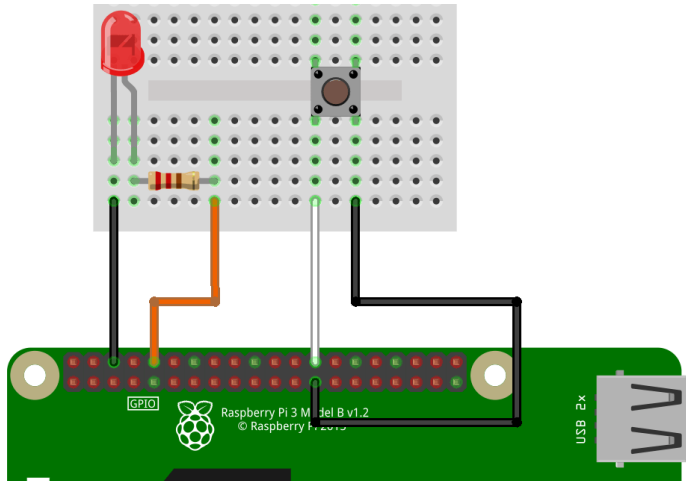


{ Pete Codes }



{Blog}



Explorations in Dot Net Core 3 for Raspberry Pi -

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

Part 2 - Controlling GPIO

🕒 September 7, 2019 👤 PeteCodes 💬

Leave a comment

*This is part 2 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 3 – Sending
Azure IoT Hub Messages – right
here...*

*You can see Part 4 – Receiving
Azure IoT Hub Messages – right
here...*

*You can see Part 5 – Remote
Deployment and Debugging –
right here...*

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 Bootcamp 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

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 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...!

Previously on Pete Codes...

In the last blog post we got our Pi all setup with the release version of Dot Net Core 3.0. We then scaffolded, built and ran an ASP.Net Core Blazor Web App.

In this post, we'll look at getting a simple Console App up and running, and using the IoT GPIO library, toggle an LED connected to our Pi on and off and read a push button.

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

Thanks Scott Hansleman!

Before I go any further, I need to thank Scott Hansleman for his excellent article which helped me get started writing this post!

What you'll need

- You'll need to have completed the steps in the previous blog post – [Read that here](#)
- You'll need a bread board, a 10k resistor, an LED and a push button – The easiest thing to do is buy one of these packs
- You'll then need to assemble the circuit I detail in the sections a bit further down

Hello (*Console*) World

If you've followed along with the previous blog, you'll already have

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](#)

installed the Dot Net Core 3.0 binaries and runtimes.

At this stage, we want to create a new Dot Net Core 3 Console Application.

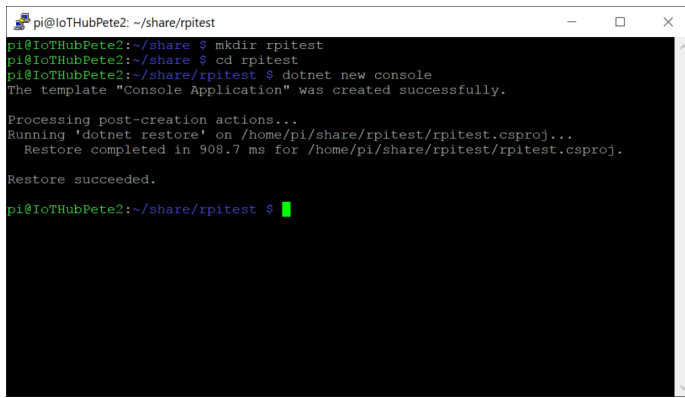
Back in your terminal window, create a new directory in the “**share**” folder called “**rpitest**”;

```
mkdir rpitest  
cd rpitest
```

Next we can scaffold a basic console application;

```
dotnet new console
```

This will take a few seconds to complete... You’ll notice that it’s considerably faster than scaffolding a new Blazor app... Mainly because there are far less files involved in a console app, which only really relies on the already installed binaries.

