```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using Newtonsoft.Json;
using RabbitMQ.Client;
using RabbitMQ.Client.Framing;
using RabbitMQ.Client.MessagePatterns;
using RabbitRx.Advanced.Subscription;
using RabbitRx.Core.Message;
using RabbitRx.Core.Subscription;
using RabbitRx.Json.Subscription;

namespace SyntacticAnalyzer
{
    class Program
    {
        /// <summary>
        /// Connecting to a Broker
        /// </summary>
        static readonly ConnectionFactory Factory = new ConnectionFactory
          { HostName = "66.128.60.46", UserName = "dev", Password = "dev",
          VirtualHost = "/" };
        static readonly IConnection Connection = Factory.CreateConnection();

        static string exchangeName = "deviceTopic";
        static string syntacticAnalyzerQueue = "syntacticAnalyzer";
        static string contextProcessorQueue = "contextProcessor";

        static void Main(string[] args)
        {
            Start();
        }

        private static CancellationTokenSource _tokenSource;

        /// <summary>
        /// Title: RabbitRx
        /// Author: Ben Johnson
        /// Date: Jan 27, 2015
        /// Availability: https://github.com/bensmind/RabbitRx
        /// </summary>
        private static void Start()
        {
            _tokenSource = new CancellationTokenSource();

            Console.WriteLine("Syntactic Analyzer Service: Press Enter to
              Start");
            Console.ReadLine();
            Task.Run(() => ConsumeThrottle());
            Console.WriteLine("Press Any Key to Stop");
            Console.ReadLine();
            _tokenSource.Cancel();
            Start();
        }

        static void ConsumeThrottle()
        {
            var channel = Connection.CreateModel();

            channel.BasicQos(0, 50, false);
            channel.ExchangeDeclare(exchangeName, "topic");
            //Queue to send data to Context Processor
            channel.QueueDeclare(contextProcessorQueue, false, false, false, null);
            channel.QueueBind(contextProcessorQueue, exchangeName, contextProcessorQueue);

            var settings = new BasicProperties()
            {
                ContentType = "application/json",
                DeliveryMode = 1
            };

            var consumer = new JsonObservableSubscription<object>(channel, syntacticAnalyzerQueue, true);

            var throttlingConsumer = new ThrottlingConsumer<RabbitMessage<object>>(consumer, 4);

            throttlingConsumer.Subscribe(message =>
            {
                if (message != null)
                {
                    var bytes = Encoding.UTF8.GetBytes(JsonConvert.SerializeObject(message.Payload));
                    channel.BasicPublish(exchangeName, contextProcessorQueue, settings, bytes);
                    Console.WriteLine("Received (Thread {1}): {0}\n", message.Payload, Thread.CurrentThread.GetHashCode());
                }
                else
                {
                    Console.WriteLine("Received (Thread {1}): {0}\n", "INVALID JSON", Thread.CurrentThread.GetHashCode());
                }
            }, _tokenSource.Token);

            var start = throttlingConsumer.Start(_tokenSource.Token, TimeSpan.FromSeconds(1));

            start.ContinueWith(t =>
            {
                consumer.Close();
                channel.Dispose();
            });
        }
    }
}

```