HeartBeat v1.0 Component Specification

1. Design

The HeartBeat component provides functionality to provide one or more "HeartBeats" to keep resources from timing out or expiring. This component separates the fundamental concepts between HeartBeat action and the triggering of that action. By providing separation between these concepts, the component becomes far more flexible by allowing the mixing and matching of actions and triggers. Heartbeat actions and the triggering of that action will most likely be very unique to an application and this component addresses that by allowing the application to implement their own actions and triggers (or mixing their own with some of the standard implementations).

The HeartBeat component provides the following:

1) A HeartBeatManager as a centralized class to manage both the heart beat actions and their triggers. This class provides the ability to add, query, retrieve and remove both triggers and actions. The class also provides a convenience method to handle the most widely used action – scheduling a heart beat action that repeats every x amount of time.
2) A HeartBeatTrigger interface the application can implement to provide their own custom triggering process (such as alert triggers, etc). This component includes two implementations of this interface:
    a) A generic manual trigger that an application can use to manually trigger any heart beat action registered to it. This is a convenience class an application can utilize to trigger actions in some customized way without having to write their own custom trigger).
    b) A timer based trigger allowing for single execution triggers, repeating triggers and time delayed triggers. This trigger uses the java.util.Timer object to manage triggering of the heart beat action and exposes some of the methods that the Timer allows.
3) A HeartBeat interface the application can implement to provide their own custom heart beat actions and return the last exception that was encountered. This component provides three implementation of this interface:
    a) A URL action that will simply read the given URL fully whenever it is triggered. This implementation is useful in keeping an http session alive or notifying any URL based resource that the application is still alive.
    b) A Socket action that will write a given message (as represented by a byte array) to a given socket.
    c) A OutputStream action that will write a given message (as represented by a byte array) to a given output stream.

1.1 Thread Safety

The manager and the provided implementations are internally thread safe. If a calling application provides custom implementation – the thread safety of the component is dependent on the thread safety of the custom implementation. Please note that removing a HeartBeat simply guarantee's the HeartBeat will no longer be **triggered** from

that point and makes no guarantee's that any executing HeartBeat will be completed prior to returning.

Example using the TimerTrigger:
OurThread = ThrA
TimerThread = ThrT

** ThrT realizes it is time to run our task.*
** ThrT calls the appropriate TimerTriggerTask.Run*
** ThrA calls remove(heartBeat) which calls cancel() of the task, and then returns from the function*
** ThrT finishes running the heart beat AFTER the function above finishes.*

1.2 Design Patterns

A Template Pattern is used to define both HeartBeat and HeartBeatTrigger implementations.

1.3 Industry Standards

None

1.4 Required Algorithms

None

1.5 Component Class Overview

HeartBeat

Interface defining a heart beat action. Implementing classes should save the last exception that was encountered during the keep alive process to be returned via the getLastException method. The last exception should be set back to null if any later keep alive process were successful.

HeartBeatManager:

Manages heart beats and triggers.

HeartBeatTrigger

Interface defining a triggering mechanism for a heart beat action. Note: an add HeartBeat is not part of the contract because the adding of an action is highly specific to how the trigger functions.

ManualTrigger

Implementation of a HeartBeatTrigger. This convenience trigger allows an application to manually trigger a HeartBeat action.

OutputStreamHeartBeat

Implementation of a HeartBeat. This action will write a specified message to an OutputStream when triggered. If any IOException occurs while writing the message, the exception should be saved and returned via the getLastException method. If the message is later sent successfully, the last exception should be set back to null.

SocketHeartBeat

Implementation of a HeartBeat. This action will write a specified message to a Socket when triggered. If any IOException occurs while writing the message, the

exception should be saved and returned via the getLastException method. If the message is later sent successfully, the last exception should be set back to null.

TimerTrigger

Implementation of a HeartBeatTrigger. This trigger exposes an API similar to the java.util.Timer to allow delayed and repeatable triggering of a HeartBeat action. As HeartBeat's are added to this trigger, they are wrapped in a TimerTriggerTask, saved to the list of timer tasks and then scheduled with the internal Timer. When a single HeartBeat is removed, the internal list of TimerTriggerTasks is search for **any** matching HeartBeat (== comparison) and the remove method of the task is executed.