A terminal window titled 'pi@ioTHubPete2: ~/share/rptest' showing the following commands and output:

```
pi@ioTHubPete2:~/share $ mkdir rptest
pi@ioTHubPete2:~/share $ cd rptest
pi@ioTHubPete2:~/share/rptest $ dotnet new console
The template "Console Application" was created successfully.

Processing post-creation actions...
Running 'dotnet restore' on /home/pi/share/rptest/rptest.csproj...
Restore completed in 908.7 ms for /home/pi/share/rptest/rptest.csproj.
Restore succeeded.

pi@ioTHubPete2:~/share/rptest $
```

Dotnet New Console Command

If you do a run a “**dir**” command, you’ll see there’s an “**obj**” folder which contains “**nuget**” references primarily, along with our project file and program.cs. So not much to see so far.

Dot Net Core Console Files

Bob the Builder

Let’s see if our vanilla project compiles by running;

```
dotnet build
```

Just like in the Blazor alternative, this may take some time to complete on a Pi. On mine, this took roughly 51 seconds... Less than the Blazor app of course.

Dot Net Build Command

Now we're ready to run our really basic Dot Net Core 3 Console App;

```
dotnet run
```

This may take a second or two to complete, but you should see "Hello World!" printed in the terminal.

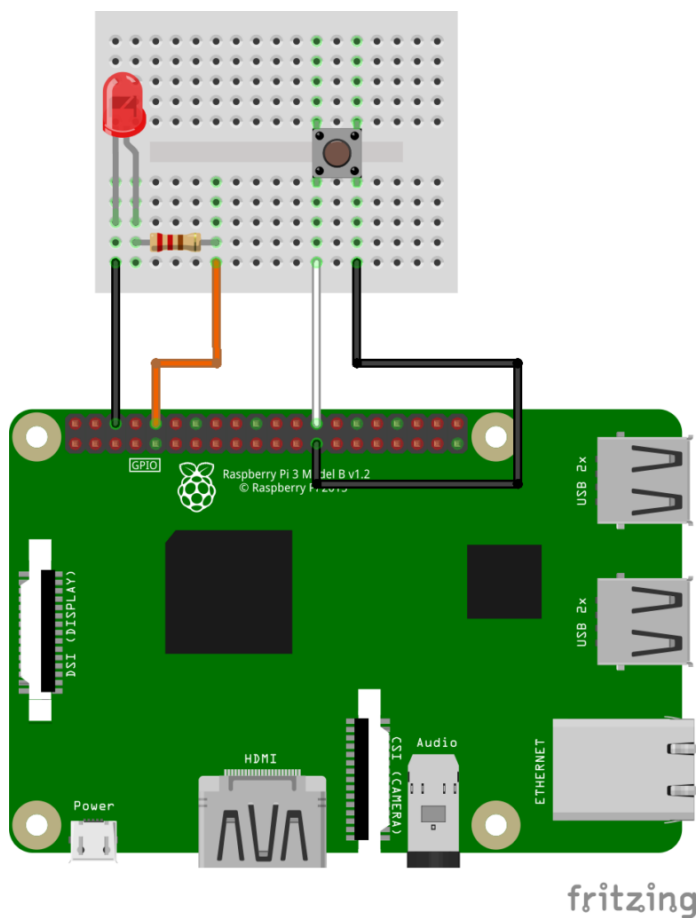
Dot Net Run Command

Whoop! That was pretty easy eh...

Light me up!

No self respecting IoT project can possibly get any further now without lighting an LED up of course.

The first thing you're going to need to do is build up your basic circuit.



Once you've got that built up, we need to add a few bits to our vanilla Console App code.

Go ahead and open your project in VS Code. Again, on a Windows machine, you can do this by browsing to the folder in your mapped drive, finding some blank space, right clicking and hitting “Open with Code”;

Open with Code

Nuggets of Nuget

First up, let's add a reference to the System.Device.IoT Nuget package. The IoT Package doesn't come as standard with the Dot Net Core Binaries, and so we need to add this separately.

