1 Design Patterns

Strategy pattern – HeraProxiedEmailManager uses pluggable instances of IProxiedEmailPersistence and IProxyEmailGenerator implementations as strategies.

DTO/DAO pattern – HeraProxiedEmailManager and IProxiedEmailPersistence are DAOs for EmailLog and EmailUserParams DTOs.

1.2 Industry Standards

T-SQL, ADO.NET, XML

1.3 Required Algorithms

1.3.1 Logging

This component must perform logging in all public methods of HeraProxiedEmailManager.

All information must be logged using logger:Logger attribute. If logger attribute is null, then logging is not required to be performed.

In all mentioned methods method entrance with input argument, method exit with return value and call duration time must be logged at DEBUG level.

All errors (for all thrown exceptions) must be logged at ERROR level with exception message and stack trace.

1.3.2 Calling stored procedures

In this component stored procedures are called with use of Data Access Interface component. ProxiedEmailPersistence uses dataAccess:IDataAccess attribute for this. First, database connection must be created:

```
IDbConnection connection = dataAccess.CreatePredefinedConnection(connectionName);
```

Next IDataParameter instance must be created for each SP parameter and properly filled.

```
IDataParameter parameter = dataAccess.CreateDataParameter(...);
parameter.Name = ...
parameter.Value = ...
```

All parameters must be put to parameters:IDataParameter[]. Then one of the following methods must be called:

```
string spName = ... // name of the stored procedure, see titles of 1.3.3.X sections

dataAccess.ExecuteNonQuery(connection, CommandType.StoredProcedure, spName, parameters);
// or
IDataReader reader =
    dataAccess.ExecuteReader(connection, CommandType.StoredProcedure, spName, parameters);
```

If ExecuteReader() was used, then result data is read with use of reader:IDataReader. See method docs for details.

Finally connection must be closed:

```
connection.Close();
```

1.3.3 Stored procedures

This component must access data in the database via stored procedures only. Thus developers must provide implementations for all stored procedures mentioned in this section. Names of subsections indicate names of stored procedures.

All SP parameter names are provided in subsections as an example only. Developers can use any SP parameter names.

It's assumed that ON DELETE CASCADE option is used for all foreign keys, and EmailLog#TimeStamp has a default value (equal to the current time).

SP parameters indicating the record to be updated/retrieved/deleted/used

All SPs in which the caller can indicate the usage of an existing UserEmailMap record must accept 3 SP parameters for each such record: @UserId, @RealEmail and @ProxyEmail. The requirement is that exactly one of these 3 parameters must be not null, two others must be null. Thus the corresponding check of input parameters must be done in SPs (raise an error associated with ProxiedEmailPersistenceException if such check fails). Next to match the required UserEmailMap record the following WHERE clause can be used:

```
WHERE ((@UserId IS NULL OR UserId = @UserId) AND
       (@RealEmail IS NULL OR RealEmail = @RealEmail) AND
       (@ProxyEmail IS NULL OR ProxyEmail = @ProxyEmail))
```

NOTE: All stored procedures that modify data in the database must use transactions.

1.3.3.1 spCreateEmailUser

This stored procedure creates a record in UserEmailMap and UserHistoryEnabled tables. The default value for HistoryEnabled field must be used (TRUE).

INPUT

User ID, real and proxy email addresses, blocked flag

RAISES ERROR that corresponds to the following exception:

- ProxiedEmailPersistenceException (when the provided user ID or email addresses are not unique in the database; this uniqueness check is performed by DB)

1.3.3.2 spUpdateEmailUser

This stored procedure updates a record in UserEmailMap table.

INPUT

User ID, real and proxy email addresses of existing record (only one of 3 parameters should be specified)

Updated user ID, real and proxy email addresses, blocked flag

RAISES ERROR that corresponds to the following exception:

- EmailUserNotFoundException (when email user record with the given parameters doesn't exist in DB)
- ProxiedEmailPersistenceException (when the provided user ID or email addresses are not unique in the database; this uniqueness check is performed by DB)

1.3.3.3 spGetEmailUser

This stored procedure retrieves a record from UserEmailMap table.

