Idempotency Manager 1.0 Component Specification

1. Design

The Idempotency Manager is responsible for managing server responses to possibly redundant client requests. The component keeps track of requests as they are received and remembers the response generated by the server. If a duplicate request is received, the Idempotency Manager simply provides the previously stored response.

All the functionality of the component is provided with the IdempotencyManager. There are typically four types operations:

- **Claiming a request has arrived.** The client will be notified of the elapsed time from the last request activity with the same identifier, and whether there is a response available in the persistence so that the client could retrieve it without ever processing the request.
- **Retrieve a response associated with the request.** There are a number of ways to do this. Even if the response is not available at that point, the client could still choose to wait for it synchronously or asynchronously.
- **Store a response in association with a request.** When the client has processed a request and would wish to store it for future reference or reference from the peers, they could store the response to persistence.
- **Various cache management functionalities.** These functionalities includes, drop entry, clear the whole cache, shrink cache size to entry count or last activity date, purge persistence to synchronize with the cache, save cache image to persistence or load cache image from persistence.

The cache is supported with Distributed Simple Cache, with request identifiers mapping to cache entries that include information about last activity time and whether the response exists in persistence.

The component defines a simple request and response objects. These are light-weight representation of the real request and response at client domain.

In order to maximize flexibility most of the functionality on IdempotencyManager is refactored out to allow pluggable implementation. These interfaces include:

- **IResponsePersitence** that handles the persistence of the response objects.
- **ICachePersistence** that handles the persistence of the cache image.
- **ICacheManagement** that handles some high-level management of the cache.
- **ICacheStrategy** that handles the strategy to decide whether a request will be processed.
- **IActivityLogger** that handles the logging of IdempotencyManager activities.

With these responsibilities factored out, the component is quite clear in structure and claims maximum flexibility.

1.1 Design Patterns

- **Singleton**:

  IdempotencyManager implements singleton pattern to provide a global entry point. There is no need to run two instances of the manager.

- **Façade**:

  IdempotencyManager implements the façade pattern to centralize various features. Client will only need to use this class to access all the functionalities.

- **Strategy:**

IdempotencyManager allows custom implementation of persistence, logging, cache management and caching decision to be pluggable.

- **Decorator**:
  CachedResponsePersistence implements the decorator pattern to facilitate re-usable caching with any IResponsePersistence implementation.

- **Composite**
  ActivityBroadcaster implements the composite pattern to allow activity notification to be sent to multiple loggers.

1.2 Industry Standards

None

1.3 Required Algorithms

Most of the methods in the component is straight-forward in nature. Those with some level of complexity have implementation notes in Poseidon documentation.

1.3.1 DatabasePersistence Table Schemas

This Table Schema works for SQL Server.

```
DROP TABLE response;
DROP TABLE cache_image;

-- the response table has entries for the responses in association with requests

CREATE TABLE response (
        identifier              VARCHAR(255) NOT NULL,
                        -- the request identifier
        security_key            VARCHAR(255) NOT NULL,
                        -- the request security key
        response_object         IMAGE NOT NULL,
                        -- the response object in binary form
        time_stamp              DATETIME NOT NULL,
                        -- the time at which the response is created
        life_time               BIGINT NOT NULL
                        -- the intended life time of the response
);

-- the cache_image table has entries for each cache entry

CREATE TABLE cache_image (
        identifier              VARCHAR(255) NOT NULL,
                         -- the request identifier
        last_activity           DATETIME NOT NULL,
                        -- the last activity of any request with the identifier
        in_storage              BIT NOT NULL
                        -- whether at least one response exists for the request
);
```

1.3.2 Transaction Requirements

It is required that all database operations that is not atomic be wrapped into a transaction. Specifically, the following methods will be transactioned:

- **DatabasePersistence.Store(request, response)**
- **DatabasePersistence.SaveCache(cache)**

1.3.3 Locking Schemas

This component is implemented with concurrency handling and a number of entities require explicit locking. It is a bit hard to sum up in the Poseidon documentation so I am reiterating here.

- **Request**: all operations accessing the attribute set will lock on attributes.
- **ActivityStatistics**: all operations performed on a certain collection will lock on that collection (receiveCounts, storeCounts, retrieveCounts).
- **CachedResponsePersistence**: all operations lock on instance.
- **DatabasePersistence**: all operations lock on connection. A few operations need to implement transaction.
- **FileSystemCachePersistence**: no locking but should open the file with appropriate share mode.
- **IdempotencyManager**: all operations on the cache should lock the cache. Operations on the callbackList should lock the list. Operations accessing either one of the instance variables should lock the instance.

