# Messaging with RabbitMQ and .NET review part 8: routing and topics

AUGUST 25, 2016 1 COMMENT (HTTPS://DOTNETCODR.COM/2016/08/25/MESSAGING-WITH-RABBITMQ-AND-NET-REVIEW-PART-8-ROUTING-AND-TOPICS/#COMMENTS)

**Introduction**

In the previous post (https://dotnetcodr.com/2016/08/18/messaging-with-rabbitmq-and-net-review-part-7-two-way-messaging/) we looked at two-way messaging in RabbitMq. This messaging type corresponds to the Remote Procedure Call (RPC) messaging pattern. RPC is slightly different from the previous MEPs in that there's a response queue involved. The sender sends an initial message to a destination queue via the default exchange. The message properties include a temporary queue where the consumer can reply. The receiver processes the message and responds using the response queue extracted from the message properties. The sender then processes the response. We managed to set up a rudimentary chat application in our demo project at the end of the post.

In this post we'll concentrate on two message filtering techniques: routing keys and topics. The two are quite similar so a single post is enough to handle them both.

**Routing keys**

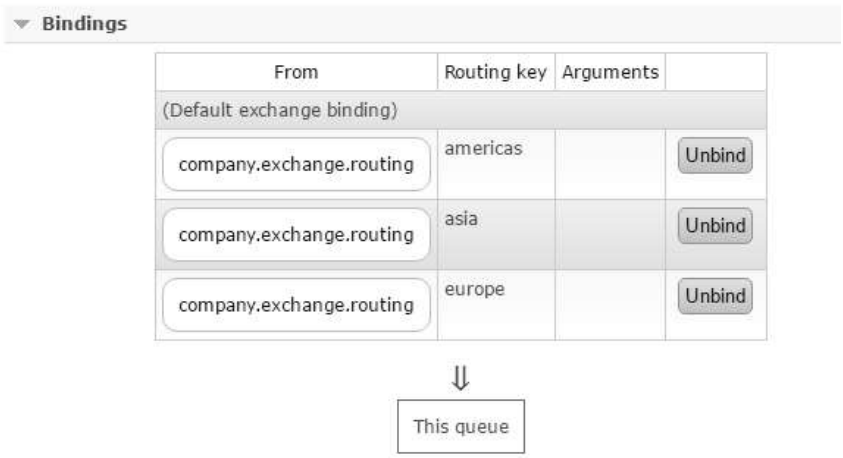
Here the client sends a message to an exchange and attaches a routing key to it. The message is sent to all queues with the matching routing key. Each queue has a receiver attached which will process the message – or more than one if you want to distribute the load, see an example earlier in this series. We'll initiate a dedicated message exchange and not use the default one in the below example. Note that a queue can be dedicated to one or more routing keys.

We'll be working in the same .NET demo console project as before. At present we have one project to set up various types of sender: RPC, Fanout, Direct. Here's the example code for Program.cs that includes the new method SetUpDirectExchangeWithRoutingKey:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using RabbitMQ.Client;
using RabbitMQ.Client.MessagePatterns;
using RabbitMQ.Client.Events;
using System.Threading;

namespace RabbitMqNetTests
{
    class Program
    {
        static void Main(string[] args)
        {
            SetUpDirectExchangeWithRoutingKey();
        }

        private static void SetUpDirectExchangeWithRoutingKey()
        {
            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.ExchangeDeclare("company.exchange.routing", ExchangeType.Direct, true, false, nul
            channel.QueueDeclare("company.exchange.queue", true, false, false, null);
            channel.QueueBind("company.exchange.queue", "company.exchange.routing", "asia");
            channel.QueueBind("company.exchange.queue", "company.exchange.routing", "americas");
            channel.QueueBind("company.exchange.queue", "company.exchange.routing", "europe");

            IBasicProperties properties = channel.CreateBasicProperties();
            properties.Persistent = true;
            properties.ContentType = "text/plain";
            PublicationAddress address = new PublicationAddress(ExchangeType.Direct, "company.exchang
            channel.BasicPublish(address, properties, Encoding.UTF8.GetBytes("The latest news from As

            address = new PublicationAddress(ExchangeType.Direct, "company.exchange.routing", "europe
            channel.BasicPublish(address, properties, Encoding.UTF8.GetBytes("The latest news from Eu

            address = new PublicationAddress(ExchangeType.Direct, "company.exchange.routing", "americ
            channel.BasicPublish(address, properties, Encoding.UTF8.GetBytes("The latest news from th

