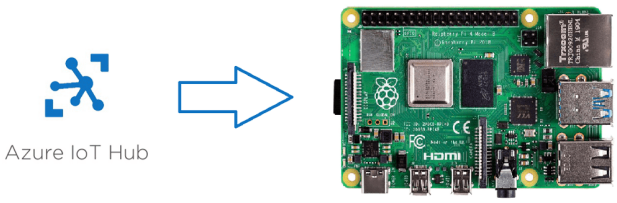
{ Pete Codes }

{Blog}



Explorations in Dot Net Core 3.0 for Raspberry Pi - Part 4

Search …

Recent Talks

Build a Robot Arm with .NET 5, a Raspberry Pi, Blazor and SignalR – A Workshop

A Short History of IoT

IoT Lunch and Learn – Online

DevOps and IoT – Webcast

A tour of Azure IoT – Webcast

Upcoming Presentations

Festive Tech Calendar - December 2020

Past Presentations

Brighton Web Development Meetup - 29th October 2020

HackSoc - 13th October 2020

Techfast - 27th August 2020

September 28, 2019 PeteCodes

Leave a comment

This is part 4 of a (at least) 5 part blog series on Dot Net Core on the Raspberry Pi....

You can see Part 1 – Installation and Hello World – right here...
You can see Part 2 – GPIO – right here...
You can see Part 3 – Sending Azure IoT Hub Messages – right here...
You can see Part 5 – Remote Deployment and Debugging – right here...

On September 25th 2019, the Dot Net Team released version 3.0 of the Dot Net Core Framework at .Net Conf.

To join in the fun, I held a special with Notts IoT, the IoT group I organise in

Virtual Azure Community Day - 28th July 2020
NeXt Generation - 13th May 2020
MS How To - 5th May 2020
MS How To - 29th April 2020
Azure Bootamp Virtual - 23rd April 2020
Dot Net York - 5th March 2020
Dot Net Sheff - 3rd March 2020
DDD North - 29th February 2020
DevOps Notts - 25th February 2020
Dot Net Oxford - 21st January 2020
AI Bootcamp - Nottingham - 14th December 2019
DDD Reading - 12th October 2019
HackSoc Nottingham - 10th October 2019
LATi Bar - 3rd October 2019
Code Club and STEM Ambassador Meetup - 2nd October 2019
.net Liverpool - 26th September 2019
Notts IoT - 25th September 2019
Digital Lincoln - 30th July 2019
Tech Nottingham - 13th May 2019
Global Azure Bootcamp Nottingham - 27-04-19
Notts IoT - 25th April 2019

Nottingham, where I gave a talk on Dot Net Core 3.0 on the Raspberry Pi.

This blog post is what I've learnt along the way to preparing for the talk as well as afterwards...!

Previously on Pete Codes...

In the last blog post we'd created a Console app that read the status of a button and sent a message to an Azure IoT Hub.

In this post we'll add code to allow us to receive a message back from our IoT Hub that we send from the Microsoft IoT Device Explorer.

What you'll need

- You'll need to have completed the steps in the previous blog post – Read that here
- You'll need an Azure subscription – Grab a trial here

IoT Leeds - 18th March 2019
DDD North - 2nd March 2019
Lancashire Tech Talks - 28th February 2019
Tech Nottingham - 11th February 2019
Notts IoT - 24th January 2019
Derbyshire Dot Net – 29th November 2018
PHPem Unconference - 24th November 2018
Lincoln Hack - 11-11-18
Beeston Raspberry Jam - 20th October 2018
Notts IoT Lightning Talks - 18th October 2018
Notts Dev Workshop - 04-09-18
Tech On Toast - 22nd August 2018
Code Club Meetup - 09-07-18
Notts IoT - 21st June 2018
Notts IoT - 19th April 2018
Notts IoT - 15th March 2018
Notts Dev Workshop - 6th February 2018

Recent Posts

Install and use Microsoft Dot NET 5 with the Raspberry Pi

An ESP8266 based, Morse Code Decoding Telegraph

Short History of the Telegraph with DIY Science Experiments

- You'll need the Azure IoT Device Explorer – Get it here

Putting even more "I" in IoT

If you've followed along with the previous blog, you'll already have installed the Dot Net Core 3.0 binaries and runtimes, and have a console app running that flashes an LED and sends a message to an IoT Hub when you press a button.

Our next step is to write some code which receives messages from the IoT Hub. We can send these messages through the hub using our Azure Device Explorer

Listen up friend

Let's add a subroutine and some code that will listen for messages from the IoT Hub.