INPUT

User ID, real and proxy email addresses of existing record (only one of 3 parameters should be specified)

Updated user ID, real and proxy email addresses, blocked flag

OUTPUT

Empty dataset if email user with the specified parameters is not found. Otherwise one row with user ID, read/proxy addresses and block flag

1.3.3.4 spDeleteEmailUser

This stored procedure deletes a record from UserEmailMap table (and automatically from all other tables by foreign key).

INPUT

User ID, real and proxy email addresses of existing record (only one of 3 parameters should be specified)

RAISES ERROR that corresponds to the following exception:

- EmailUserNotFoundException (when email user record with the given parameters doesn't exist in DB)

1.3.3.5 spCreateEmailLog

This stored procedure creates a record in EmailLog table, and more records in EmailRecipients and EmailBccRecipients tables. Before inserting records to EmailRecipients and EmailBccRecipients tables proxy addresses from XML strings must be converted to primary ID values of UserEmailMap table (don't confuse UserEmailMapId and UserId fields).

INPUT

Sender address, XML string with "To" recipient proxy addresses, XML string with "BCC" recipient proxy addresses, subject and body

See details in implementation notes of ProxiedEmailPersistence#CreateEmailLog().

RAISES ERROR that corresponds to the following exception:

- ProxiedEmailPersistenceException (when the provided user ID or email addresses are not unique in the database; this uniqueness check is performed by DB)

1.3.3.6 spSetUsersBlock

This stored procedure creates or deletes a record in/from BlockingMap table. When @enabled=true, it's required to check whether a record already exists in BlockingMap.

INPUT

User ID, real and proxy email addresses of user to be blocked/unblocked (only one of 3 parameters should be specified)

User ID, real and proxy email addresses of the blocking/unblocking user (only one of 3 parameters should be specified)

Flag indicating whether blocking or unblocking is performed

RAISES ERROR that corresponds to the following exception:

- EmailUserNotFoundException (when email user record with the given parameters doesn't exist in DB)

1.3.3.7 spGetUserBlocked

This stored procedure checks whether a record exists in BlockingMap table. Additionally it checks Blocked field in UserEmailMap table. Double joining of UserEmailMap table must be used.

INPUT

User ID, real and proxy email addresses of the (possibly) blocked user (only one of 3 parameters should be specified)

User ID, real and proxy email addresses of the blocking user (only one of 3 parameters should be specified)

OUTPUT

0 if not blocked (Blocked is 0 and there is no record in BlockingMap), 1 if blocked

RAISES ERROR that corresponds to the following exception:

- EmailUserNotFoundException (when email user record with the given parameters doesn't exist in DB)

1.3.3.8 spSearchEmailLogs

This stored procedure retrieves records from EmailLog table that match the given parameters. Validation of recipient IDs specified in XML strings is not required.

INPUT

From user, subject, recipient IDs XML, BCC recipient IDs XML and timestamp search parameters

See details in implementation notes of ProxiedEmailPersistence#SearchEmailLogs().

OUTPUT

1 dataset contains matched records from EmailLog table.

Next for each record in the first returned dataset go the following datasets:

- Dataset with the list of UserEmailMap#ProxyEmail fields associated with the record ofEmailLog via EmailRecipients table.
- Dataset with the list of UserEmailMap#ProxyEmail fields associated with the record ofEmailLog via EmailBccRecipients table.

IMPLEMENTATION NOTES

- Parse TO and BCC recipient proxy addressed from the input XML strings.
- SELECT records from EmailLog (use DISTINCT search). Additionally join UserEmailMap (by FromUser), UserEmailMap via EmailRecipients and UserEmailMap via EmailBccRecipients. In WHERE clause use the following check:

```
WHERE ((@FromUser IS NULL OR FromUser = @FromUser) AND
       (@Subject IS NULL OR Subject = @Subject) AND
       (@TimeStamp IS NULL OR TimeStamp = @TimeStamp) AND
       (@RecipientId IS NULL OR U1.UserId IN @RecipientId) AND
       (@BccRecipientId IS NULL OR U2.UserId IN @BccRecipientId))
```

- For each record returned by the previous query do:
  - SELECT ProxyEmail from UserEmailMap by joining it with EmailLog via EmailRecipients. In WHERE clause check EmailLogId field.
  - SELECT ProxyEmail from UserEmailMap by joining it with EmailLog via EmailBccRecipients. In WHERE clause check EmailLogId field.