                        address = new PublicationAddress(ExchangeType.Direct, "company.exchange.routi
            channel.BasicPublish(address, properties, Encoding.UTF8.GetBytes("The latest news from Af

            address = new PublicationAddress(ExchangeType.Direct, "company.exchange.routing", "austra
            channel.BasicPublish(address, properties, Encoding.UTF8.GetBytes("The latest news from Au

            channel.Close();
            connection.Close();
        }

        private static void RunRpcQueue()
        {
            //code ignored
        }

        private static void SendRpcMessagesBackAndForth(IModel channel)
        {
            //code ignored
        }

        private static void SetUpFanoutExchange()
```

```
72          {
73              //code ignored
74          }
75
76          private static void SetUpDirectExchange()
77          {
78              //code ignored
79          }
80      }
81  }
```

We set up a normal exchange of type direct and attach a queue with multiple routing keys to it: asia, americas and europe. If you run the above code then the bindings will be visible as follows in the RabbitMq management console:




(https://dotnetcodr.files.wordpress.com/2016/08/single-queue-with-multiple-routing-keys-attached-to-same-exchange-rabbitmq.png)

We then publish five messages, one with each configured routing key and two more with no corresponding binding. Note that the above is to demonstrate that the same queue can be bound to an exchange with multiple routing keys. It's perfectly feasible to set up 3 separate queues instead where each queue will have its own routing key. The effect of the above code is that if a message with any of the given 3 routing keys, i.e. "asia", "americas" or "europe" is sent to the exchange "company.exchange.routing" then that message will be forwarded to the queue called "company.exchange.queue". The messages with "africa" and "australia" are not expected to be routed to the queue. Those messages will be discarded as there's no matching queue to which they could be forwarded. The management UI is giving us a hint that this is indeed the case:

| Overview | | | | Messages | | | Message rates | | | +/- |
|---|---|---|---|---|---|---|---|---|---|---|
| Virtual host | Name | Features | State | Ready | Unacked | Total | incoming | deliver / get | ack | |
| accounting | company.exchange.queue | D | idle | 3 | 0 | 3 | 0.00/s | 0.00/s | 0.00/s | |
| accounting | my.first.queue | D | idle | 0 | 0 | 0 | | | | |

(https://dotnetcodr.files.wordpress.com/2016/08/messages-with-false-routing-keys-were-not-directed-to-the-rabbitmq-queue.png)

The queue received 3 messages from the exchange, the other two were not routed anywhere since there is no single queue with the routing key "africa" or "australia".

The code for the receiver will be the same as what we saw before in this (https://dotnetcodr.com/2016/08/10/messaging-with-rabbitmq-and-net-review-part-5-one-way-messaging-with-an-event-based-consumer/), alternatively this post (https://dotnetcodr.com/2016/08/08/messaging-with-rabbitmq-and-net-review-part-4-one-way-messaging-with-a-basic-consumer/), depending on how you prefer to build your receiver: derive from DefaultBasicConsumer or go with the event based variant.

Here's the receiver that will monitor the configured queue:

```
using RabbitMQ.Client;
using RabbitMQ.Client.Events;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

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

        private static void ReceiveMessagesWithEvents()
        {
            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);
            EventingBasicConsumer eventingBasicConsumer = new EventingBasicConsumer(channel);

            eventingBasicConsumer.Received += (sender, basicDeliveryEventArgs) =>
            {
                IBasicProperties basicProperties = basicDeliveryEventArgs.BasicProperties;
                Console.WriteLine("Message received by the event based consumer. Check the debug windo
                Debug.WriteLine(string.Concat("Message received from the exchange ", basicDeliveryEve
                Debug.WriteLine(string.Concat("Routing key: ", basicDeliveryEventArgs.RoutingKey));
                Debug.WriteLine(string.Concat("Message: ", Encoding.UTF8.GetString(basicDeliveryEvent
                channel.BasicAck(basicDeliveryEventArgs.DeliveryTag, false);
            };

