Messaging with RabbitMQ and .NET review part 4: one way messaging with a basic consumer

AUGUST 8, 2016 3 COMMENTS (HTTPS://DOTNETCODR.COM/2016/08/08/MESSAGING-WITH-RABBITMQ-AND-NET-REVIEW-PART-4-ONE-WAY-MESSAGING-WITH-A-BASIC-CONSUMER/#COMMENTS)

Introduction

In the previous post (https://dotnetcodr.com/2016/08/05/messaging-with-rabbitmq-and-net-review-part-3-the-net-client-and-some-initial-code/) we looked at the RabbitMq .NET client. The client is a library that can be downloaded from NuGet and which allows us to work with RabbitMq messages in our .NET projects in an object-oriented way. In particular we saw how to create an exchange, a queue and a binding in code. We also successfully sent a message to the queue we created in a simple .NET console application. We also discussed the notion of durability whereby we can make all resources in RabbitMq fault tolerant so that they survive a server restart.

In this post we'll see how to consume one-way direct messages in code.

We started working on a demo console application in the previous post. We'll be working in it throughout this series so you can open it now. At this point we have one console app in the project which creates the resources for messaging and also sends a message to a queue.

One-way messages

A one-way message is a kind of **message exchange pattern** (MEP). This is the simplest MEP type: a message is sent to the broker which is then processed by the receiver.

Add a new console application called RabbitMq.OneWayMessage.Receiver to our demo project. Add the same RabbitMq .NET client package from NuGet to it as we did previously. The little code we created in the previous post is actually a one-way message sender with its usage of the BasicPublish method.

In the original series we used the QueueingBasicConsumer class to process messages from a queue. However, it is now deprecated and been replaced by DefaultBasicConsumer. It is a basic implementation of the IBasicConsumer interface which handles functions such as message delivery and cancellation. The recommended solution at this point in time is to derive from this class and override its HandleBasicDeliver method. So let's do that!

Add a new class called OneWayMessageReceiver to RabbitMq.OneWayMessage.Receiver. Here's the skeleton which we'll fill in in a short time:

```
using RabbitMQ.Client;

namespace RabbitMq.OneWayMessage.Receiver
{
    public class OneWayMessageReceiver : DefaultBasicConsumer
    {
        private readonly IModel _channel;

        public OneWayMessageReceiver(IModel channel)
        {
            _channel = channel;
        }

        public override void HandleBasicDeliver(string consumerTag, ulong deliveryTag, bool redeliver
        {

        }
    }
}
```

I know that we'll need an IModel object later on for the message acknowledgement. That will become clearer soon.

Before we do anything else in this code let's add the code that will hook up OneWayMessageReceiver with the queue. Add the following bit of code into the Main method of the Program class in RabbitMq.OneWayMessage.Receiver:

```
using RabbitMQ.Client;

namespace RabbitMq.OneWayMessage.Receiver
{
    class Program
    {
        static void Main(string[] args)
        {
            ReceiveSingleOneWayMessage();
        }

        private static void ReceiveSingleOneWayMessage()
        {
            ConnectionFactory connectionFactory = new ConnectionFactory();

            connectionFactory.Port = 5672;
            connectionFactory.HostName = "localhost";
            connectionFactory.UserName = "accountant";
            connectionFactory.Password = "accountant";
            connectionFactory.VirtualHost = "accounting";

            IConnection connection = connectionFactory.CreateConnection();
            IModel channel = connection.CreateModel();
            channel.BasicQos(0, 1, false);
            DefaultBasicConsumer basicConsumer = new OneWayMessageReceiver(channel);
            channel.BasicConsume("my.first.queue", false, basicConsumer);

        }
    }
}
```

The first section in ReceiveSingleOneWayMessage where we set up the RabbitMq connection and channel is identical to what we had in the publisher previously. I know that this is some serious code duplication but let's not worry about that for now. This series is not about clean code and layered applications so it's up to you to organise all these code examples into some well structured application as you develop your messaging project.

The last three lines of code are new. The BasicQos function, where QOS stands for quality of service, sets up the basic behaviour of message handling. The parameters mean that we require one message at a time and we don't want to process any additional messages until the actual one has been processed. You can use these parameters to receive messages in batches. The first integer parameter, i.e. the prefetch size, sets the maximum size of for the messages fetched from the queue where 0 means there is no upper

limit. The second integer, prefetch count, is the number of messages to be fetched from the queue at a time. E.g. if it's set to 5 then if there are 20 messages in the queue then 5 of them will be delivered to the consumer(s) in one batch. The boolean "global" parameter is set to false which means that the prefetch limits are valid for the current channel only, not for the entire connection.

We then declare a new one way message receiver that we derived from DefaultBasicConsumer. We finally call the channel's BasicConsume function. The first parameter is the queue name to be monitored. The second is a boolean which if set to false then we require an acknowledgement from the receiver. Why "false" if we do require an acknowledgement??? The parameter is called "noAck", i.e. "no acknowledgement" where true means that we do not need any acknowledgement and this must be negated with a false. Lastly we pass in our basic consumer.

