Messaging through a service bus in .NET using MassTransit part 8: observing events in the bus

OCTOBER 6, 2016 1 COMMENT (HTTPS://DOTNETCODR.COM/2016/10/06/MESSAGING-THROUGH-A-SERVICE-BUS-IN-NET-USING-MASSTRANSIT-PART-8-OBSERVING-EVENTS-IN-THE-BUS/#COMMENTS)

Introduction

In the previous post (https://dotnetcodr.com/2016/10/04/messaging-through-a-service-bus-in-net-using-masstransit-part-7-intercepting-messages/) we learned how to intercept messages in MassTransit. We can intercept messages using various observer interfaces when sending, publishing, receiving and consuming messages. We cannot modify the message content so they are read-only operations. The implemented interfaces need to be registered with the bus so that we can do any extra work on the messages such as logging or debugging.

In this post we'll see how to register events that happen in the bus.

The bus observer

The IBusObserver interface is similar to the ones we saw in the previous post in that it has a couple of events that are triggered at various stages of the bus life cycle. All methods return a Task so they are prepared for the async-await paradigm.

The following BusObserver class in the MassTransit.Receiver project of the demo application shows a minimal implementation:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MassTransit.Receiver
{
    public class BusObserver : IBusObserver
    {
        private readonly string _logger = "[BusObserver]";

        public async Task CreateFaulted(Exception exception)
        {
            Console.WriteLine(string.Concat(_logger, ": Bus exception: ", exception.Message));
            await Task.FromResult(0);
        }

        public async Task PostCreate(IBus bus)
        {
            Console.WriteLine(string.Concat(_logger, ": Bus has been created with address ", bus.Addr
            await Task.FromResult(0);
        }

        public async Task PostStart(IBus bus, Task busReady)
        {
            Console.WriteLine(string.Concat(_logger, ": Bus has been started with address ", bus.Addr
            await busReady;
        }

        public async Task PostStop(IBus bus)
        {
            Console.WriteLine(string.Concat(_logger, ": Bus has been stopped with address ", bus.Addr
            await Task.FromResult(0);
        }

        public async Task PreStart(IBus bus)
        {
            Console.WriteLine(string.Concat(_logger, ": Bus is about to start with address ", bus.Add
            await Task.FromResult(0);
        }

        public async Task PreStop(IBus bus)
        {
            Console.WriteLine(string.Concat(_logger, ": Bus is about to stop with address ", bus.Addr
            await Task.FromResult(0);
        }

        public async Task StartFaulted(IBus bus, Exception exception)
        {
            Console.WriteLine(string.Concat(_logger, ": Bus exception at start-up: ", exception.Messa
                , " for bus ", bus.Address));
            await Task.FromResult(0);
        }

        public async Task StopFaulted(IBus bus, Exception exception)
        {
            Console.WriteLine(string.Concat(_logger, ": Bus exception at shut-down: ", exception.Mess
                , " for bus ", bus.Address));
            await Task.FromResult(0);
        }
    }
}
```

The method names reveal that we can monitor events related to creating, starting and stopping the bus. We can also observe exceptions. The IBus object offers most bus-related methods and functions we used for our IBusControl object such as Send, Publish, ConnectConsumer etc., so there’s nothing stopping us from sending an extra message to a queue from within the bus observer.

We can register the bus observer within the block where we create the IBusControl object. The RunMassTransitReceiverWithRabbit method in Program.cs of MassTransit.Receiver where we start and stop the bus can be extended with an extra line of code:

```
IBusControl rabbitBusControl = Bus.Factory.CreateUsingRabbitMq(rabbit =>
{
    IRabbitMqHost rabbitMqHost = rabbit.Host(new Uri("rabbitmq://localhost:5672/accounting"), setting
    {
        settings.Password("accountant");
        settings.Username("accountant");
    });

    rabbit.BusObserver(new BusObserver());
        //rest of code ignored
}
```

I've commented out all other observers we created in the previous post to easily identify the messages from the bus observer. We can additionally put a Console.ReadKey after the code that stops the bus so that we can see the messages related to stopping the bus:

```
rabbitBusControl.Start();
Console.ReadKey();

rabbitBusControl.Stop();
Console.ReadKey();
```

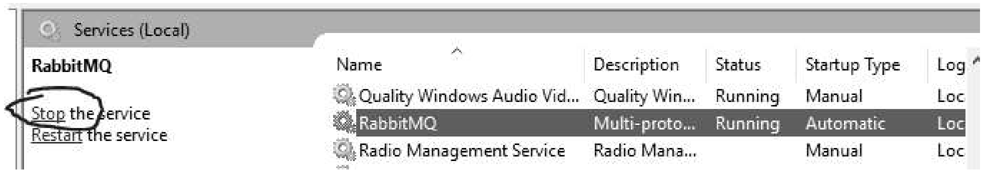
For the demo it is enough to start the receiver, we don't need to send messages here. There should be messages similar to the following after a successful bus start-up:

```
[BusObserver]: Bus has been created with address rabbitmq://localhost:5672/accounting/bus-ANDRASPC1-MassTransit.Receiver.vshost-4g7yyyg5o96d6puubdj658n3ra?durable=false&autodelete=true&prefetch=8
[BusObserver]: Bus is about to start with address rabbitmq://localhost:5672/accounting/bus-ANDRASPC1-MassTransit.Receiver.vshost-4g7yyyg5o96d6puubdj658n3ra?durable=false&autodelete=true&prefetch=8
[BusObserver]: Bus has been started with address rabbitmq://localhost:5672/accounting/bus-ANDRASPC1-MassTransit.Receiver.vshost-4g7yyyg5o96d6puubdj658n3ra?durable=false&autodelete=true&prefetch=8
```

Press a key so that the bus is stopped. We'll see some additional messages:

```
[BusObserver]: Bus is about to stop with address rabbitmq://localhost:5672/accounting/bus-ANDRASPC1-MassTransit.Receiver.vshost-4g7yyyg5o96d6puubdj658n3ra?durable=false&autodelete=true&prefetch=8
[BusObserver]: Bus has been stopped with address rabbitmq://localhost:5672/accounting/bus-ANDRASPC1-MassTransit.Receiver.vshost-4g7yyyg5o96d6puubdj658n3ra?durable=false&autodelete=true&prefetch=8
```

We can also simulate an exception where RabbitMq is unreachable. Open the Services application – search for "Services" using the standard Windows search function – and locate the service called RabbitMQ and press Stop:



Name	Description	Status	Startup Type	Log
Quality Windows Audio Vid...	Quality Win...	Running	Manual	Loc
RabbitMQ	Multi-proto...	Running	Automatic	Loc
Radio Management Service	Radio Mana...		Manual	Loc

(https://dotnetcodr.files.wordpress.com/2016/10/stop-the-rabbitmq-service-for-masstransit.png)

Start the Receiver now. You should see that the same RabbitMqConnectionException is thrown repeatedly with exception messages similar to the following:

```
[BusObserver]: Bus has been created with address rabbitmq://localhost:5672/accounting/bus-ANDRASPC1-MassTransit.Receiver.vshost-4g7yyyg5o96d97ucbdj658joby?durable=false&autodelete=true&prefetch=8
[BusObserver]: Bus is about to start with address rabbitmq://localhost:5672/accounting/bus-ANDRASPC1-MassTransit.Receiver.vshost-4g7yyyg5o96d97ucbdj658joby?durable=false&autodelete=true&prefetch=8
RabbitMQ connection failed: Connect failed: accountant@localhost:5672/accounting
[BusObserver]: Bus is about to stop with address rabbitmq://localhost:5672/accounting/bus-ANDRASPC1-MassTransit.Receiver.vshost-4g7yyyg5o96d97ucbdj658joby?durable=false&autodelete=true&prefetch=8
[BusObserver]: Bus has been stopped with address rabbitmq://localhost:5672/accounting/bus-ANDRASPC1-MassTransit.Receiver.vshost-4g7yyyg5o96d97ucbdj658joby?durable=false&autodelete=true&prefetch=8
```