1.3.3.9 spChangeLoggingHistoryEnabled

This stored procedure updates a record in UserHistoryEnabled table.

INPUT

User ID, real and proxy email addresses of the user (only one of 3 parameters should be specified)

Flag indicating whether logging history should be enabled or disabled

RAISES ERROR that corresponds to the following exception:

- EmailUserNotFoundException (when email user record with the given parameters doesn't exist in DB)
- ProxiedEmailPersistenceException (when UserHistoryEnabled record doesn't contain a record associated with UserEmailMap#UserId)

1.3.3.10 spGetLoggingHistoryEnabled

This stored procedure retrieves HistoryEnabled field from UserHistoryEnabled table. UserEmailMap table must be joined.

INPUT

User ID, real and proxy email addresses of the user (only one of 3 parameters should be specified)

OUTPUT

1 if enabled, 0 if disabled

RAISES ERROR that corresponds to the following exception:

- EmailUserNotFoundException (when email user record with the given parameters doesn't exist in DB)
- ProxiedEmailPersistenceException (when UserHistoryEnabled record doesn't contain a record associated with UserEmailMap#UserId)

1.4 Component Class Overview

EmailLog

This class is a simple container of data for a single logged email message. All contained data are represented with public auto-implemented properties. This class is a DTO used by HeraProxiedEmailManager and IProxiedEmailPersistence.

EmailUserParams

This class is a simple container of data for a single email user. It can be used in three different situations: 1) when it holds some identifier of a user only - in this case only one of UserId, RealEmail and ProxyEmail properties are expected to be set; values of each of these properties are unique among all users; 2) when it holds parameters of the updated email user data - in this case null property means that this property is not required to be updated; 3) when all properties are not null - in this case it holds complete data for a single email user. This class is a DTO used by HeraProxiedEmailManager and IProxiedEmailPersistence.

HashBasedProxyEmailGenerator

This class is an implementation of IProxyEmailGenerator that generates proxy email address by concatenating user ID, hash of the real email address and domain name.

HeraProxiedEmailManager

This is the main class of this component. It represents a manager for proxied email data in persistence. It uses pluggable IProxyEmailGenerator instance to generate proxy email addresses and IProxiedEmailPersistence instance to access data in persistence. This class can be configured using Configuration API and File Based Configuration components. This class can perform logging of errors and debug information using Logging Wrapper component.

IProxiedEmailPersistence [interface]

This is an interface that defines methods for accessing proxy email mapping and email user data in the persistence. It is expected that implementations of this interface will be configured with use of Configuration API by calling Configure() method right after construction.

IProxyEmailGenerator [interface]

This interface represents a proxy email generator used by HeraProxiedEmailManager for generating proxy email addresses by user ID, real email address and domain name. It is expected that implementations of this interface will be configured with use of Configuration API by calling Configure() method right after construction.

ProxiedEmailPersistence

This class is an implementation of IProxiedEmailPersistence that uses Data Access Interface component and stored procedures to access data in the database. It must be configured by calling the Configure() method.

1.5 Component Exception Definitions

EmailUserNotFoundException

This exception is thrown by HeraProxiedEmailManager and implementations of IProxiedEmailPersistence when user with the specified parameters cannot be found in persistence.

HeraProxiedEmailManagementConfigurationException
This exception is thrown by HeraProxiedEmailManager and implementations of IProxyEmailGenerator and IProxiedEmailPersistence interfaces when some error occurs while reading the configuration (e.g. when some required property is missing or has invalid format) or initializing the class using this configuration.

