Idempotency Manager Requirements Specification

1. Scope

1.1 Overview

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.

1.2 Logic Requirements

1.2.1 Redundant Request Detection

When a request is received by an application, it will first pass a lightweight representation of the request to the Idempotency Manager to determine if the request has already been received. If the request has previously been received, the Idempotency Manager will return two pieces of information: 1) Time elapsed since the request was last received (before now) and 2) Whether or not there is a corresponding response stored for the request. It is possible that there may not be a corresponding response stored for the request even if the component determines the request was previously received. This is because the application may still be processing the original request.

When the application performs this request to the component, the component recognizes that an instance of the specified request has been received; it caches the lightweight representation of the request if it does not already exist and updates related information (last time request was received is set to now).

1.2.2 Response Storage

The component provides a means to persist responses to permanent or temporary storage. The actual type of storage used is configurable. The Idempotency Manager comes prepackaged with an in-memory storage implementation which uses the Simple Cache component.

When an application wishes to store a response, it will provide the component with two things: 1) A lightweight representation of the request and 2) The actual response. The Idempotency Manager will store the response and will cache the lightweight representation of the request using the Distributed Simple Cache component. Caching the lightweight representation of the request guarantees speedy redundant request detection, because it removes the need to go to the potentially slower storage implementation.

1.2.3 Response Retrieval

The component provides a means to retrieve responses from storage. The actual type of storage used is configurable. The Idempotency Manager comes prepackaged with an in-memory storage implementation which uses the Simple Cache component.

When an application wishes to retrieve a response, it will provide the component with the lightweight representation of the request. The Idempotency Manager will retrieve the response and return it to the application. If the request/response pair does not exist, an error will occur.

1.2.4 Wait for Response

It is possible that a redundant request could be received and that there is not yet a corresponding

response available. If this is the case, the application may choose to wait for the response. The Idempotency Manager provides a method to wait for a response to a given request.

When an application wishes to wait for a response, it will provide the component with the lightweight representation of the request. The application may optionally provide a timeout value, after which the thread should be allowed to continue without waiting further.

1.2.5 Response Notification

It is possible that a redundant request could be received and that there is not yet a corresponding response available. If this is the case, the application may choose to wait request notification when a response becomes available. The Idempotency Manager provides a method to add a listener for a particular response.

When an application wishes to receive notification of a response, it will provide the component with the lightweight representation of the request and a listener object .

1.2.6 Special Request Type Handling

The Idempotency Manager can be configured to handle a set of request types differently than other requests. This is accomplished by specifying a 'request type list' of request types. Request types appearing in the request type list may be configured to have idempotency management enabled or disabled for that particular request type. If idempotency management is disabled for the request, it will *not* notify the application of redundant requests during 1.2.1 Redundant Request Detection.

If a request is received of a type that is not in the request type list, the component will use a configurable default setting of enabled or disabled idempotency management (the component may also be configured to generate an error in this case).

1.2.7 Lightweight Request Representation

Requests represented in a lightweight fashion must have the following attributes:

- **Primary Identifier**
  Each request must have exactly one primary identifier (String). The primary identifier is used to uniquely identify a request during 1.2.1 Redundant Request Detection and 1.2.4 Wait for Response. Two requests are considered equal if they their primary identifiers are equal.
- **Secondary Identifier**
  Each request may have an optional secondary identifier (String). The secondary identifier is used in combination with the primary identifier to uniquely identify a request during 1.2.2 Request/Response Storage and 1.2.3 Request Response Retrieval. Two requests are considered equal if their primary and secondary identifiers are equal.
- **Request Type**
  Each request must have exactly one request type (String).

1.2.8 Cache Persistence

The component provides a means to persist the current cache to the file system. The application may later choose to load all elements from the persisted cache-file in to the current cache.

When the application wishes to persist the current cache to the file system, it provides the Idempotency Manager with a file location to which the cache should be written. The component will attempt to write the cache to the specified location. If the component cannot successfully

write the cache to the file, an error will occur.

When the application wishes to retrieve a cache-file from the file system, it provides the Idempotency Manager with the location of the file to be loaded. The component will attempt to read the file and load the requested data in to the cache. If the component cannot successfully read from the file or load the data in to the cache, an error will occur.

1.2.9 Cache Management

The component provides a means to manage the size of the stored cache. At any time, the application may choose to: 1) Clear the entire cache, 2) Clear all requests from the cache that are older than a specified age or 3) Clear requests from the cache (starting with the oldest requests) to limit the size of the cache to a specified size.

The component may be configured to automatically manage using a pluggable cache management implementation. The Idempotency Manager comes prepackaged with an implementation that can be configured to manage the cache by either: 1) Clearing requests from the cache that are older than a specified age or 2) Clearing request from the cache (starting with the oldest requests) to keep the size of the cache under a specified maximum size.

1.2.10 Thread-Safety and Performance

The component must be thread-safe. Performance is a primary concern with this component. The act of checking for redundant requests (1.2.1) must be optimized for speed and accuracy.

1.2.11 Distributed Component

The Idempotency Management component is a distributed component. Two instances of the component running on different servers must be able to share request and response data as if they were running on a single machine.

1.3 Required Algorithms

None.

1.4 Example of the Software Usage

An eCommerce application driven by an XML Web Service might use this component to detect and avoid processing redundant user requests, for example: clicking twice on the button that charges the user's credit card. The component might also be used to handle redundant requests that may result from network connectivity issues.

1.5 Future Component Direction

None.

2. Interface Requirements

2.1.1 Graphical User Interface Requirements

None.

2.1.2 External Interfaces

None.

2.1.3 Environment Requirements

- Development language: C#

2.1.4 Namespace

TopCoder.Util.Idempotency

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the application need to be configurable?

- Request Type List
  - Request Type – Enabled/Disabled
- Default Request Type – Enabled/Disabled
- Pluggable Storage Implementation
- Cache Management Implementation

3.2 Technical Constraints

3.2.1 Are there particular frameworks or standards that are required?

None.

3.2.2 TopCoder Software Component Dependencies:

None.
**Please review the TopCoder Software component catalog for existing components that can be used in the design.

3.2.3 Third Party Component, Library, or Product Dependencies:

None.

3.2.4 QA Environment:

- Windows 2000
- Windows Server 2003

3.3 Design Constraints

The component design and development solutions must adhere to the guidelines as outlined in the TopCoder Software Component Guidelines. Modifications to these guidelines for this component should be detailed below.

3.4 Required Documentation

3.4.1 Design Documentation

- Use-Case Diagram
- Class Diagram
- Sequence Diagram
- Component Specification

3.4.2 Help / User Documentation

XML documentation must provide sufficient information regarding component design and usage.