**Brief introduction of SocketPro high performance persistent message queue**

**Introduction**

Persistent message queue allows applications running on separate machines/processes to communicate in a failsafe manner. A message queue is a temporary storage location or file from which messages can be saved and read reliably, as and when conditions permit. Unlike sockets and other common channels that require direct connections always exist, persistent message queue enables communication among applications which may not always be connected. There are many persistent message queues implemented in own ways. SocketPro comes with an extremely high performance persistent message queue for you to reuse.

Both SocketPro client and server core libraries are internally implemented with persistent message queue. Its client queue is used to back up requests so that all requests can be resent to a server for processing in case the server is not accessible for whatever reasons such as server power-off, server application down, network off and so on. Essentially, client queue is used as a tool for fault auto recovery to increase application stability and reduction of development complexity. This feature is demonstrated by the following articles.

This article is focused on SocketPro server side persistent message queue. It is noted that SocketPro server side persistent message queue is totally free to you with open source codes which are extremely simple and understandable. You can also rely on the open source codes to extend them for your complex needs.

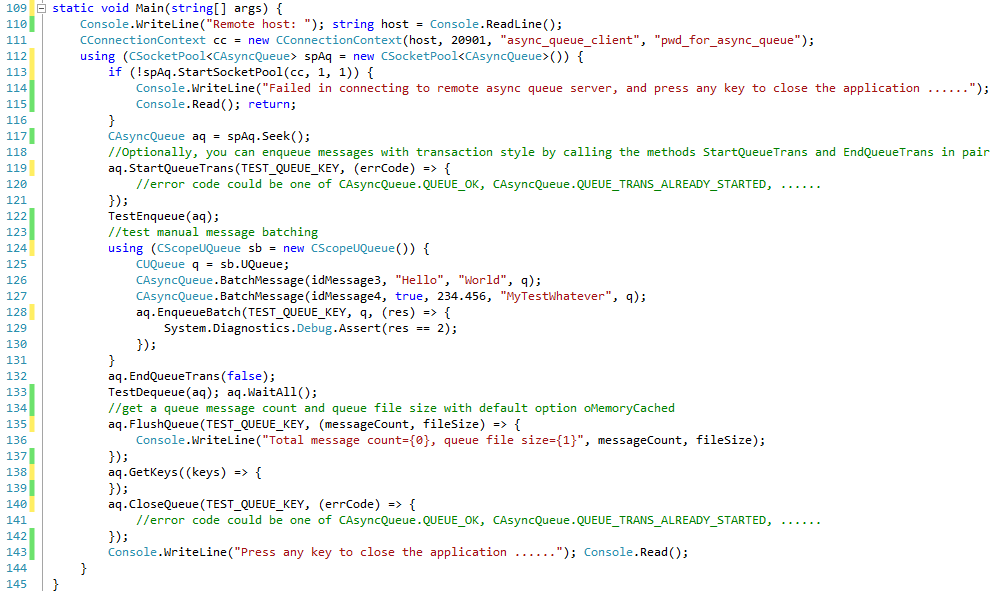
**Source codes and samples**

All related source codes and samples are located at https://github.com/udaparts/socketpro. After cloning it into your computer by GIT, pay attention to the subdirectory uasyncqueue inside the directory socketpro/samples/module\_sample. You can see these samples are created from .NET, C/C++, Java and Python development environments. They can be compiled and run on either Linux or window platforms. SocketPro comes with a pre-compiled system library uasyncqueue, which is located at directories socketpro/bin/win and socketpro/bin/linux for both windows and linux variants, respectively. In addition, you can figure out how to load the SocketPro queue service into a server application with your familiar development environment by looking at tutorial sample all\_servers at the directory socketpro/tutorials/(cplusplus|csharp|vbnet|java/src)/all\_servers. However, we only use C# client code (socketpro/samples/module\_sample/uasyncqueue /test\_csahrp) at this article for explanations.

You should distribute these system libraries inside the directory socketpro/bin into your system directory before running these sample applications. In regards to SocketPro communication framework, you may also refer to its development guide documentation at socketpro/doc/SocketPro development guide.pdf.

**Main function**

SocketPro is written from bottom to support parallel computation by use of one or more pools of non-blocking sockets. Each of pools may be made of one or more threads, and each of threads hosts one or more non-blocking sockets at client side. To increase scalability, you can create one or more pools having multiple non-block sockets that are connected to different queue servers so that you can send messages for queuing in parallel style. However, we just use one pool for demonstration clarity here. Further, the pool is only made of one thread and one socket for this sample at client side as shown in the below Figure 1.



*Figure 1: Main function for demonstration of use of SocketPro persistent message queue at client side*

**Starting one socket pool:** The above Figure 1 starts one socket pool which only has one worker thread that only hosts one non-blocking socket at line 113 for demonstration clarity by use of one instance of connection context. It is noted that you can create multiple pools within one client application if necessary. Afterwards, we get one asynchronous CAsyncQueue handler at line 117.

**Streaming message queues:** We can send individual messages onto a server for saving in stream style without batching at client side at line 122. We’ll talk by use of a new section.

**Manual message batching:** When there are many small messages to be sent for saving, these small messages will require very much CPU costs at both client and server sides because of thread synchronization, function processing, SocketPro internal inline batching as well as others. To reduce these costs, we can batch these small messages into one bigger chunk, and send them as one larger unit to server for saving. This is a way to improve message en-queue performance, but it also increases latency because it requires a time interval for collecting an enough number of small messages before manual batching. Also, it requires more codes. It is **NOT** recommended with SocketPro as long as either performance of streaming message queues meets your needs or message sizes are not very small.

**Saving message in transaction style:** SocketPro persistent message queue supports saving messages in transaction style. To use this feature, you have to call the methods *StartQueueTrans* and *EndQueueTrans* in pair as shown at lines 119 and 132. It is noted that total size of messages shouldn’t be over four gig bytes.

**Reading messages in a queue file from multiple consumers:** Certainly, you can read messages from a queue as shown at line 133. We’ll elaborate it more at the section ***TestDequeue*** in detail. It is noted that one SocketPro queue supports message writing from multiple providers and message reading from multiple consumers simultaneously at the same time. Just for your information, it is common that a queue implementation doesn’t support multiple consumers on one queue file.

**Scalability:** A client is able to create a pool that has multiple sockets connected to different server queue machines. A client is able to use the pool method *Seek* or *SeekByQueue*, and distribute messages onto different servers for saving. Don’t fool by this sample code because the demonstration is designed for clarity and beginner.

**No message loss at server side:** Message saving requires transferring messages from client or provider to a message queue server. The server and network may be down for many possible reasons as described at the beginning section ***Introduction***. Therefore, messages could be lost without your care by extra coding. You can prevent it with SocketPro easily by use of client or local message queue for backing up these messages before putting them on wire.

**Other functionalities:** SocketPro persistent message queue provides other methods to check the count of messages, the size of a queue file and keys to different message queues as well as closing a queue at lines 135, 138 and 140, respectively.

TestEnqueue