Messaging with RabbitMQ and .NET C# part 2: persistence

MAY 1, 2014 3 COMMENTS (HTTPS://DOTNETCODR.COM/2014/05/01/MESSAGING-WITH-RABBITMQ-AND-NET-C-PART-2-PERSISTENCE/#COMMENTS)

Introduction

In the previous part (https://dotnetcodr.com/2014/04/28/messaging-with-rabbitmq-and-net-c-part-1-foundations-and-setup/) of this tutorial we looked at the basics of messaging. We also set up RabbitMQ on Windows and looked at a couple of C# code examples.

We'll continue where we left off so have the RabbitMQ manager UI and the sample .NET console app ready.

Most of the posts on RabbitMQ on this blog are based on the work of RabbitMQ guru Michael Stephenson (http://geekswithblogs.net/michaelstephenson/Default.aspx).

Sending a message in code

Let's first put the code that creates the IConnection into another class. Add a new class called RabbitMqService:

```
public class RabbitMqService
{
    public IConnection GetRabbitMqConnection()
    {
        ConnectionFactory connectionFactory = new ConnectionFactory();
        connectionFactory.HostName = "localhost";
        connectionFactory.UserName = "guest";
        connectionFactory.Password = "guest";

        return connectionFactory.CreateConnection();
    }
}
```

Put the following lines of code…

```
model.QueueDeclare("queueFromVisualStudio", true, false, false, null);
model.ExchangeDeclare("exchangeFromVisualStudio", ExchangeType.Topic);
model.QueueBind("queueFromVisualStudio", "exchangeFromVisualStudio", "superstars");
```

…into a private method for later reference…:

```
private static void SetupInitialTopicQueue(IModel model)
{
    model.QueueDeclare("queueFromVisualStudio", true, false, false, null);
    model.ExchangeDeclare("exchangeFromVisualStudio", ExchangeType.Topic);
    model.QueueBind("queueFromVisualStudio", "exchangeFromVisualStudio", "superstars");
}
```

…so now we have the following code in Program.cs:

```
static void Main(string[] args)
{
    RabbitMqService rabbitMqService = new RabbitMqService();
    IConnection connection = rabbitMqService.GetRabbitMqConnection();
    IModel model = connection.CreateModel();
}

private static void SetupInitialTopicQueue(IModel model)
{
    model.QueueDeclare("queueFromVisualStudio", true, false, false, null);
    model.ExchangeDeclare("exchangeFromVisualStudio", ExchangeType.Topic);
    model.QueueBind("queueFromVisualStudio", "exchangeFromVisualStudio", "superstars");
}
```

In Main we'll create some properties and we'll set the message persistence to non-persistent – see below for details:

```
IBasicProperties basicProperties = model.CreateBasicProperties();
basicProperties.SetPersistent(false);
```

We need to send our message in byte array format:

```
byte[] payload = Encoding.UTF8.GetBytes("This is a message from Visual Studio");
```

We then construct the address for the exchange we created in the previous part:

```
PublicationAddress address = new PublicationAddress(ExchangeType.Topic, "exchangeFromVisualStudio", "s
```

Finally we send the message:

```
model.BasicPublish(address, basicProperties, payload);
```

Run the application. Go to the RabbitMQ management UI, navigate to queueFromVisualStudio and you should be able to extract the message:

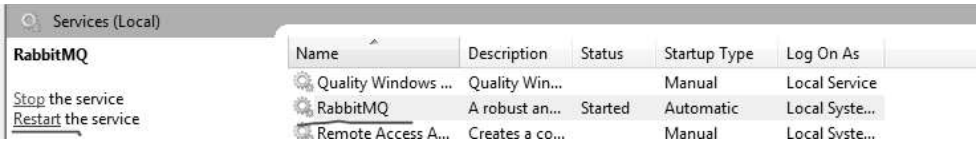
(https://dotnetcodr.files.wordpress.com/2014/02/messagefromvisualstudioincode.png)

Queue and exchange persistence

There are two types of queues and exchanges from a persistence point of view:

- Durable: messages are saved to disk so they are available even after a server restart. There's some overhead incurred while reading and saving messages
- Non-durable: messages are persisted in memory. They disappear after a server restart but offer a faster service

Keep these advantages and disadvantages in mind when you're deciding which persistence strategy to go for. Recall that we set persistence to non-durable for the message in the previous section. For a quick server restart open the Services console and restart the Windows service called RabbitMQ:



(https://dotnetcodr.files.wordpress.com/2014/02/restartrabbitmqservice.png)

