**Routing**

In the [**previous tutorial**](https://www.rabbitmq.com/tutorials/tutorial-three-dotnet.html) we built a simple logging system. We were able to broadcast log messages to many receivers.

In this tutorial we're going to add a feature to it - we're going to make it possible to subscribe only to a subset of the messages. For example, we will be able to direct only critical error messages to the log file (to save disk space), while still being able to print all of the log messages on the console.

**Bindings**

In previous examples we were already creating bindings. You may recall code like:

channel.QueueBind(queue: queueName,

exchange: "logs",

routingKey: "");

A binding is a relationship between an exchange and a queue. This can be simply read as: the queue is interested in messages from this exchange.

Bindings can take an extra routingKey parameter. To avoid the confusion with a BasicPublish parameter we're going to call it a binding key. This is how we could create a binding with a key:

channel.QueueBind(queue: queueName,

exchange: "direct\_logs",

routingKey: "black");

The meaning of a binding key depends on the exchange type. The fanout exchanges, which we used previously, simply ignored its value.

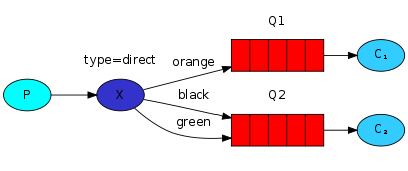
**Direct exchange**

Our logging system from the previous tutorial broadcasts all messages to all consumers. We want to extend that to allow filtering messages based on their severity. For example we may want the script which is writing log messages to the disk to only receive critical errors, and not waste disk space on warning or info log messages.

We were using a fanout exchange, which doesn't give us much flexibility - it's only capable of mindless broadcasting.

We will use a direct exchange instead. The routing algorithm behind a direct exchange is simple - a message goes to the queues whose binding key exactly matches the routing key of the message.

To illustrate that, consider the following setup:

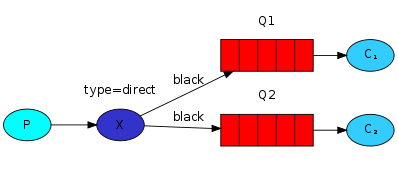


digraph { bgcolor=transparent; truecolor=true; rankdir=LR; node [style="filled"]; // P [label="P", fillcolor="#00ffff"]; subgraph cluster\_X1 { label="type=direct"; color=transparent; X [label="X", fillcolor="#3333CC"]; }; subgraph cluster\_Q1 { label="Q1"; color=transparent; Q1 [label="{||||}", fillcolor="red", shape="record"]; }; subgraph cluster\_Q2 { label="Q2"; color=transparent; Q2 [label="{||||}", fillcolor="red", shape="record"]; }; C1 [label=<C<font point-size="7">1</font>>, fillcolor="#33ccff"]; C2 [label=<C<font point-size="7">2</font>>, fillcolor="#33ccff"]; // P -> X; X -> Q1 [label="orange"]; X -> Q2 [label="black"]; X -> Q2 [label="green"]; Q1 -> C1; Q2 -> C2; }

In this setup, we can see the direct exchange X with two queues bound to it. The first queue is bound with binding key **orange**, and the second has two bindings, one with binding key **black** and the other one with **green.**

In such a setup a message published to the exchange with a routing key orange will be routed to queue Q1. Messages with a routing key of black or green will go to Q2. **All other messages will be discarded**.

**Multiple bindings**



digraph { bgcolor=transparent; truecolor=true; rankdir=LR; node [style="filled"]; // P [label="P", fillcolor="#00ffff"]; subgraph cluster\_X1 { label="type=direct"; color=transparent; X [label="X", fillcolor="#3333CC"]; }; subgraph cluster\_Q1 { label="Q1"; color=transparent; Q1 [label="{||||}", fillcolor="red", shape="record"]; }; subgraph cluster\_Q2 { label="Q2"; color=transparent; Q2 [label="{||||}", fillcolor="red", shape="record"]; }; C1 [label=<C<font point-size="7">1</font>>, fillcolor="#33ccff"]; C2 [label=<C<font point-size="7">2</font>>, fillcolor="#33ccff"]; // P -> X; X -> Q1 [label="black"]; X -> Q2 [label="black"]; Q1 -> C1; Q2 -> C2; }

It is perfectly legal to bind multiple queues with the same binding key. In our example we could add a binding between X and Q1 with binding key black. In that case, the direct exchange will behave like fanout and will broadcast the message to all the matching queues. A message with routing key black will be delivered to both Q1 and Q2.

**Emitting logs**

We'll use this model for our logging system. Instead of fanout we'll send messages to a direct exchange. We will supply the log severity as a routing key. That way the receiving script will be able to select the severity it wants to receive. Let's focus on emitting logs first.

As always, we need to create an exchange first:

channel.ExchangeDeclare(exchange: "direct\_logs", type: "direct");

And we're ready to send a message:

**var** body = Encoding.UTF8.GetBytes(message);

channel.BasicPublish(exchange: "direct\_logs",

routingKey: severity,

basicProperties: **null**,

body: body);

To simplify things we will assume that 'severity' can be one of 'info', 'warning', 'error'.

**Subscribing**

Receiving messages will work just like in the previous tutorial, with one exception - we're going to create a new binding for each severity we're interested in.

