RabbitMQ for Windows: Headers Exchanges

29 May, 2012. It was a Tuesday.

Posts In This Series

This is the eighth and final installment to the series: RabbitMQ for Windows. In the last installment, we walked through creating a topic exchange example. As the last installment, we'll walk through a headers exchange example.

Headers exchanges examine the message headers to determine which queues a message should be routed to. As discussed earlier in this series, headers exchanges are similar to topic exchanges in that they allow you to specify multiple criteria, but offer a bit more flexibility in that the headers can be constructed using a wider range of data types (1).

To subscribe to receive messages from a headers exchange, a dictionary of headers is specified as part of the binding arguments. In addition to the headers, a key of "x-match" is also included in the dictionary with a value of "all", specifying that messages must be published with all the specified headers in order to match, or "any", specifying that the message needs to only have one of the specified headers specified.

As our final example, we'll create a Producer application which publishes the message "Hello, World!" using a headers exchange. Here's our Producer code:

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Text;
using System.Threading;
using RabbitMQ.Client;
using RabbitMQ.Client.Framing.v0_9_1;

namespace Producer
{
  class Program
  {
    const string ExchangeName = "header-exchange-example";

    static void Main(string[] args)
    {
      var connectionFactory = new ConnectionFactory();
```

```
connectionFactory.HostName = "localhost";

IConnection connection = connectionFactory.CreateConnection();
IModel channel = connection.CreateModel();
channel.ExchangeDeclare(ExchangeName, ExchangeType.Headers, false, true, null);
byte[] message = Encoding.UTF8.GetBytes("Hello, World!");

var properties = new BasicProperties();
properties.Headers = new Dictionary<string, object>();
properties.Headers.Add("key1", "12345");

TimeSpan time = TimeSpan.FromSeconds(10);
var stopwatch = new Stopwatch();
Console.WriteLine("Running for {0} seconds", time.ToString("ss"));
stopwatch.Start();
var messageCount = 0;

while (stopwatch.Elapsed < time)
{
  channel.BasicPublish(ExchangeName, "", properties, message);
  messageCount++;
  Console.Write("Time to complete: {0} seconds - Messages published: {1}\r", (time - stopwatch.El
  Thread.Sleep(1000);
}

Console.Write(new string(' ', 70) + "\r");
Console.WriteLine("Press any key to exit");
Console.ReadKey();
message = Encoding.UTF8.GetBytes("quit");
channel.BasicPublish(ExchangeName, "", properties, message);
connection.Close();
```

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

In the Producer, we've used a generic dictionary of type Dictionary<string, object> and added a single key "key1" with a value of "12345". As with our previous example, we're using a stopwatch as a way to publish messages continually for 10 seconds.

For our Consumer application, we can use an "x-match" argument of "all" with the single key/value pair specified by the Producer, or we can use an "x-match" argument of "any" which includes the key/value pair specified by the Producer along with other potential matches. We'll use the latter for our example. Here's our Consumer code:

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Text;
using RabbitMQ.Client;
using RabbitMQ.Client.Events;

namespace Consumer
{
  class Program
  {
    const string QueueName = "header-exchange-example";
    const string ExchangeName = "header-exchange-example";

    static void Main(string[] args)
    {
      var connectionFactory = new ConnectionFactory();
      connectionFactory.HostName = "localhost";
```

```
      IConnection connection = connectionFactory.CreateConnection();
      IModel channel = connection.CreateModel();
      channel.ExchangeDeclare(ExchangeName, ExchangeType.Headers, false, true, null);
      channel.QueueDeclare(QueueName, false, false, true, null);

      IDictionary specs = new Dictionary<string object ,>();
      specs.Add("x-match", "any");
      specs.Add("key1", "12345");
      specs.Add("key2", "123455");
      channel.QueueBind(QueueName, ExchangeName, string.Empty, specs);

      channel.StartConsume(QueueName, MessageHandler);
      connection.Close();
    }

    public static void MessageHandler(IModel channel, DefaultBasicConsumer consumer, BasicDeliverEventA
    {
      string message = Encoding.UTF8.GetString(eventArgs.Body);
      Console.WriteLine("Message received: " + message);
      foreach (object headerKey in eventArgs.BasicProperties.Headers.Keys)
      {
        Console.WriteLine(headerKey + ": " + eventArgs.BasicProperties.Headers[headerKey]);
      }

      if (message == "quit")
        channel.BasicCancel(consumer.ConsumerTag);
    }
  }
}
```