1.3.4 Callback Polling

Due to the limitation of Distributed Simple Cache, we are not notified if a cache update occurs remotely (and writes into the local cache). The workaround is explicit polling. There are two places in the component that uses this polling. One is in the RetrieveResponse() with a timeout value. It will poll the cache with an interval of 100ms until it timeouts. The second is the manager level polling on the callbackList to handle all the registered callbacks. This polling interval is configurable. CallbackEntry forms up the callbackList. This class is marked package. All the polling could be replaced when the DSC notification version is in, without affecting the public API.

1.4 Component Class Overview

Only overview is provided here. Refer to Poseidon for detailed documentation.

1.4.1 TopCoder.Util.Idempotency

- ***IdempotencyManager:***

This is the main class of the component, which centralizes all the functionality the client will need to access. It's a singleton. There are a number of interfaces defined where custom logic could be plugged into.

- ***Request:***

This is a light-weight request representation that the component works with. It defines request identifier, security key, request type and an attribute set.

- ***Response:***

This is the response the component stores in association with request. The response contains a serializable response object, a creation timestamp and an expected life. However the IdempotencyManager itself does not interpret these metrics.

- ***ResponseAvailableCallback:***

This is the delegate used for clients that wish to be notified when a certain response is available. Client registers callback with a request.

- ***CacheEntry:***

This simple data structure is used in the cache to help identify whether a request has arrived before and whether response is available in the persistence. This entry is serializable in order to distribute around the idempotency peers.

- ***CallbackEntry:***

This is used to keep all the callbacks registered with the IdempotencyManager.

- ***ICacheManagement:***

This interface defines some cache management functionality and allows for custom implementation. GenericCacheManagement is its pre-packaged implementation.

- ***GenericCacheManagement:***

This implementation of ICacheManagement provides functionality to shrink a cache according to entry number of activity date, and to purge the persistence to synchronize with the cache content.

1.4.2 TopCoder.Util.Idempotency.Persistence

- ***IResponsePersistence:***

This interface defines the persistence contract for response instances. CachedResponsePersistence and DatabasePersistence are its pre-packaged implementations.

- ***ICachePersistence:***

This interface defines the persistence contract for cache image. DatabasePersistence and FileSystemCachePersistence are its pre-packaged implementations.

- ***CachedResponsePersistence:***

This implementation of IResponsePersistence uses a Distributed Simple Cache to cache the responses among the idempotency group. It can serve as standalone in-memory persistence or a caching persistence for another persistence implementation.

- ***DatabasePersistence:***

This class implements both IResponsePersistence and ICachePersistence with database as backup.

- ***FileSystemCachePersistence:***

This implementation of ICacheResponse uses file system to save cache image to or load cache image from.

1.4.3 TopCoder.Util.Idempotency.Strategy

- ***ICacheStrategy:***

ICacheStrategy interface defines the strategy for the IdempotencyManager to decide whether to work with given request. TypeMappingCacheStrategy is its pre-packaged implementation.

- ***TypeMappingCacheStrategy:***

This is an ICacheStrategy that maps request types to CacheDecision's. If the request type is not specified, a default decision will be adopted.

- ***CacheDecision:***

This is an enumeration used by TypeMappingCacheStrategy and RuleListCacheStrategy. The CacheDecision has three values – Accept, Deny, and Default.

1.4.4 TopCoder.Util.Idempotency.Loggr

- ***IActivityLogger:***

This interface defines a callback that will be notified for various activities of IdempotencyManager. ActivityLogging and ActivityStatistics are the pre-packaged implementations.

- ***ActivityLogging:***

This IActivityLogger implementation uses Logging Wrapper to log the activities.

- ***ActivityStatistics:***

This IActivityLogger implementation uses in-memory tables to bring up statistics about the requests.

- **ActivityBroadcaster**

This IActivityLogger implementation allows a number of loggers aggregated into one and receive notifications from one manager source.

1.5 Component Exception Definitions

1.5.1 Custom Exceptions:

- ***PersistenceException:***

This exception is generally used for the ICachePersistence and IResponsePeresistence interface if the underlying media fails for any reason. Logic errors will not lead to this exception. For all the method that uses the persistence implementation directly or indirectly, this exception is usually propagated.

- ***ResponseRetrievalException:***

This exception is used when there is no response available in the persistence that matches both the request identifier and the security key. Usually the client will make sure if a response exists with HasResponse() on the IdempotencyManager, so ResponseRetrievalException could be interpreted as a kind of unauthorized access.