Create the following subroutine beneath our existing

Recent Comments

My Web App Journey – Taming the fire | Jonathan Tweedle on Explorations in Dot Net Core 3 for Raspberry Pi – Part 1 – Installation and Hello World

Install DotNet Core 3.1.0 onto Raspberry PI – TheCoreCoder on Explorations in Dot Net Core 3 for Raspberry Pi – Part 1 – Installation and Hello World

Install your first Hello World .NET Core 3.0 app on Raspberry – TheCoreCoder on Explorations in Dot Net Core 3 for Raspberry Pi – Part 1 – Installation and Hello World

Categories

Blog

Uncategorized

Meta

Log in

Entries feed

Comments feed

WordPress.org

SendDeviceToCloudMessageAsync subroutine;

```
private static async void ReceiveCloudTo
{

}
```

We'll add a Console.WriteLine first so that we know our sub has been called successfully;

```
Console.WriteLine("Receiving Cloud to De
```

Next we'll sit in a loop waiting for a message to be received from the IoT Hub... This is safe to do because we're using an Asynchronous Subroutine. This way the rest of the code will continue. Add the following while loop to the subroutine;

```
while (true)
{
}
```

Next we need to try receiving a Message from the IoT Hub... We do this by calling

the Device Client ReceiveAsync method. Add the following to the while loop;

```
Message receivedMessage = await deviceCl
```

If we don't actually receive any messages from the IoT Hub, the above call will return a Null. So we need to check for this before doing anything else. Add the following below the ReceiveAsync call;

```
if (receivedMessage != null)
{
}
```

Next, assuming that we've received a valid message from the IoT Hub, we can decode that message in a similair way to how we encoded the message string when we were sending a message to the IoT Hub... Add the following within the if statement which will convert the IoT Hub message into an ASCII String;

```
string receivedMessageString = Encoding.
```

We can now spit our message out to the console so we know we've received it;

```
Console.WriteLine("Received message: {0}
```

Next we need to tell the IoT Hub that we've received the message successfully. We do this by calling the Device Client CompleteAsync method on our Device Client;

```
await deviceClient.CompleteAsync(receive
```

When the IoT Hub receives the Complete Acknowledgement it sets the message to completed, which in effect deletes the message we received from the Cloud to Device Queue. You can read more about how the Cloud to Device Queue on the Microsoft Docs Website.

The finished subroutine should now look like;

```
private static async void ReceiveCloudTo
{
    Console.WriteLine("Receiving Cloud t

    while (true)
    {
        Message receivedMessage = await

        if (receivedMessage != null)
```

```
        {
            string receivedMessageString
            Console.WriteLine("Received
            await deviceClient.CompleteA
        }

    }
}
```

Finally we need to kick our subroutine off. After our initial Hello World Console WriteLine we can add a call to the subroutine;

```
ReceiveCloudToDeviceMessageAsync();
```

The finished code should now look like;

```
using System;
using System.Device.Gpio;
using Microsoft.Azure.Devices.Client;
using System.Text;

namespace rpitest
{
    class Program
    {

        private static readonly string c
        private static DeviceClient devi

        static void Main(string[] args)
        {
            Console.WriteLine("Hello Wor
```

```
            deviceClient = DeviceClient.

            ReceiveCloudToDeviceMessageA

            GpioController controller =
            var pin = 10;
            var buttonPin = 26;
            var buttonPressed = false;

            controller.OpenPin(pin, PinM
            controller.OpenPin(buttonPir

            try
            {
                while (true)
                {
                    if (controller.Read(
                    {
                        controller.Write

                        if (buttonPresse
                        {
                            buttonPresse
                            SendDeviceTo
                        }
                    }
                    else
                    {
                        controller.Write
                        buttonPressed =
                    }
                }
            }
            finally
            {
                controller.ClosePin(pin)
            }
        }

        private static async void SendDe
```

```
        {
            var messageString = "Button
            Message message = new Messag

            message.Properties.Add("butt

            await deviceClient.SendEvent
            Console.WriteLine("Sending M

        }

        private static async void Receiv
        {
            Console.WriteLine("Receiving

            while (true)
            {
                Message receivedMessage

                if (receivedMessage != r
                {
                    string receivedMessa
                    Console.WriteLine("R
                    await deviceClient.C
                }

            }
        }
    }
}
```

We can now run the latest version of the application;

```
dotnet run
```

Console App waiting for Cloud to Device Message

Return to the Device Explorer

If we now open the Device Explorer Application again and click the "Messages to Device" tab;

Device Explorer "Message to Device" tab

Make sure that the IoT Hub and Device ID dropdowns have got the right

information in them.

Now enter a message into the “Message” box and press the “Send” Button;

Sent Cloud Message to Device

You should then see our Message appearing in the console of our app;

Cloud to Device Message received

Next Time

In the next post we'll switch to using our main Development PC to code on, remotely deploying and debugging code direct from VS Code.

< Part 3 – Azure IoT

Part 5 – Remote Deployment >

Posted in: Blog
Filed under: Dot Net Core, Electronics, IoT, Raspberry Pi

← Explorations in Dot Net Core 3 for Raspberry Pi – Part 3 – Azure IoT Hub

Azure DevOps with Azure IoT Edge and the Raspberry Pi – Part 1 – Setting up Raspbian Buster →

Leave a Reply

You must be logged in to post a comment.

MICROSOFT AZURE MVP

STEM AMBASSADOR

PLURALSIGHT AUTHOR