Now that the System.Device.GPIO Nuget Package is in full release we can simply

add the GPIO Nuget package to our project using the following command;

```
dotnet add package System.Device.Gpio
```

Add GPIO Nuget Package

Get it flashing!

With the relevant references now added, we can start writing some code to control our LED.

Open up the “*program.cs*” file

Dot Net Console program.cs file

Add the following two lines beneath the “using System;” line;

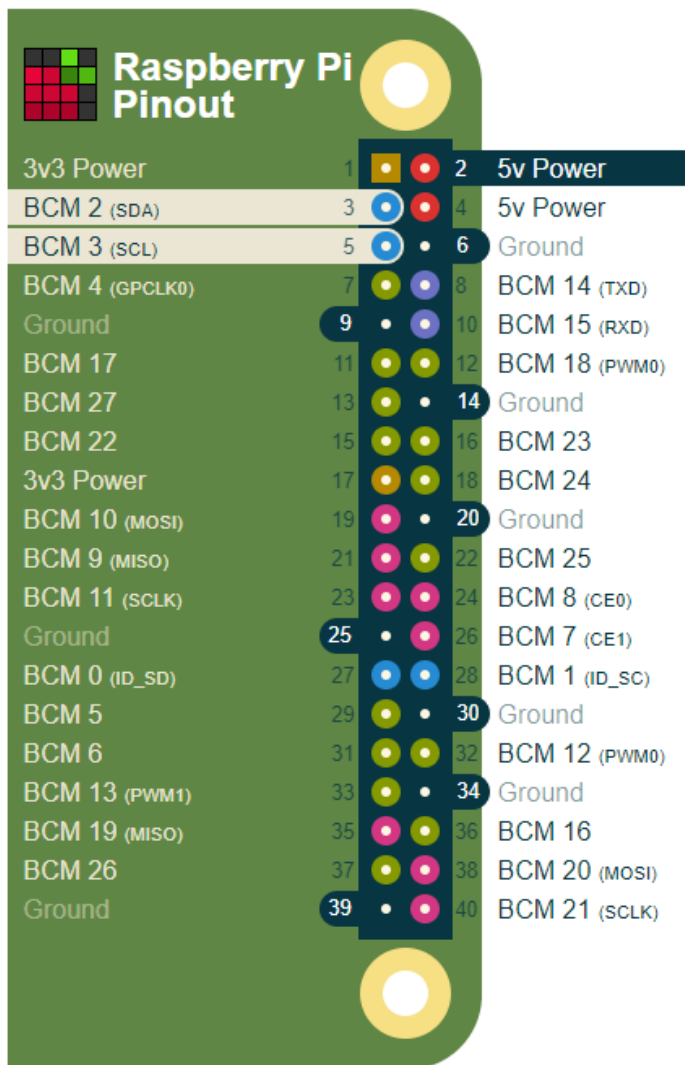
```
using System.Device.Gpio;  
using System.Threading;
```

We’ll be using the System.Threading library to create a small delay shortly.

Next we need to create a new GPIOController object;

```
GpioController controller = new GpioCont
```

Here we’re going to be using the Board Pin Numbering Scheme. The Raspberry Pi has the following pin outs;



Raspberry Pi Pinout
(Courtesy of raspberrypi.org)

Referring back to our circuit, we have our LED connected between Ground (Pin 6) and Pin 10.

So, let's create a pin variable for our LED pin. Add the following below the

“Console.WriteLine(“Hello World!”)” line;

```
var pin = 10;
```

Next we'll create a variable to hold our delay time;

```
var lightTime = 300;
```

Now we need to open our Controller Pin;

```
controller.OpenPin(pin, PinMode.Output);
```

Now we add a try-finally block just in case we have any issues with our code;

```
try
{

}
finally
{

}
```

In the try section we first add a “**while**” loop so we can loop indefinitely flashing our LED;

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

Now, in the “**while**” loop we can add the code to turn our LED pin on and off;

```
controller.Write(pin, PinValue.High);  
Thread.Sleep(lightTime);  
controller.Write(pin, PinValue.Low);  
Thread.Sleep(lightTime);
```

What we’ve done here is write a “High” value to our LED pin via the GPIO Controller. We then wait for our lightTime, which we set to 300, which is a figure in milliseconds.