1.5.2 Foreign Component Exceptions:

- ***InvalidConfigFileException:***

This exception is used for all the constructors that loads configuration from Configuration Manager. If some required property does not exist, or invalid property value is specified, the exception will be thrown since there is no way to properly instantiate the entity in question.

1.5.3 System Exceptions:

- ***ArgumentNullException:***

This exception is thrown in all the places that do not allow for null arguments.

- ***ArgumentException:***

This exception is thrown in all places where invalid argument is received, including empty string, negative count or time interval, etc.

1.6 Thread Safety

Thread safety is important for this component. The component will be used to reduce duplicate process for services and will be mostly typically used in multi-threaded environment.

All the data structures are either immutable by nature or will be manipulated thread-safely with proper locking. The ICachePersistence, IResponsePersistence, IActivityLogger implementations are all implemented with thread-safety in mind. The ICacheManagement and ICacheStrategy implementations are stateless and support concurrency in nature. The IdempotencyManager itself uses locking schemas to ensure thread-safety. Refer to the algorithm section for more details.

The dependency components Distributed Simple Cache, Configuration Manager, Connection Factory and Logging Wrapper are all thread-safe. Linked List is not thread-safe and will be used with explicit locking.

It is required that any pluggable implementations of the various interfaces to be thread-safe to work with IdempotencyManager.

2. Environment Requirements

2.1 Environment

.NET Framework 1.0

2.2 TopCoder Software Components

- *Distributed Simple Cache 1.1*

  Distributed Simple Cache supports a group of caches to synchronize over networking. The component defines an IDistributedSimpleCache interface that provides the functionality of a common cache. And client could use this cache without ever notice any underlying traffic.

- *Configuration Manager 1.0*

  Configuration Manager provides a centralized way for the component to manage its configuration properties. Configuration is used for seven entities in the component and in order for the IdempotencyManager to start up on itself, configuration should be preloaded. The component will generally use ConfigManager.GetValue()/GetValues() to load properties as strings.

- *Connection Factory 1.0*

  Connection Factory takes the actual connection creation logic out of the component. With this the connection is fully configurable outside the component. The DatabasePersistence uses ConnectionManager.CreatePredefinedDbConnection() to create the connection to the database.

- *Logging Wrapper 1.0*

  Logging Wrapper provides pluggable log support for the component. It is used in the ActivityLogging implementation of IActivityLogger. The Logging Wrapper component should be initialized prior to use. The actual logging will be performed with LogManager.Log().

- *Linked List 1.0*

  Linked List provides custom implementation of the linked list collection which is not available in the .NET collection framework. The callback list in the IdempotencyManager is maintained as a linked list so that entries can be removed with constant complexity.

2.3 Third Party Components

This component does not depend on any third party components. Logging Wrapper might need log4net to run if the client chooses to use this implementation for the underlying logging.

3. Installation and Configuration

3.1 Package Name

TopCoder.Util.Idempotency

TopCoder.Util.Idempotency.Persistence

TopCoder.Util.Idempotency.Strategy

TopCoder.Util.Idempotency.Logger

3.2 Configuration Parameters

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ConfigManager SYSTEM "cm.dtd">

<ConfigManager>
  <namespace name="TopCoder.Util.Idempotency.Strategy.TypeMappingCacheStrategy">
    <!-- a list of known request types -->
    <property name="known_types">
      <value>GET</value>
      <value>POST</value>
    </property>
    <!-- the cache decision for the type GET -->
    <property name="GET_decision">
      <value>Accept</value>
    </property>
    <!-- the cache decision for the type POST -->
    <property name="POST_decision">
      <value>Deny</value>
    </property>
    <!-- the default cache decision if the request type is unknown -->
    <property name="default">
      <value>Accept</value>
    </property>
  </namespace>
  <namespace name="TopCoder.Util.Idempotency.Logger.ActivityLogging">
    <!-- the logging format for a request – identifier, security key, type and time in that order -->
    <property name="format">
      <value>req {0} with key {1} of type {2} at {3}</value>
    </property>
```

