RabbitMQ for Windows: Building Your First Application

7 March, 2012. It was a Wednesday.

Posts In This Series

- RabbitMQ for Windows: Introduction
- RabbitMQ for Windows: Building Your First Application
- RabbitMQ for Windows: Hello World Review
- RabbitMQ for Windows: Exchange Types
- RabbitMQ for Windows: Direct Exchanges
- RabbitMQ for Windows: Fanout Exchanges
- RabbitMQ for Windows: Topic Exchanges
- RabbitMQ for Windows: Headers Exchanges

This is the second installment to the RabbitMQ for Windows series. In our first installment, we walked through getting RabbitMQ installed on a Microsoft Windows machine. In this installment, we'll discuss a few high-level concepts and walk through creating our first RabbitMQ application.

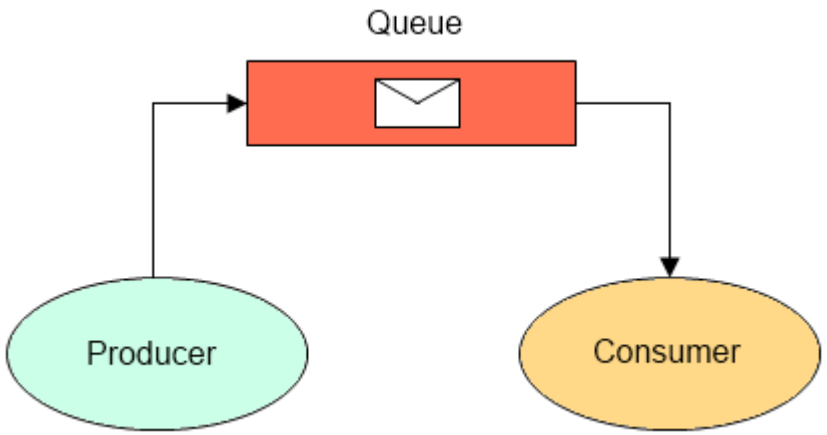
Basic Concepts

To being, let's discuss a few basic concepts. Each of the examples we'll be working through will have two roles represented: a *Producer* and a *Consumer*. A Producer sends messages and a Consumer receives messages.

Messages are basically any blob of bytes you'd like to send. This could be a simple ASCII string, JavaScript Object Notation (JSON), or a binary-serialized object.

Messages are sent to *Queues*. A Queue is a First-In-First-Out (FIFO) data structure. You can think of a queue as a sort of pipe where you put messages in one side of the pipe and the messages arrive at the other end of the pipe.

The following diagram depicts these concepts:



We'll introduce other concepts further into the series, but that's the basics. Let's move on to creating our first example.

Hello, World!

Our first application will be an obligatory “Hello World” example. We’ll create a Publisher application which sends the string “*Hello, World!*” to a RabbitMQ queue and a Consumer application which receives the message from the queue and displays it to the console.

For all of our examples, we’ll be using the official RabbitMQ .Net client available here. This library is also available via NuGet, so if you have the NuGet Package Manager installed you can retrieve it through the “Tools->Library Package Manager” menu item, or if you have the NuGet.exe command line utility then you can issue the following command in the directory you’d like it installed to:


```
nuget install RabbitMQ.Client
```

Create the Producer

To start, let’s create a new empty solution named HelloWorldExample (File->New->Project->Other Project Types->Visual Studio Solutions->Blank Solution). Once you have that created, add a new project of type “Console Application” to the solution and name it “Producer”.

Next, add a reference to the RabbitMQ.Client.dll assembly.

The first thing we’ll need to do for our producer is to establish a connection to the RabbitMQ server using a ConnectionFactory:

```
namespace Producer
{
  class Program
  {
```

```
    static void Main(string[] args)
    {
      var connectionFactory = new ConnectionFactory();
      IConnection connection = connectionFactory.CreateConnection();
    }
  }
}
```

The ConnectionFactory has a number of properties that can be set for our connection. In this example, we're establishing a connection using the default connection settings which assumes you have the RabbitMQ Windows service running on your local development machine. If you've installed it on a different machine then you'll need to set the Host property of the connectionFactory instance to the DNS name where you've installed RabbitMQ.

Next, we need to create a Channel:

```
IModel channel = connection.CreateModel();
```

A channel is a light-weight connection which RabbitMQ uses to enable multiple threads of communication over a single TCP/IP socket. Note that the actual type created is *RabbitMQ.Client.IModel*. In most RabbitMQ client libraries the term channel is used, but for some reason the authors of the .Net client library chose to use the term "Model". Descriptive, eh? We'll use the instance name of "channel" to be more descriptive.