After the delay we then write a “Low” value back to the LED and wait again...

The “while” loop ensures that this then continues ad-infinitum

Finally (pun intended), we can add some code to make sure that we release control of the LED pin when we exit the program.

Add the following line to the “finally” section;

```
controller.ClosePin(pin);
```

Save your file and your finished code
should look something like;

Dot Net Core Console App Finished

```
using System;
using System.Device.Gpio;
using System.Threading;

namespace rpitest
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello Wor

            GpioController controller =

            var pin = 10;
            var lightTime = 300;

            controller.OpenPin(pin, PinM

            try
            {
                while(true)
```

```
        {  
            controller.Write(pin);  
            Thread.Sleep(lightTime);  
            controller.Write(pin);  
            Thread.Sleep(lightTime);  
        }  
    }  
    finally  
    {  
        controller.ClosePin(pin);  
    }  
}  
}  
}
```

Flashy!

Our code's now finished! So let's return to the Terminal and rebuild our code with;

```
dotnet build
```

All being well, you should have a successfully built console app;

LED Console App Built

Go ahead and run the app with;

```
dotnet run
```

Flashing LED

Push the Button

Returning to VS Code, we can now move on to reading our connected button...

Firstly we need to add a variable for our button pin. So add the following after the “***var pin = 10;***” line;

```
var buttonPin = 26;
```

Next we need to set our button input up.

Add the following line after the ”

controller.OpenPin(pin,
PinMode.Output);” line;

```
controller.OpenPin(buttonPin, PinMode.In
```

Here, we're setting the Button Pin to be an input and we're pulling that pin high. We've got the other side of our button connected to 0v... So when we press the button, it'll go from being High to being Low

We can now remove the following lines of code;

```
var lightTime = 300;
```

and...

```
controller.Write(pin, PinValue.High);  
Thread.Sleep(lightTime);  
controller.Write(pin, PinValue.Low);  
Thread.Sleep(lightTime);
```

We now replace the above lines of code with the following;

```
if (controller.Read(buttonPin) == false)  
{  
    controller.Write(pin, PinValue.High)  
}  
else  
{  
    controller.Write(pin, PinValue.Low);  
}
```

Your finished code should now look
something like;

Finished Button Code

```
using System;
using System.Device.Gpio;

namespace rpitest
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello Wor

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

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

            try
            {
                while (true)
                {
```

```
        if (controller.Read()  
        {  
            controller.Write  
        }  
        else  
        {  
            controller.Write  
        }  
    }  
}  
finally  
{  
    controller.ClosePin(pin)  
}  
}  
}
```

Save your file and return to the terminal.

Running the following two lines will
build and start your code;

```
dotnet build  
dotnet run
```

If everything has worked, you should be
able to press the button and see your LED
light up.

Button Code Running

And with that, you've now got code that can set outputs and read inputs running in a Dot Net Core app on a Raspberry Pi.

Thanks for reading!

Next Time

In the next post we'll hook our code up to Azure's IoT Hub and send messages from the Raspberry Pi to Azure.

< Part 1 – Installation and Hello World

Part 3 – Receiving IoT Hub Messages

>

Posted in: Blog

Filed under: Dot Net Core, Electronics, IoT, Raspberry Pi

← Explorations in Dot Net Core 3 for Raspberry Pi – Part 1 – Installation and Hello World Explorations in Dot Net Core 3 for Raspberry Pi – Part 3 – Azure IoT Hub →

Leave a Reply

You must be logged in to post a comment.

MICROSOFT AZURE MVP

STEM AMBASSADOR

PLURALSIGHT AUTHOR

Copyright © 2020 PJG Creations Ltd