Rather than handling our messages inline as we've done in previous examples, this example uses an extension method named StartConsume() which accepts a callback to be invoked each time a message is received. Here's the extension method used by our example:

```
using System;
using System.IO;
using System.Threading;
using RabbitMQ.Client;
using RabbitMQ.Client.Events;

namespace Consumer
{

  public static class ChannelExtensions
  {
    public static void StartConsume(this IModel channel, string queueName,  Action<IModel, DefaultBasic
    {
      var consumer = new QueueingBasicConsumer(channel);
      channel.BasicConsume(queueName, true, consumer);

      while (true)
      {
        try
        {
          var eventArgs = (BasicDeliverEventArgs)consumer.Queue.Dequeue();
          new Thread(() => callback(channel, consumer, eventArgs)).Start();
        }
        catch (EndOfStreamException)
        {
          // The consumer was cancelled, the model closed, or the connection went away.
```

```
            break;
          }
        }
      }
    }
  }
```

Setting our solution to run both the Producer and Consumer applications upon startup, running our example produces output similar to the following:

Producer

```
Running for 10 seconds
Time to complete: 08 seconds - Messages published: 2
```

Consumer

```
Message received: Hello, World!
key1: 12345
Message received: Hello, World!
key1: 12345
```

That concludes our headers exchange example as well as the RabbitMQ for Windows series. For more information on working with RabbitMQ, see the documentation at http://www.rabbitmq.com or the purchase the book RabbitMQ in Action by Alvaro Videla and Jason Williams. I hope you enjoyed the series.

Footnotes:

1 – See http://www.rabbitmq.com/amqp-0-9-1-errata.html#section_3 and http://hg.rabbitmq.com/rabbitmq-dotnet-client/diff/4def852523e2/projects/client/RabbitMQ.Client/src/client/impl/WireFormatting.cs for supported field types.

← RabbitMQ for Windows: Topic Exchanges

Expected Objects Custom Comparisons →

Recent Author Posts

- Hello, React! - A Beginner's Setup Tutorial
- Exploring TypeScript

- Git on Windows: Whence Cometh Configuration
- Separation of Concerns: Application Builds & Continuous Integration
- Survey of Entity Framework Unit of Work Patterns
- Introducing NUnit.Specifications
- Being Agile
- Expected Objects Custom Comparisons
- RabbitBus: An Example
- Adventures in Debugging: The NHibernate 'don't flush the Session' Error

Recent Site Posts

- Domain-Driven Refactoring: Defactoring and Pushing Behavior Down
- Domain-Driven Refactoring: Extracting Domain Services
- Domain-Driven Refactoring: Long Methods
- Domain-Driven Refactoring: Procedural Beginnings
- Domain-Driven Refactoring: Intro
- Local Development with Azure Service Bus
- Taming the WSL 2 Resource Monster
- Crossing the Generics Divide
- OpenTelemetry 1.0 Extensions Released
- Choosing a ServiceLifetime

Authors

- Andrew Siemer
- Chad Myers
- Chris Missal
- Chris Patterson

- Derek Greer
- Derik Whittaker
- Eric Anderson
- Eric Hexter
- Gabriel Schenker
- Gregory Long
- Hugo Bonacci
- James Gregory
- Jimmy Bogard
- John Teague
- Josh Arnold
- Joshua Flanagan
- Joshua Lockwood
- Keith Dahlby
- Matt Hinze
- Patrick Lioi
- Rod Paddock
- Ryan Rauh
- Ryan Svihla
- Scott Densmore
- Sean Biefeld
- Sean Chambers
- Sharon Cichelli
- Steve Donie