```xml
    <!-- whether to log the security key -->
    <property name="log_security_key">
        <value>true</value>
    </property>
</namespace>
<namespace name="TopCoder.Util.Idempotency.Logger.ActivityBroadcaster">
    <!-- a list of activity loggers to broadcast to -->
    <property name="activity_loggers">
        <value>TopCoder.Util.Idempotency.Logger.ActivityLogging</value>
        <value>TopCoder.Util.Idempotency.Logger.ActivityStatistics</value>
    </property>
</namespace>
<namespace name="TopCoder.Util.Idempotency.Persistence.CachedResponsePersistence">
    <!-- the cache url for the distributed simple cache -->
    <property name="cache_url">
        <value>http://localhost:14701/cachedpersistence</value>
    </property>
    <!-- the start url for the distributed simple cache -->
    <property name="start_url">
        <value>http://192.168.1.156:14701/cachedpersistence</value>
    </property>
    <!-- the inner persistence for caching -->
    <property name="response_persistence">
        <value>TopCoder.Util.Idempotency.Persistence.DatabasePersistence</value>
    </property>
</namespace>
<namespace name="TopCoder.Util.Idempotency.Persistence.DatabasePersistence">
    <!-- the predefined connection defined in connection factory -->
    <property name="predefined_connection">
        <value>idempotency</value>
    </property>
</namespace>
<namespace name="TopCoder.Util.Idempotency.Persistence.FileSystemCachePersistence">
    <!-- the filename for persistence the cache image -->
    <property name="filename">
        <value>/etc/idempotency/cache.image</value>
    </property>
    </namespace>
    <namespace name="TopCoder.Util.Idempotency.IdempotencyManager">
        <!-- the cache url for the distributed simple cache -->
        <property name="cache_url">
            <value>http://localhost:14700/idempotency</value>
        </property>
        <!-- the start url for the distributed simple cache -->
        <property name="start_url">
            <value>http://192.168.1.156:14700/idempotency</value>
        </property>
        <!-- the cache management implementation to use -->
        <property name="cache_management">
            <value>TopCoder.Util.Idempotency.GenericCacheManagement</value>
        </property>
        <!-- the cache strategy implementation to use -->
        <property name="cache_strategy">
            <value>TopCoder.Util.Idempotency.Strategy.TypeMappingCacheStrategy</value>
        </property>
        <!-- the cache persistence implementation to use -->
        <property name="cache_persistence">
            <value>TopCoder.Util.Idempotency.Persistence.DatabasePersistence</value>
        </property>
        <!-- the response persistence implementation to use -->
        <property name="response_persistence">
            <value>TopCoder.Util.Idempotency.Persistence.CachedResponsePersistence</value>
        </property>
        <!-- the activity logger implementation to use -->
        <property name="activity_logger">
            <value> TopCoder.Util.Idempotency.Logger.ActivityBroadcaster</value>
        </property>
        <!-- the polling interval for the background thread handling callback list -->
        <property name="polling_interval">
            <value>5000</value>
        </property>
    </namespace>
</ConfigManager>
```

3.3 Dependencies Configuration

Dependency configuration is required for Configuration Manager (which requires a preload list), Connection Factory and Distributed Simple Cache. Logging Wrapper potentially needs configuration for log4net. Please refer to the respective documentation for sample configuration.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration.
- Execute ‘nant test’ within the directory that the distribution was extracted to.

4.2 Required steps to use the component

None

4.3 Demo

```
// Get Idempotency Manager
IdempotencyManager manager = IdempotencyManager.Instance;

// New request arrives
Request request = new Request("/importal/readme.txt", "", "GET");
DateTime elapsed = manager.RequestArrival(request);
bool exists = manager.HasResponse(request);

// Retrieve response
Response response = manager.RetrieveResponse(request, TimeSpan.FromSeconds(5));
manager.RetrieveResponse(request,
                new ResponseAvailableCallback.ResponseAvailable(ResponseHandler),
                TimeSpan.FromMinutes(10));

// Store response
response = new Response("hello world", DateTime.Now, TimeSpan.FromDays(1));
manager.StoreResponse(request, response);

// Manage pluggable sources
DatabasePersistence persistence = new DatabasePersistence(new SqlConnection(@"..."));
manager.ResponsePersistence = persistence;
manager.CachePersistence = persistence;
manager.CacheStrategy = new TypeMappingStrategy();
manager.CacheManagement = new GenericCacheManagement();
manager.ActivityLogger = new ActivityStatistics();

// Manage cache
manager.DropEntry("/importal/readme.txt");
manager.ClearCache();
manager.ShrinkCacheToSize(50);
manager.ShrinkCacheToDate(DateTime.Now - TimeSpan.FromDays(5));
manager.PurgeResponsePersistence();
manager.SaveCache();
manager.LoadCache();
```

5. Future Enhancements

The most pressing issue naturally to replace the polling mechanism for notifying mechanism. This will only be feasible once the Distributed Simple Cache is updated. Another obvious direction is to provide more custom implementations for various interfaces. Utilities could be developed to normalize real request into light-weight request and magnify light-weight response back into real response.