**var** queueName = channel.QueueDeclare().QueueName;

**foreach**(**var** severity **in** args)

{

channel.QueueBind(queue: queueName,

exchange: "direct\_logs",

routingKey: severity);

}

**Putting it all together**



**digraph { bgcolor=transparent; truecolor=true; rankdir=LR; node [style="filled"]; // P [label="P", fillcolor="#00ffff"]; subgraph cluster\_X1 { label="type=direct"; color=transparent; X [label="X", fillcolor="#3333CC"]; }; subgraph cluster\_Q2 { label="amqp.gen-S9b..."; color=transparent; Q2 [label="{||||}", fillcolor="red", shape="record"]; }; subgraph cluster\_Q1 { label="amqp.gen-Ag1..."; color=transparent; Q1 [label="{||||}", fillcolor="red", shape="record"]; }; C1 [label=<C<font point-size="7">1</font>>, fillcolor="#33ccff"]; C2 [label=<C<font point-size="7">2</font>>, fillcolor="#33ccff"]; // P -> X; X -> Q1 [label="info"]; X -> Q1 [label="error"]; X -> Q1 [label="warning"]; X -> Q2 [label="error"]; Q1 -> C2; Q2 -> C1; }**

**EmitLogDirect.cs class**:

**using** System;

**using** System.Linq;

**using** RabbitMQ.Client;

**using** System.Text;

**class** **EmitLogDirect**

{

**public** **static** **void** **Main**(**string**[] args)

{

**var** factory = **new** ConnectionFactory() { HostName = "localhost" };

using(**var** connection = factory.CreateConnection())

using(**var** channel = connection.CreateModel())

{

channel.ExchangeDeclare(exchange: "direct\_logs",

type: "direct");

**var** severity = (args.Length > 0) ? args[0] : "info";

**var** message = (args.Length > 1)

? **string**.Join(" ", args.Skip( 1 ).ToArray())

: "Hello World!";

**var** body = Encoding.UTF8.GetBytes(message);

channel.BasicPublish(exchange: "direct\_logs",

routingKey: severity,

basicProperties: **null**,

body: body);

Console.WriteLine(" [x] Sent '{0}':'{1}'", severity, message);

}

Console.WriteLine(" Press [enter] to exit.");

Console.ReadLine();

}

}

**ReceiveLogsDirect.cs:**

**using** System;

**using** RabbitMQ.Client;

**using** RabbitMQ.Client.Events;

**using** System.Text;

**class** **ReceiveLogsDirect**

{

**public** **static** **void** **Main**(**string**[] args)

{

**var** factory = **new** ConnectionFactory() { HostName = "localhost" };

using(**var** connection = factory.CreateConnection())

using(**var** channel = connection.CreateModel())

{

channel.ExchangeDeclare(exchange: "direct\_logs",

type: "direct");

**var** queueName = channel.QueueDeclare().QueueName;

**if**(args.Length < 1)

{

Console.Error.WriteLine("Usage: {0} [info] [warning] [error]",

Environment.GetCommandLineArgs()[0]);

Console.WriteLine(" Press [enter] to exit.");

Console.ReadLine();

Environment.ExitCode = 1;

**return**;

}

**foreach**(**var** severity **in** args)

{

channel.QueueBind(queue: queueName,

exchange: "direct\_logs",

routingKey: severity);

}

Console.WriteLine(" [\*] Waiting for messages.");

**var** consumer = **new** EventingBasicConsumer(channel);

consumer.Received += (model, ea) =>

{

**var** body = ea.Body;

**var** message = Encoding.UTF8.GetString(body);

**var** routingKey = ea.RoutingKey;

Console.WriteLine(" [x] Received '{0}':'{1}'",

routingKey, message);

};

channel.BasicConsume(queue: queueName,

noAck: **true**,

consumer: consumer);

Console.WriteLine(" Press [enter] to exit.");

Console.ReadLine();

}

}

}

Compile as usual (see [**tutorial one**](https://www.rabbitmq.com/tutorials/tutorial-one-dotnet.html) for compilation advice).

If you want to save only 'warning' and 'error' (and not 'info') log messages to a file, just open a console and type:

$ ReceiveLogsDirect.exe warning error > logs\_from\_rabbit.log

If you'd like to see all the log messages on your screen, open a new terminal and do:

$ ReceiveLogsDirect.exe info warning error

**[**\***]** Waiting **for** logs. To exit press CTRL+C

And, for example, to emit an error log message just type:

$ EmitLogDirect.exe error "Run. Run. Or it will explode."

**[**x**]** Sent 'error':'Run. Run. Or it will explode.'

(Full source code for [**(EmitLogDirect.cs source)**](https://github.com/rabbitmq/rabbitmq-tutorials/blob/master/dotnet/EmitLogDirect/EmitLogDirect.cs) and [**(ReceiveLogsDirect.cs source)**](https://github.com/rabbitmq/rabbitmq-tutorials/blob/master/dotnet/ReceiveLogsDirect/ReceiveLogsDirect.cs))

Move on to [**tutorial 5**](https://www.rabbitmq.com/tutorials/tutorial-five-dotnet.html) to find out how to listen for messages based on a pattern.