HeraProxiedEmailManagementException
This exception is a base class for all other custom checked exceptions defined in this component. It is never thrown directly, subclasses are used instead.

InvalidSearchParameterFormatException
This exception is thrown by HeraProxiedEmailManager and implementations of IProxiedEmailPersistence when value of some parameter has invalid format.

ProxiedEmailPersistenceException
This exception is thrown by HeraProxiedEmailManager and implementations of IProxiedEmailPersistence when some other error occurs while accessing the persistence. Also this exception is used as a base class for other specific custom exceptions.

ProxyEmailGenerationException
This exception is thrown by HeraProxiedEmailManager and implementations of IProxyEmailGenerator when some error occurs while generating a proxy email address.

UnknownSearchParameterException
This exception is thrown by HeraProxiedEmailManager and implementations of IProxiedEmailPersistence when some parameter name is unknown.

1.6 Thread Safety

This component is thread safe.

HeraProxiedEmailManager is immutable and thread safe. It uses IProxyEmailGenerator and IProxiedEmailPersistence instances in thread safe manner.

Implementations of IProxiedEmailPersistence and IProxyEmailGenerator must be thread safe when Configure() method is called once right after construction.

ProxiedEmailPersistence is mutable, but thread safe when Configure() method is called once right after construction. In this case mutable attributes never change after initialization. ProxiedEmailPersistence uses thread safe IDataAccess instance.

HashBasedProxyEmailGenerator is immutable and thread safe.

EmailLog and EmailUserParams are mutable and not thread safe DTO entities.

Transactions are used in SPs that modify data in the database to preserve data integrity.

2. Environment Requirements

2.1 Environment

Development Language: C# 3.5
Compile Target: C# 3.5
QA Environment: Windows Server 2003 or 2008, SQL Server 2005

2.2 TopCoder Software Components

Configuration API 1.0 – is used for configuring this component.

File Based Configuration 2.0 – is used for reading configuration of HeraProxiedEmailManager.

Logging Wrapper 3.0.1 – is used for logging errors and debug information.

Exception Manager 2.0 – is used by custom exceptions defined in this component.

Data Access Interface 2.1 – is used for accessing the database.

Object Factory 1.2.1 – is used for creating pluggable object instances.

Object Factory Configuration API Plugin 1.1.1 – is used for creating predefined objects defined with use of Configuration API component.

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

None

3. Installation and Configuration

3.1 Package Name

Hera.ProxiedEmail
Hera.ProxiedEmail.Generators
Hera.ProxiedEmail.Persistence

3.2 Configuration Parameters

3.2.1 Configuration of HeraProxiedEmailManager

The following table describes the structure of IConfiguration used in the constructor of HeraProxiedEmailManager class (angle brackets are used for identifying child configuration objects).

Parameter	Description	Values
<loggerConfig>	The configuration of Logging Wrapper logger to be used by this class. When not specified, logging is not performed.	IConfiguration. Optional.
<objectFactoryConfig>	This section contains configuration of Object Factory used by this class for creating IProxiedEmailPersistence and IProxyEmailGenerator instances.	IConfiguration. Required.
persistenceKey	The Object Factory key that is used for creating an instance of IProxiedEmailPersistence to be used by this class.	String. Not empty. Required.
<persistenceConfig>	The configuration passed to Configure() method of the created IProxiedEmailPersistence instance. See section 3.2.2 for details.	IConfiguration. Required.
proxyEmailGeneratorKey	The Object Factory key that is used for creating an instance of IProxyEmailGenerator to be used by this class. When not specified, an instance of HashBasedProxyEmailGenerator is used.	String. Not empty. Optional.
<proxyEmailGeneratorConfig>	The configuration passed to Configure() method of the created IProxyEmailGenerator instance.	IConfiguration. Required.
domainName	The domain name to be used when generating proxy email addresses.	String. Not empty. Required.

3.2.2 *Configuration of ProxiedEmailPersistence*

The following table describes the structure of IConfiguration used in the Configure() methods of ProxiedEmailPersistence class (angle brackets are used for identifying child configuration objects).