            channel.BasicConsume("company.exchange.queue", false, eventingBasicConsumer);
        }
    }
}
```

If you run the above code the consumer should receive 3 messages as expected:

Message received from the exchange company.exchange.routing
Routing key: asia
Message: The latest news from Asia!
Message received from the exchange company.exchange.routing
Routing key: europe
Message: The latest news from Europe!
Message received from the exchange company.exchange.routing
Routing key: americas
Message: The latest news from the Americas!

It's up to the consumer to analyse the routing key and execute some action accordingly.

**Topics**

The Topic MEP is similar to Routing. The sender sends a message to an exchange with a routing key attached. The message will be forwarded to queues with a matching **expression**. The routing key can include special characters:

- ‘*’ to replace one word
- ‘#’ to replace 0 or more words

The purpose of this pattern is that we can specify a routing pattern for our queue, sort of like a regular expression, as the routing key it is interested in, e.g. “#world#”. Then the sender sends a message with a routing key “world news” and then another one with a routing key “the end of the world” and the queue will receive both messages. If there are no queues with a matching routing key pattern then the message is discarded.

Here’s the sender portion of the code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using RabbitMQ.Client;
using RabbitMQ.Client.MessagePatterns;
using RabbitMQ.Client.Events;
using System.Threading;

namespace RabbitMqNetTests
{
    class Program
    {
        static void Main(string[] args)
        {
            SetUpTopicsExchange();
        }

        private static void SetUpTopicsExchange()
        {
            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.ExchangeDeclare("company.exchange.topic", ExchangeType.Topic, true, false, null);
            channel.QueueDeclare("company.queue.topic", true, false, false, null);
            channel.QueueBind("company.queue.topic", "company.exchange.topic", "*.world");
            channel.QueueBind("company.queue.topic", "company.exchange.topic", "world.#");

            IBasicProperties properties = channel.CreateBasicProperties();
            properties.Persistent = true;
            properties.ContentType = "text/plain";
            PublicationAddress address = new PublicationAddress(ExchangeType.Topic, "company.exchange
            channel.BasicPublish(address, properties, Encoding.UTF8.GetBytes("This is some random new

            address = new PublicationAddress(ExchangeType.Topic, "company.exchange.topic", "news.of.t
            channel.BasicPublish(address, properties, Encoding.UTF8.GetBytes("trololo"));

            address = new PublicationAddress(ExchangeType.Topic, "company.exchange.topic", "the world
            channel.BasicPublish(address, properties, Encoding.UTF8.GetBytes("whatever"));

            address = new PublicationAddress(ExchangeType.Topic, "company.exchange.topic", "the.world
            channel.BasicPublish(address, properties, Encoding.UTF8.GetBytes("Hello"));

            address = new PublicationAddress(ExchangeType.Topic, "company.exchange.topic", "world.new
            channel.BasicPublish(address, properties, Encoding.UTF8.GetBytes("It's Friday night"));

            address = new PublicationAddress(ExchangeType.Topic, "company.exchange.topic", "world new
            channel.BasicPublish(address, properties, Encoding.UTF8.GetBytes("No more tears"));

            address = new PublicationAddress(ExchangeType.Topic, "company.exchange.topic", "beautiful
            channel.BasicPublish(address, properties, Encoding.UTF8.GetBytes("The world is beautiful"

            channel.Close();
            connection.Close();
        }
    }
}
```

We set up an exchange of type Topic and bind two queues to it, each with a different routing key. We then send various messages to the exchange where each message has a different routing key. Only 2 of the 6 messages will be forwarded to the queue. Can you guess which ones?

Here's the receiver's code. It is identical to the receiver above. The only difference is the queue name:

```
using RabbitMQ.Client;
using RabbitMQ.Client.Events;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

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

        private static void ReceiveMessagesWithEvents()
        {
            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);
            EventingBasicConsumer eventingBasicConsumer = new EventingBasicConsumer(channel);

            eventingBasicConsumer.Received += (sender, basicDeliveryEventArgs) =>
            {
                IBasicProperties basicProperties = basicDeliveryEventArgs.BasicProperties;
                Console.WriteLine("Message received by the event based consumer. Check the debug wind
                Debug.WriteLine(string.Concat("Message received from the exchange ", basicDeliveryEve
                Debug.WriteLine(string.Concat("Routing key: ", basicDeliveryEventArgs.RoutingKey));
                Debug.WriteLine(string.Concat("Message: ", Encoding.UTF8.GetString(basicDeliveryEvent
                channel.BasicAck(basicDeliveryEventArgs.DeliveryTag, false);
            };

            channel.BasicConsume("company.queue.topic", false, eventingBasicConsumer);
        }
    }
}
```

The consumer received the following 2 messages:

Message received from the exchange company.exchange.topic
Routing key: world.news.and.more
Message: It's Friday night
Message received from the exchange company.exchange.topic
Routing key: beautiful.world
Message: The world is beautiful

The other 4 were discarded as their routing key patterns didn't match any of the queues.

In the next post (https://dotnetcodr.com/2016/08/29/messaging-with-rabbitmq-and-net-review-part-9-headers/) we'll look at a different message filtering technique called Headers.

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.

## One Response to *Messaging with RabbitMQ and .NET review part 8: routing and topics*

**Anubhav Tiwari says:**
November 28, 2019 at 8:04 am
Hi Andras
Great with updated part. I think you have deleted your exsiting article. I was looking for the article of adding queue with the container. But I couldnt find it.

**Reply**