Go back to the RabbitMQ managenment UI on http://localhost:15672/ (http://localhost:15672/) If you were logged on before then you have probably been logged out. Navigate to queueFromVisualStudio, check the available messages and you'll see that there's none. The queue is still available as we set it to durable in code:

```
1 | model.QueueDeclare("queueFromVisualStudio", true, false, false, null);
```

The second parameter 'true' means that the queue itself is durable. Had we set this to false, we would have lost the queue as well in the server restart. The exchange "exchangeFromVisualStudio" itself was non-durable so it was lost. Remember the following exchange creation code:

```
1 | model.ExchangeDeclare("exchangeFromVisualStudio", ExchangeType.Topic);
```

We haven't specified the durable property so it was set to false by default. The ExchangeDeclare method has an overload which allows us to declare a durable exchange:

```
1 | model.ExchangeDeclare("exchangeFromVisualStudio", ExchangeType.Topic, true);
```

Also, recall that we created an exchange called newexchange through the UI in the previous post and it was set to durable in the available options. That's the reason why it is still available in the list of exchanges but exchangeFromVisualStudio isn't:

Name	Type	Policy	Parameters	Message rate in	Message rate out
(AMQP default)	direct		D		
amq.direct	direct		D		
amq.fanout	fanout		D		
amq.headers	headers		D		
amq.match	headers		D		
amq.rabbitmq.log	topic		D		
amq.rabbitmq.trace	topic		D		
amq.topic	topic		D		
newexchange	fanout		D		

(https://dotnetcodr.files.wordpress.com/2014/02/durableexchangeavailable.png)

Add a private method to set up durable components:

```
1 | private static void SetupDurableElements(IModel model)
2 | {
3 |     model.QueueDeclare("DurableQueue", true, false, false, null);
4 |     model.ExchangeDeclare("DurableExchange", ExchangeType.Topic, true);
5 |     model.QueueBind("DurableQueue", "DurableExchange", "durable");
6 | }
```

Call this method from Main after…

```
1 | IModel model = connection.CreateModel();
```

Comment out the rest of the code in Main or put it in another method for later reference. Now we have a durable exchange and a durable queue. Let's send a message to it:

```
private static void SendDurableMessageToDurableQueue(IModel model)
{
        IBasicProperties basicProperties = model.CreateBasicProperties();
    basicProperties.SetPersistent(true);
    byte[] payload = Encoding.UTF8.GetBytes("This is a persistent message from Visual Studio");
    PublicationAddress address = new PublicationAddress(ExchangeType.Topic, "DurableExchange", "durabl
    model.BasicPublish(address, basicProperties, payload);
}
```

Call this method from Main and then check in the management UI that the message has been delivered. Restart the RabbitMQ server and the message should still be available:

Overview					Messages			Message rates		
Name	Exclusive	Parameters	Policy	Status	Ready	Unacked	Total	incoming	deliver / get	ack
DurableQueue		D		Idle	1	0	1			

(https://dotnetcodr.files.wordpress.com/2014/02/durablemessagestillavailable.png)

Therefore we can set the persistence property on three levels:

- Queue
- Exchange
- Message

Before I forget: you can specify an empty string as the exchange name as follows.

```
model.BasicPublish("", "key", basicProperties, payload);
```

The empty string will be translated into the default exchange:

Name	Type	Policy	Parameters	Message rate in	Message rate out
(AMQP default)	direct		D		

(https://dotnetcodr.files.wordpress.com/2014/02/defaultexchange.png)

In the next part (https://dotnetcodr.com/2014/05/05/messaging-with-rabbitmq-and-net-c-part-3-message-exchange-patterns/) of the series we'll be looking at messaging patterns.

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

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

3 Responses to *Messaging with RabbitMQ and .NET C# part 2: persistence*

Riddik says:
May 13, 2016 at 2:13 pm
Hi! I did find calling SetupInitialTopicQueue in the code. There was no message appeared till i add SetupInitialTopicQueue(model) in Main fuction. Maybe i missed some part of your article maybe not me)
Thanks a lot. Anyway really usefull!

Reply

Jarosław Jusiak says:
October 1, 2016 at 8:28 pm
Good point! I was wondering why I can't see queue.

Reply

Andrews says:

October 11, 2017 at 11:31 am
Hi
In a single c# solution,
Is it possible to handle multiple rabbitmq exchange types.

for ex:
ExchangeType.direct
ExchangeType.Fanout.

I want to use both cases. Kindly advice.

Reply