Parameter	Description	Values
<objectFactoryConfig>	This section contains configuration of Object Factory used by this class for creating IDataAccess instance.	IConfiguration. Required.
dataAccessKey	The Object Factory key that is used for creating an instance of IDataAccess to be used by this class.	String. Not empty. Required.
connectionName	The predefined connection name passed to IDataAccess instance when establishing a connection to DB. When not specified, the default connection is used.	String. Not empty. Optional.

3.3 Dependencies Configuration

Please see docs of Object Factory, Data Access Interface, Logging Wrapper and File Based Configuration to configure them properly.

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 see the demo.

4.3 Demo

4.3.1 *API usage*

```
// Create HeraProxiedEmailManager using the default configuration name
HeraProxiedEmailManager heraProxiedEmailManager = new HeraProxiedEmailManager();

// Create HeraProxiedEmailManager using a custom configuration name
heraProxiedEmailManager = new HeraProxiedEmailManager("my_config");

// Create HeraProxiedEmailManager using IConfiguration instance
IConfiguration config = ...
heraProxiedEmailManager = new HeraProxiedEmailManager(config);

// Create and persist proxy email address for "user1@mail.com"
string proxy1 = heraProxiedEmailManager.CreateProxyEmailAddress(1, "user1@mail.com", true);
// proxy1 must be "1_348213940@test.com"

// Check if "1_348213940@test.com" proxy email exists
bool exists = heraProxiedEmailManager.IsProxyEmailExist("1_348213940@test.com");
// exists must be true

// Create and persist proxy email address for "user2@mail.com"
string proxy2 = heraProxiedEmailManager.CreateProxyEmailAddress(2, "user2@mail.com", true);
// proxy1 must be "2_348221025@test.com"

// Check if "user2@mail.com" real email exists
exists = heraProxiedEmailManager.IsRealEmailExist("user2@mail.com");
// exists must be true

// Retrieve the proxy address by real address
```

```
string proxyAddress = heraProxiedEmailManager.GetProxyAddressByRealAddress("user1@mail.com");
// proxyAddress must be "1_348213940@test.com"

// Retrieve the real address by proxy address
string realAddress =
    heraProxiedEmailManager.GetRealAddressByProxyAddress("2_348221025@test.com");
// realAddress must be "user2@mail.com"

// Retrieve the proxy address by user ID
proxyAddress = heraProxiedEmailManager.GetProxyAddressByUserId(1);
// proxyAddress must be "1_348213940@test.com"

// Retrieve the real address by user ID
realAddress = heraProxiedEmailManager.GetRealAddressByUserId(2);
// realAddress must be "user2@mail.com"

// Update proxy address for user
heraProxiedEmailManager.UpdateProxyAddressForUser(1, "first_user@test.com");

// Update real address for user
heraProxiedEmailManager.UpdateRealAddressForUser(2, "user2@gmail.com");

// Block user 2 by user 1 using real addresses
heraProxiedEmailManager.BlockUserByRealAddress("user2@gmail.com", "user1@mail.com");

// Check if user 2 is blocked by user 1 or globally
bool blocked = heraProxiedEmailManager.IsBlocked("2_348221025@test.com",
                   "first_user@test.com");
// blocked must be true

// Unblock user 2 by user 1 using proxy address
heraProxiedEmailManager.UnblockUserByProxyAddress("2_348221025@test.com",
    "first_user@test.com");

// Block user 1 by user 2 using proxy address
heraProxiedEmailManager.BlockUserByProxyAddress("first_user@test.com",
    "2_348221025@test.com");

// Unblock user 1 by user 2 using real address
heraProxiedEmailManager.UnblockUserByRealAddress("user1@mail.com", "user2@gmail.com");

// Block user 1 by real address
heraProxiedEmailManager.BlockByRealAddress("user1@mail.com");

// Unblock user 1 by real address