The implementation of the HandleBasicDeliver function of DefaultBasicConsumer is not too exciting actually since we're not developing a business application. We simply output some parameters to the debug window:

```
1  public override void HandleBasicDeliver(string consumerTag, ulong deliveryTag, bool redelivered, stri
2  {
3          Console.WriteLine("Message received by the consumer. Check the debug window for details.");
4      Debug.WriteLine(string.Concat("Message received from the exchange ", exchange));
5      Debug.WriteLine(string.Concat("Content type: ", properties.ContentType));
6      Debug.WriteLine(string.Concat("Consumer tag: ", consumerTag));
7      Debug.WriteLine(string.Concat("Delivery tag: ", deliveryTag));
8      Debug.WriteLine(string.Concat("Message: ", Encoding.UTF8.GetString(body)));
9      _channel.BasicAck(deliveryTag, false);
10 }
```

The message body is held by the "body" parameter as a byte array. The "properties" parameter is the same IBasicProperties object we saw when publishing a message to the queue. The consumer can read the message properties like the MIME type, the correlation ID, the message ID and a range of other properties that the publisher may have specified. Your code logic can be tweaked based on those parameters. The correlation ID can be used to correlate messages. E.g. if a new order is placed then various systems may want to know about it: accounting, delivery, production, supply planning etc., and they will each handle the new order in some way. The correlation ID from the publisher can be used to check what happened to a particular order in those various systems.

The delivery tag is an integer and is used for acknowledging a message. When RabbitMq has received the acknowledgement then the message is deleted from the queue. This tag usually indicates the position of the message in the queue: 1 is the first message, 2 is the second message etc. according to FIFO. The consumer tag is a unique ID on the message such as "amq.ctag-qCDfYIYQEpGqvAY7t-bhCQ". We'll come back to the redelivered argument in a short bit.

Set a breakpoint in the beginning of HandleBasicDeliver. Run the RabbitMq.OneWayMessage.Receiver application in Visual Studio. If you've followed along the series so far and published a message to RabbitMq in the previous post then code execution should stop at the breakpoint meaning there's a message for our consumer. As you step through the code you should see some debug messages similar to the following:

Message received from the exchange my.first.exchange
Content type: text/plain
Consumer tag: amq.ctag-sr_eLwVpAv8N75fUMfAscA
Delivery tag: 1
Message: This is a message from the RabbitMq .NET driver

Also, if you inspect the redelivered property it should be true. The reason is that we actually viewed the message in the management GUI in the previous post so it's already been handled once but not yet acknowledged by any receiver. If you re-run the publisher code with some modified message and immediately execute the consumer application then "redelivered" should be false.

We finally use the channel to acknowledge the message. We supply the delivery tag and a boolean parameter. If it's set to false then the acknowledgement is only for this message and not for all messages in the queue.

Worker queues

Before we finish this post I want to mention another MEP that is very similar to one-way messages and which can be easily demonstrated with the minimal code we have now. In the worker queues MEP a message is sent by the publisher. There will be many listeners waiting for messages from the same queue. However, those listeners compete to receive the message and only one of them will receive it. The purpose is that if an application is expecting to receive a large load of messages then it can create different threads/processes to process those messages. The benefit is better scalability.

We can simulate this MEP by starting two or more instances of RabbitMq.OneWayMessage.Receiver. If you're not sure how to do it then right-click the project name in VS, select Debug in the context menu and then click Start new instance. Do this process twice so that you have 2 listeners up an running. Then start the publisher multiple times using the same technique so that several messages are published to the queue. You should see that the consumers take turn in receiving the messages in a round-robin fashion:

(https://dotnetcodr.files.wordpress.com/2016/08/worker-queues-mep-demonstration-in-rabbitmq-with-two-receivers.png)

We're done for now. We'll continue with a different, event based way of handling messages in the next post (https://dotnetcodr.com/2016/08/10/messaging-with-rabbitmq-and-net-review-part-5-one-way-messaging-with-an-event-based-consumer/).

View the list of posts on Messaging here (https://dotnetcodr.com/messaging/).
FILED UNDER .NET, MESSAGING TAGGED WITH C#, MESSAGING, RABBITMQ

About Andras Nemes
I'm a .NET/Java developer living and working in Stockholm, Sweden.

3 Responses to *Messaging with RabbitMQ and .NET review part 4: one way messaging with a basic consumer*

Alon says:
November 27, 2017 at 3:54 pm
Hi,

By far this is the most extensive and thorough instruction over the internet regarding RabbitMQ. Thanks a lot!!

While running the sample above, I'm getting an unclear error:

An unhandled exception of type 'RabbitMQ.Client.Exceptions.AlreadyClosedException' occurred in RabbitMQ.Client.dll

Additional information: Already closed: The AMQP operation was interrupted: AMQP close-reason, initiated by Peer, code=404, text="NOT_FOUND – no queue 'my.first.queue' in vhost 'Replication'", classId=60, methodId=20, cause=

While I can see on the console the queue and the message as was added on the previous post.

I know it is an old post but I hope you can help me with this exception as it blocks me on a very basic stage…

The versions I'm using are the latest:
Erlang OTP 20.1
rabbitmq-server-3.6.14
RabbitMQ.Client 5.0.1

and .Net 4.6.1 on VS2015

Thanks in advanced!

Reply
Alon says:
November 27, 2017 at 8:08 pm
Hi,
Me again. I've played around with the Exchange and Queue names and all of a sudden all started to work. Not sure what was the problem beforehand but now all works.

You can just delete these two posts if you like.

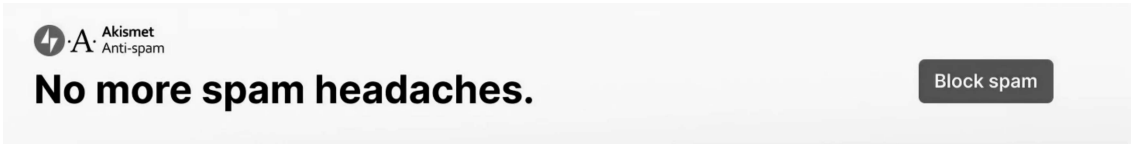
Thanks again for amazing series of posts on RabbitMQ!

Alon
Reply
saineshwar Bageri says:
June 13, 2018 at 5:18 am
Sir after creating this console listener do we want to schedule it in the task windows scheduler

Reply

Blog at WordPress.com.

Advertisements



REPORT THIS AD