TimerTriggerTask

This is a class used internally by the TimerTrigger to wrap the HeartBeat action into a timer task.

URLHeartBeat

Implementation of a HeartBeat. This action will fully read a specified URL when triggered.

1.6 Component Exception Definitions

HeartBeatManager:

A NullPointerException will be thrown by the add, remove and contains methods if any of the specified parameter(s) are null.

A IllegalArgumentException will be thrown by the add(HeartBeat, int) method if the specified period is less than or equal to zero.

ManualTrigger:

A NullPointerException will be thrown by the add, remove and contains methods if any of the specified parameter(s) are null.

OutputStreamHeartBeat:

A NullPointerException will be thrown by the constructor if any of the specified parameter(s) are null.

SocketStreamHeartBeat:

A NullPointerException will be thrown by the constructor if any of the specified parameter(s) are null.

TimerTrigger:

A NullPointerException will be thrown by the schedule, remove and contains methods if any of the specified parameter(s) are null.

A IllegalArgumentException will be thrown by the schedule methods if the specified period is less than or equal to zero or the specified delay is less than zero.

TimerTriggerTask:

A NullPointerException will be thrown by the constructor if the specified parameter is null.

URLHeartBeat:

A NullPointerException will be thrown by the constructor if the specified parameter is null.

2. Environment Requirements

2.1 TopCoder Software Components:

- None

2.2 Third Party Components:

- None

3. Installation and Configuration

3.1 Package Name

HeartBeat.jar

3.2 Configuration Parameters

None

3.3 Dependencies Configuration

None.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Modify the URLs in URLHeartBeatTest and the ports in the SocketHeartBeatTest for your local environment.
- Execute 'ant test' within the directory that the distribution was extracted to.

4.2 Demo

The following demonstrates what an application can be done. Note: the operations would like occur in different threads at different times – but you get the picture…

Performing the '90%' case – simple as pie …

```
// Execute the action every 5 minutes
new HeartBeatManager().add(new URLHeartBeat(mySession), 300000);

// Have another one running every 15 minutes
new HeartBeatManager().add(new URLHeartBeat(mySession2), 900000);
```

Stopping heart beats from executing – this stops heart beats regardless of what trigger(s) they are in.

```
// Stop all heart beats.  Note: this method returns the heart beats
// that were stopped – allowing the application to add them back
```

```
// to restart them...
heartBeatManager.removeAllHeartBeats();
// Stop a specific heart beat
if(heartBeatManager.removeHeartBeat(myHeartBeat)) {
   System.out.println("MyHeartBeat was stopped!");
}
```

Starting a heart beat in 10 minutes repeating every 5 minutes

```
// Create the heart beat
HeartBeat action = new SocketHeartBeat(mySocket, "myMessage".getBytes());

// Create a timer trigger
TimerTrigger timerTrigger = new TimerTrigger();

// Add the heart beat to the trigger starting in 10 minutes
// repeating every 5 minutes
timerTrigger.schedule(action, 10*60*1000, 5*60*1000);

// Add the trigger to the manager to be later managed
new HeartBeatManager().add(timerTrigger);
```

More complex case - allowing a HeartBeat to be triggered both manually and automatically.

```
// Create the heartbeat manager
HeartBeatManager manager = new HeartBeatManager();

// Create a heart beat action
HeartBeat action = new URLHeartBeat(myURL);

// Have the Heartbeat occur every 5 minutes
manager.add(action, 5*60*1000);

// Create a Manual trigger for the action
ManualTrigger manualTrigger = new ManualTrigger();
manualTrigger.add(action);

// Add it back to the chat Server so others can access it...
manager.add(manualTrigger);

// Manually trigger the heart beat (the action
// will still happen 5 minutes from the above)
// This fires IN ADDITION to the above...
manualTrigger.fireKeepAlive();
```