Next, we need to create a queue:

```
channel.QueueDeclare("hello-world-queue", false, false, false, null);
```

This creates a queue on the server named "*hello-world-queue*" which is non-durable (won't survive a server restart), is non-exclusive (other channels can connect to the same queue), and is not auto-deleted once it's no longer being used. We'll discuss these parameters in more detail further in our series.

Next, we'll declare a byte array containing a UTF8-encoded array of bytes from the string "Hello, World!" and use the BasicPublish() method to publish the message to the queue:

```
byte[] message = Encoding.UTF8.GetBytes("Hello, World!");
channel.BasicPublish(string.Empty, “hello-world-queue”, null, message);
```

Again, don't worry about understanding the parameters just yet. We'll get to that soon enough.

Finally, we'll prompt the user to press a key to exit the application and close our channel and connection:

```
Console.WriteLine("Press any key to exit");
Console.ReadKey();
channel.Close();
connection.Close();
```

Here's the full Producer listing:

```
using System.Text;
using RabbitMQ.Client;

namespace Producer
{
  class Program
```

```
  {
    static void Main(string[] args)
    {
      var connectionFactory = new ConnectionFactory();
      IConnection connection = connectionFactory.CreateConnection();
      IModel channel = connection.CreateModel();
      channel.QueueDeclare("hello-world-queue", false, false, false, null);
      byte[] message = Encoding.UTF8.GetBytes("Hello, World!");
      channel.BasicPublish(string.Empty, "hello-world-queue", null, message);
      Console.WriteLine("Press any key to exit");
      Console.ReadKey();
      channel.Close();
      connection.Close();
    }
  }
}
```

Create the Consumer

Next, let's create our Consumer application. Add a new Console Application to the solution named "Consumer" and add a reference to the RabbitMQ.Client assembly. We'll start our consumer with the same connection, channel, and queue declarations:

```
var connectionFactory = new ConnectionFactory();
IConnection connection = connectionFactory.CreateConnection();
IModel channel = connection.CreateModel();
channel.QueueDeclare("hello-world-queue", false, false, false, null);
```

Next, we'll use the BasicGet() method to consume the message from the queue "hello-world-queue":

```
BasicGetResult result = channel.BasicGet("hello-world-queue", true);
```

Next, we'll check to ensure we received a result. If so, we'll convert the byte array contained within the Body property to a string and display it to the console:

```
if (result != null)
{
  string message = Encoding.UTF8.GetString(result.Body);
  Console.WriteLine(message);
}
```

Lastly, we'll prompt the user to press a key to exit the application and close our channel and connection:

```
Console.WriteLine("Press any key to exit");
Console.ReadKey();
channel.Close();
connection.Close();
```

Here's the full Consumer listing:

```
using System;
using System.Text;
using RabbitMQ.Client;
```

```
namespace Consumer
{
  class Program
  {
    static void Main(string[] args)
    {
      var connectionFactory = new ConnectionFactory();
      IConnection connection = connectionFactory.CreateConnection();
      IModel channel = connection.CreateModel();
      channel.QueueDeclare("hello-world-queue", false, false, false, null);
      BasicGetResult result = channel.BasicGet("hello-world-queue", true);
      if (result != null)
      {
        string message = Encoding.UTF8.GetString(result.Body);
        Console.WriteLine(message);
      }
      Console.WriteLine("Press any key to exit");
      Console.ReadKey();
      channel.Close();
      connection.Close();
    }
  }
}
```

To see the application in action, start the Publisher application first and then start the Consumer application. If all goes well, you should see the Consumer application print the following:

```
Hello, World!
Press any key to exit
```

Congratulations! You've just completed your first RabbitMQ application. Next time, we'll take a closer look at the concepts used within our Hello World example.

← RabbitMQ for Windows: Introduction

Dependency Management in .Net: Offline Dependencies with NuGet Command Line Tool →

Comments for this thread are now closed

3 Comments Los Techies Disqus' Privacy Policy 1 Login

Recommend 1 Tweet Share Sort by Best

Badarinath Katti • 7 years ago
Good one.. thanks :)
Share ›

Annie • 7 years ago
Thanks a lot.
Share ›

PeterNL • 8 years ago
Nice to follow....and it works! Firsttimer.
Share ›

Subscribe Add Disqus to your siteAdd DisqusAdd Do Not Sell My Data

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

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