heraProxiedEmailManager.UnblockByRealAddress("user1@mail.com");

// Block user 2 by proxy address
heraProxiedEmailManager.BlockByProxyAddress("2_348221025@test.com");

// Unblock user 2 by proxy address
heraProxiedEmailManager.UnblockByProxyAddress("2_348221025@test.com");

// Log email message
string fromAddress = "first_user@test.com";
IList<string> toAddresses = new List<string> { "2_348221025@test.com" };
IList<string> bccAddresses = new List<string>();
string subject = "Test message";
byte[] body = Encoding.ASCII.GetBytes("Message body");
heraProxiedEmailManager.LogEmail(fromAddress, toAddresses, bccAddresses, subject, body);

// Search email messages from "first_user@test.com"
IDictionary<string, string> parameters =
    new Dictionary<string, string> { {"FromUser", "first_user@test.com"} };
IList<EmailLog> emailLogs = heraProxiedEmailManager.SearchEmailLogs(parameters);
// emailLogs.Count must be 1
// emailLogs[0].From must be "first_user@test.com"

// Disable logging history for user 1
heraProxiedEmailManager.ChangeLoggingHistoryEnabledForUser(false, 1);
```

```
// Check if logging history enabled for user 1
bool enabled = heraProxiedEmailManager.IsLoggingHistoryEnabledForUser("first_user@test.com");
// enabled must be false

// Check if logging history enabled for user 2
enabled = heraProxiedEmailManager.IsLoggingHistoryEnabledForUser("2_348221025@test.com");
// enabled must be true

// Enable block of user 1 by user 2 using IDs
heraProxiedEmailManager.SetUsersBlock(new EmailUserParams {UserId = 1},
    new EmailUserParams {UserId = 2}, true);

// Check if user 1 is blocked by user 2 or globally
blocked = heraProxiedEmailManager.IsUserBlocked(new EmailUserParams {UserId = 1},
    new EmailUserParams {UserId = 2});
// blocked must be true

// Update proxy address and blocked flag of the user 2
heraProxiedEmailManager.UpdateEmailUser(new EmailUserParams {UserId = 2},
    new EmailUserParams {ProxyEmail = "second_user@test.com", Blocked = true});

// Enable user history for user 1 by proxy address
heraProxiedEmailManager.ChangeLoggingHistoryEnabledForUser("first_user@test.com", true);

// Check if logging history is enabled for user 1 real address
enabled = heraProxiedEmailManager.IsLoggingHistoryEnabledForUser("user1@mail.com");
// enabled must be true

// Retrieve the email user with ID=1
EmailUserParams emailUserParams =
    heraProxiedEmailManager.GetEmailUser(new EmailUserParams {UserId = 1});
// emailUserParams.UserId must be 1
// emailUserParams.RealEmail must be "user1@mail.com"
// emailUserParams.ProxyEmail must be "first_user@test.com"
// emailUserParams.Blocked must be false

// Delete email user with ID=2
heraProxiedEmailManager.DeleteEmailUser(new EmailUserParams {UserId = 2});

// Delete address mappings for user 2 by proxy address
heraProxiedEmailManager.DeleteAddressMappingsForUser("second_user@test.com");
```

4.3.2 *Sample HeraProxiedEmailManager configuration file*

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <Hera.ProxiedEmail.HeraProxiedEmailManager
    persistenceKey="ProxiedEmailPersistence"
    proxyEmailGeneratorKey="HashBasedProxyEmailGenerator"
    domainName="test.com">
    <loggerConfig>
      <!-- Put logger config here -->
    </loggerConfig>
    <objectFactoryConfig>
      <!-- Put OF config here -->
    </objectFactoryConfig>
    <persistenceConfig dataAccessKey="SqlServerDataAccess"
                       connectionName="myConnection">
      <objectFactoryConfig>
        <!-- Put OF config here -->
      </objectFactoryConfig>
    </persistenceConfig>
    <proxyEmailGeneratorConfig />
  </Hera.ProxiedEmail.HeraProxiedEmailManager>
</configuration>
```

5. Future Enhancements

None