ASP.NET Comet Handler 1.0 Requirements Specification

1. Scope

1.1 Overview

Comet technology enables server to push events/messages to clients under a synchronous request/ response protocol, such as HTTP. When a request is made to the server, the request is held until new events/messages are available, or a timeout occurs. Response will be sent to the client. The client will continue to make another Comet request to the server. This is to make sure that a connection is always available for the server to push events/messages to clients asynchronously. This process is known as long-polling, and continues on and on.

This component provides the Comet Handler to be used in the ASP.NET framework. The Comet Client will send requests to the Comet Handler. The Comet Handler will retrieve the new events/messages for the client from the Comet Processor. The Comet Processor is pluggable.

1.2 Logic Requirements

1.2.1 HTTP Async Handler

The Comet Handler will implement the System.Web.IHttpAsyncHandler interface. The idea behind this is to enable asynchronous processing of the Comet request, while the original ASP.NET thread continues to serve other requests without waiting for the processing to finish.

The processing starts by queuing the Comet Request into the Comet Thread Pool. When the processing of the request is finished, it will invoke a callback (the AsyncCallback argument of the BeginProcessRequest method) to notify the Comet Handler.

1.2.2 Comet Thread Pool

The Comet Thread Pool manages a number of Comet Threads. When a Comet Request is queued, it assigns the request to one of the Comet Threads using some sort of assignment strategy.

The component will provide two assignment strategies: "Round-robin" and "Load Balancing". Round-robin assignment is made by rotating the threads among the thread pool. Load Balancing assignment chooses the thread with the least number of Comet Requests.

1.2.3 Comet Thread

The Comet Thread will process the Comet Requests assigned to it. It keeps a list of Comet Requests internally, and processes them one by one in a forever running loop. Processing is done by retrieving the pending messages using the Comet Processor.

Refer to 2.1.2 for the Comet Request class, Comet Response class and the Comet Processor interface. More properties/methods can be added to the classes/interface.

1.2.4 Retrieve Pending Messages

Call the Process method of the Comet Processor. The last activated time of the Comet Request is initially null. When the Process method is called for the first time, the Comet Thread will set the last activated time to that returned in the Comet Response. This field won't be changed afterwards.

If pending messages are available in the Comet Response, the Comet Thread will set the last responded time of the Comet Request to the current time. Do not remove the Comet Request from the thread at this point. The messages will be written to the HTTP response. The AsyncCallback will be invoked to notify the Comet Handler, and the HTTP response is then sent to the client.

1.2.5 Check for Timeout

The Comet Thread will look for Comet Requests that haven't responded (because no pending message is available) in a timeout period. If timeout occurs, it will stop waiting for pending messages and send the current message in the Comet Response to the client. The handling for this is the same as sending pending messages (see 1.2.4 for details). It is expected that the Comet Processor will place a special timeout message in the message field of the Comet Response if pending messages are not available.

1.2.6 Check for Idleness

The Comet Thread will look for Comet Requests that have responded but not received another Comet request from the same client in an idle period. To do this, the Comet Thread will keep calling the Process method for those responded Comet Requests.

If the last activated time in the Comet Response is later than that in the Comet Request, the client must have connected to the handler. The Comet Thread can safely remove the Comet Request from the internal list. If the client is not seen within the idle period, the Comet Thread will call the KillIdleClient method of the Comet Processor. In other words, the thread disconnects the client from the processor on behalf of the client. The Comet Request will then be removed from the internal list.

1.2.7 Configuration

The component has its own configuration. Refer to 3.1.1 for configurable elements. Use the File Based Configuration component to load the configuration.

1.3 Required Algorithms

None.

1.4 Example of the Software Usage

In a web chat application, the Comet Handler will be used to handle the requests sent from the Comet Clients in asynchronous mode. The chat messages will be delivered to the Comet Clients by the Comet Handler.

1.5 Future Component Direction

Support more transport protocols, e.g. Bidirectional-streams Over Synchronous HTTP (BOSH)

2. Interface Requirements

2.1.1 Graphical User Interface Requirements

None.

2.1.2 External Interfaces

```
public class CometRequest
{
public bool Activated
{
get { return this.lastActivatedTime != null }
```

```
}

public bool Responded
{
get { return this.lastRespondedTime != null }
}

// provide getters and setters for the follwing properties
public DateTime? LastActivatedTime {}
public DateTime? LastRespondedTime {}
public HttpContext context {}
}

public class CometResponse
{
// provide getters and setters for the follwing properties
public object Message {}
public bool HasPendingMessage {}
public DateTime? LastActivatedTime {}
public string ContentType {}
}

public interface ICometProcessor
{
CometResponse Process(CometRequest request);
void KillIdleClient(CometRequest request);
}
```

2.1.3 Environment Requirements

- Development language: C#
- Compile target: .NET 2.0

2.1.4 Package Structure

TopCoder.Web.Comet.Handler

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the application need to be configurable?

Number of threads in the thread pool

Thread assignment strategy

Comet processor

Timeout period (in seconds)

Idleness period (in seconds)

3.2 Technical Constraints

3.2.1 Are there particular frameworks or standards that are required?

- HTTP/1.1
- ASP.NET

3.2.2 TopCoder Software Component Dependencies:

Configuration API

File Based Configuration **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:

Microsoft 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

- Design documents must clearly define intended component usage in the 'Documentation' tab of TCUML.