```
[BusObserver]: Bus exception at start-up: Connect failed: accountant@localhost:5672/accounting for bus
rabbitmq://localhost:5672/accounting/bus-ANDRASPC1-MassTransit.Receiver.vshost-4g7yyyg5o96d97ucbdj658joby?
durable=false&autodelete=true&prefetch=8
RabbitMQ connection failed: Connect failed: accountant@localhost:5672/accounting
[BusObserver]: Bus is about to start with address rabbitmq://localhost:5672/accounting/bus-ANDRASPC1-
MassTransit.Receiver.vshost-4g7yyyg5o96d97ucbdj658joby?durable=false&autodelete=true&prefetch=8
RabbitMQ receive transport failed: The supervisor is stopping, no additional scopes can be created
RabbitMQ receive transport failed: The supervisor is stopping, no additional scopes can be created
RabbitMQ receive transport failed: The supervisor is stopping, no additional scopes can be created
RabbitMQ receive transport failed: The supervisor is stopping, no additional scopes can be created
[BusObserver]: Bus is about to stop with address rabbitmq://localhost:5672/accounting/bus-ANDRASPC1-
MassTransit.Receiver.vshost-4g7yyyg5o96d97ucbdj658joby?durable=false&autodelete=true&prefetch=8
[BusObserver]: Bus has been stopped with address rabbitmq://localhost:5672/accounting/bus-ANDRASPC1-
MassTransit.Receiver.vshost-4g7yyyg5o96d97ucbdj658joby?durable=false&autodelete=true&prefetch=8
[BusObserver]: Bus exception at start-up: The supervisor is stopping, no additional scopes can be created for bus
rabbitmq://localhost:5672/accounting/bus-ANDRASPC1-MassTransit.Receiver.vshost-4g7yyyg5o96d97ucbdj658joby?
durable=false&autodelete=true&prefetch=8
RabbitMQ receive transport failed: The supervisor is stopping, no additional scopes can be created
```

The above test also reveals how MassTransit doesn’t just give up if RabbitMq is not reachable. Instead, it will try to connect to the provided RabbitMq endpoint repeatedly.

In fact we can perform at least one more test. Restart the RabbitMQ service in the Services window and start the Receiver project. Let the bus be created and started. Then stop the RabbitMQ service again. You’ll see that the bus will start complaining:

```
[BusObserver]: Bus has been started with address rabbitmq://localhost:5672/accounting/bus-ANDRASPC1-
MassTransit.Receiver.vshost-4g7yyyg5o96d96b8bdj658qfgw?durable=false&autodelete=true&prefetch=8
Timeout waiting for consumer to exit: rabbitmq://localhost:5672/accounting/mycompany.domains.queues?prefetch=8
RabbitMQ connection failed: Connect failed: accountant@localhost:5672/accounting
RabbitMQ connection failed: Connect failed: accountant@localhost:5672/accounting
RabbitMQ connection failed: Connect failed: accountant@localhost:5672/accounting
RabbitMQ connection failed: Connect failed: accountant@localhost:5672/accounting
.
.
.
```

Start the RabbitMQ service again. After a couple of seconds the MassTransit bus will be up again and the “connection failed” messages will stop appearing. You can now start the Publisher and you’ll see that the message was consumed by the Consumer like before, i.e. a gap in the RabbitMq operation wasn’t fatal to MassTransit.

We’ll continue in the next post.

View the list of posts on Messaging here (https://dotnetcodr.com/messaging/).

FILED UNDER .NET, MESSAGING TAGGED WITH C#, MASSTRANSIT, MESSAGING, RABBITMQ

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

One Response to *Messaging through a service bus in .NET using MassTransit part 8: observing events in the bus*

Peyman says:
April 12, 2021 at 3:59 pm
Thanks for useful contents.
Would you please provide some content on masstransit Saga and RoutingSlip?
Thanks in Advance.

Reply