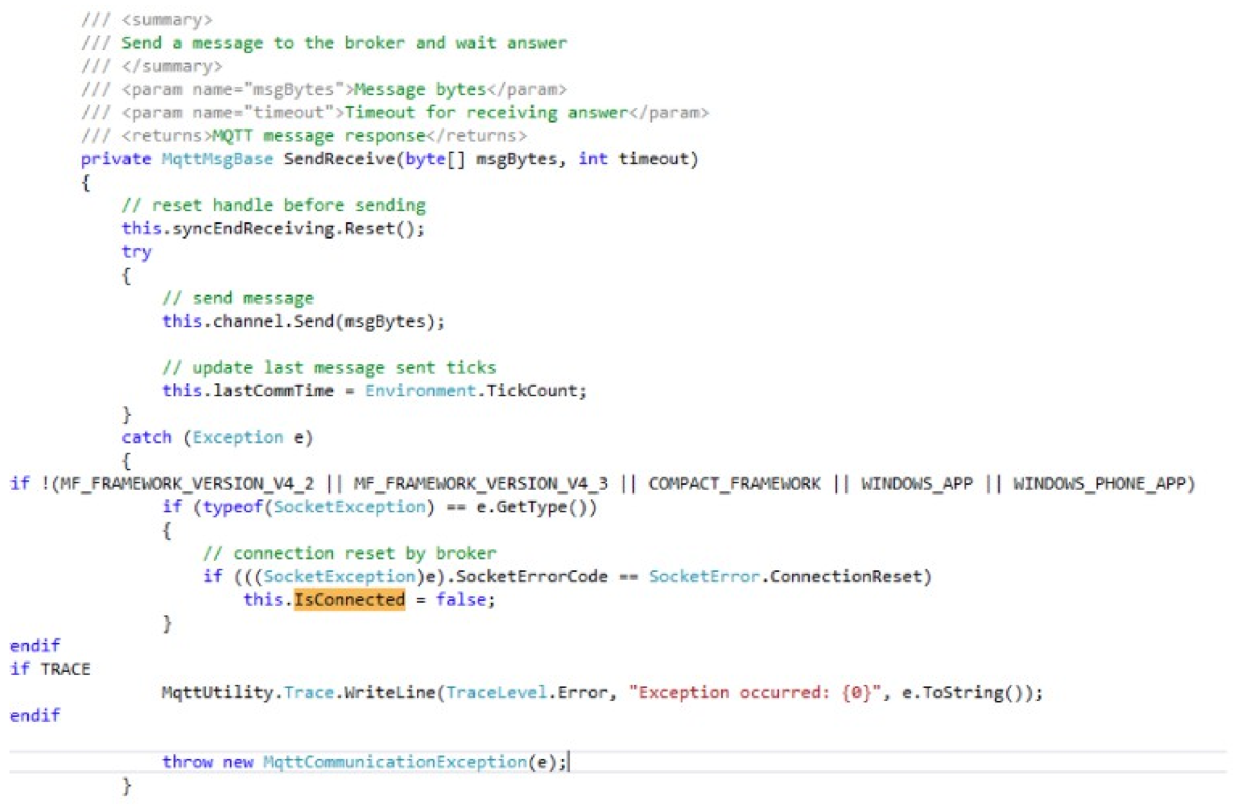
Michael code OCt2820

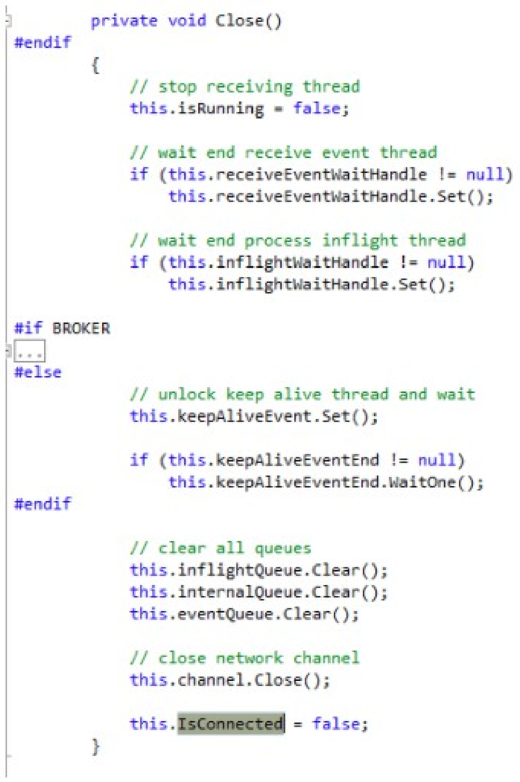


```
        /// <summary>
        /// Send a message to the broker and wait answer
        /// </summary>
        /// <param name="msgBytes">Message bytes</param>
        /// <param name="timeout">Timeout for receiving answer</param>
        /// <returns>MQTT message response</returns>
        private MqttMsgBase SendReceive(byte[] msgBytes, int timeout)
        {
            // reset handle before sending
            this.syncEndReceiving.Reset();
            try
            {
                // send message
                this.channel.Send(msgBytes);

                // update last message sent ticks
                this.lastCommTime = Environment.TickCount;
            }
            catch (Exception e)
            {
if !(MF_FRAMEWORK_VERSION_V4_2 || MF_FRAMEWORK_VERSION_V4_3 || COMPACT_FRAMEWORK || WINDOWS_APP || WINDOWS_PHONE_APP)
                if (typeof(SocketException) == e.GetType())
                {
                    // connection reset by broker
                    if (((SocketException)e).SocketErrorCode == SocketError.ConnectionReset)
                        this.IsConnected = false;
                }
endif
if TRACE
                MqttUtility.Trace.WriteLine(TraceLevel.Error, "Exception occurred: {0}", e.ToString());
endif

                throw new MqttCommunicationException(e);
            }
```

The second is on an intentional closure of the socket. This could be initiated by mcsMQTT client, but only after connection status fails previously or during a normal shutdown.



```
        private void Close()
#endif
        {
            // stop receiving thread
            this.isRunning = false;

            // wait end receive event thread
            if (this.receiveEventWaitHandle != null)
                this.receiveEventWaitHandle.Set();

            // wait end process inflight thread
            if (this.inflightWaitHandle != null)
                this.inflightWaitHandle.Set();

#if BROKER
...
#else
            // unlock keep alive thread and wait
            this.keepAliveEvent.Set();

            if (this.keepAliveEventEnd != null)
                this.keepAliveEventEnd.WaitOne();
#endif

            // clear all queues
            this.inflightQueue.Clear();
            this.internalQueue.Clear();
            this.eventQueue.Clear();

            // close network channel
            this.channel.Close();

            this.IsConnected = false;
        }
```

There is no indication in the Wireshark capture of a connection being intentionally disconnected, yet it looks like the .NET socket library has raised an exception about a port 1883 no longer being available.

There should never be a case where the same MQTT client ID has multiple connections. When this is done with Mosquitto broker then the results seen by the client are not as expected. What seems like the appropriate broker action when a new connection is requested by a client ID that already has a connection is to close the prior connection. Alternately it could reject the connection because of the duplicate client ID.