```
 1  using System;
 2  using System.Collections.Generic;
 3  using System.Linq;
 4  using System.Text;
 5  using System.Threading;
 6  using System.Threading.Tasks;
 7  using Newtonsoft.Json;
 8  using RabbitMQ.Client;
 9  using RabbitMQ.Client.Framing;
10  using RabbitMQ.Client.MessagePatterns;
11  using RabbitRx.Advanced.Subscription;
12  using RabbitRx.Core.Message;
13  using RabbitRx.Core.Subscription;
14  using RabbitRx.Json.Subscription;
15
16  namespace ContextProcessor
17  {
18      class Program
19      {
20          /// <summary>
21          /// Connecting to a Broker
22          /// </summary>
23          static readonly ConnectionFactory Factory = new ConnectionFactory
              { HostName = "66.128.60.46", UserName = "dev", Password = "dev",
              VirtualHost = "/" };
24          static readonly IConnection Connection = Factory.CreateConnection();
25
26          static string exchangeName = "deviceTopic";
27          static string contextProcessorQueue = "contextProcessor";
28          static string discoveryQueue = "discovery";
29          static string authenticationQueue = "authentication";
30          static string dataManagerQueue = "dataManager";
31
32          static List<Model.Device> deviceList = new List<Model.Device>();
33
34          static void Main(string[] args)
35          {
36              Start();
37          }
38
39          private static CancellationTokenSource _tokenSource;
40
41          /// <summary>
42          /// Title: RabbitRx
43          /// Author: Ben Johnson
44          /// Date: Jan 27, 2015
45          /// Availability: https://github.com/bensmind/RabbitRx
46          /// </summary>
47          private static void Start()
48          {
49              _tokenSource = new CancellationTokenSource();
50
```

```
51              Console.WriteLine("Context Processor Service: Press Enter to Start");
52              Console.ReadLine();
53              Task.Run(() => ConsumeThrottle());
54              Console.WriteLine("Press Any Key to Stop");
55              Console.ReadLine();
56              _tokenSource.Cancel();
57              Start();
58          }
59
60          static void ConsumeThrottle()
61          {
62              var channel = Connection.CreateModel();
63
64              channel.BasicQos(0, 50, false);
65              channel.ExchangeDeclare(exchangeName, "topic");
66              //Queue to send data to Discovery
67              channel.QueueDeclare(discoveryQueue, false, false, false, null);
68              channel.QueueBind(discoveryQueue, exchangeName, discoveryQueue);
69              //Queue to send data to Authentication
70              channel.QueueDeclare(authenticationQueue, false, false, false, null);
71              channel.QueueBind(authenticationQueue, exchangeName,
                  authenticationQueue);
72              //Queue to send data to Data Manager
73              channel.QueueDeclare(dataManagerQueue, false, false, false, null);
74              channel.QueueBind(dataManagerQueue, exchangeName, dataManagerQueue);
75
76              var settings = new BasicProperties()
77              {
78                  ContentType = "application/json",
79                  DeliveryMode = 1
80              };
81
82              var consumer = new JsonObservableSubscription<object>(channel,
                  contextProcessorQueue, true);
83
84              var throttlingConsumer = new
                  ThrottlingConsumer<RabbitMessage<object>>(consumer, 4);
85
86              throttlingConsumer.Subscribe(message =>
87              {
88                  var discoveryPayload =
                      JsonConvert.DeserializeObject<Model.Device>
                      (message.Payload.ToString());
89                  var authenticationPayload =
                      JsonConvert.DeserializeObject<Model.Authentication>
                      (message.Payload.ToString());
90
91                  if (discoveryPayload.Id != Guid.Empty && !string.IsNullOrEmpty
                      (discoveryPayload.Name) && discoveryPayload.Type != 0)
92                  {
93                      var device = deviceList.Where(d => d.Id ==
                          discoveryPayload.Id).SingleOrDefault();
```

```
94                      if (device == null)
95                      {
96                          var Salt = Model.Common.GenerateSalt();
97                          discoveryPayload.Salt = Salt;
98                          deviceList.Add(discoveryPayload);
99                      }
100
101                     var bytes = Encoding.UTF8.GetBytes
                         (JsonConvert.SerializeObject(discoveryPayload));
102                     channel.BasicPublish(exchangeName, discoveryQueue, settings,
                         bytes);
103                     Console.WriteLine("Received:\n");
104                     Console.WriteLine("Device: {0}\n", discoveryPayload.Id);
105                     Console.WriteLine("Thread: {0}\n\n",
                         Thread.CurrentThread.GetHashCode());
106                 }
107                 else if (authenticationPayload.Id != Guid.Empty &&
                      string.IsNullOrEmpty(authenticationPayload.BaseString) &&
                      string.IsNullOrEmpty(authenticationPayload.Password))
108                 {
109                     var device = deviceList.Where(d => d.Id ==
                         authenticationPayload.Id).SingleOrDefault();
110                     authenticationPayload.Salt = device.Salt;
111                     if(device.Token == null)
112                     {
113                         device.Token = Model.Common.GenerateToken();
114                     }
115
116                     authenticationPayload.Token = device.Token;
117
118                     var bytes = Encoding.UTF8.GetBytes
                         (JsonConvert.SerializeObject(authenticationPayload));
119                     channel.BasicPublish(exchangeName, authenticationQueue,
                         settings, bytes);
120                     Console.WriteLine("Received:\n");
121                     Console.WriteLine("Device: {0}\n", discoveryPayload.Id);
122                     Console.WriteLine("Base String: {0}\n",
                         authenticationPayload.BaseString);
123                     Console.WriteLine("Share Secret: {0}\n",
                         authenticationPayload.Password);
124                     Console.WriteLine("Thread: {0}\n\n",
                         Thread.CurrentThread.GetHashCode());
125                 }
126                 else
127                 {
128                     Console.WriteLine("Received (Thread {1}): {0}\n", "INVALID
                         PAYLOAD", Thread.CurrentThread.GetHashCode());
129                 }
130
131             }, _tokenSource.Token);
132
133             var start = throttlingConsumer.Start(_tokenSource.Token,
```

```
                TimeSpan.FromSeconds(1));

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

```