Windows 10 IoT Core on Raspberry Pi 2 – Blinky

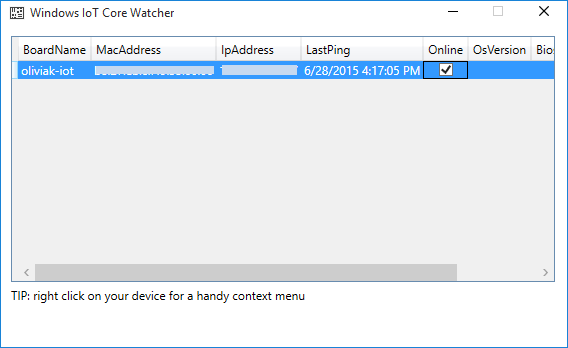
We’ll create the typical Hello World demo in the maker scene: Hello Blinky. It’s a simple LED blinking app; optionally you can connect a LED to your Windows 10 IoT Core device.

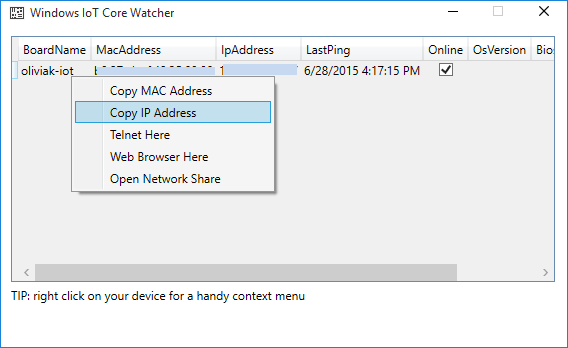
This is a headless sample. To better understand what headless mode is, go [here](http://ms-iot.github.io/content/en-US/win10/HeadlessMode.htm).

Also, be aware that the GPIO APIs are only available on Windows 10 IoT Core, so this sample cannot run on your desktop.

# 1. Find the IP address of your Raspberry Pi 2

The [Windows Developer Program for IoT](https://connect.microsoft.com/windowsembeddedIoT/Downloads/) includes some very resourceful programs, such as the **Windows IoT Core Watcher**. It is very useful to find out the IP address of the Raspberry Pi 2 when deploying a universal app remotely to it.





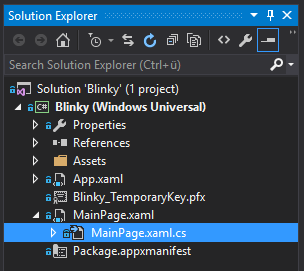
Alternatively, one can use the [Advanced IP Scanner](http://www.advanced-ip-scanner.com/).

# 2. Load the Project in Visual Studio

You can find the source code for this sample by navigating to the GitHub folder code-samples\Blinky. The sample code is available in either C++ or C#; however, the documentation here only details the C# variant. Make a copy of the folder on your disk and open the project from Visual Studio.

# 3. Blinky on your Raspberry Pi 2

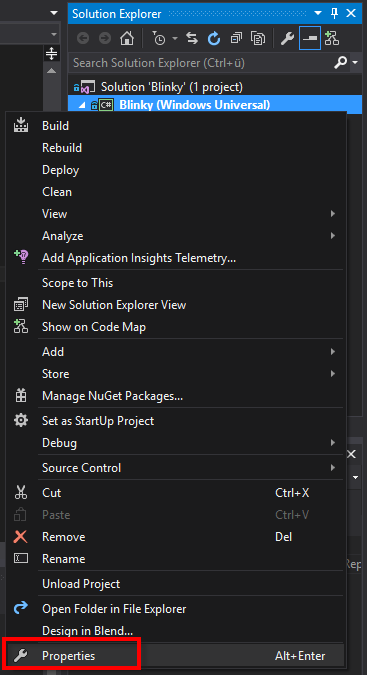
Once, the solution Blinky is opened, navigate to MainPage.xaml.cs (open the MainPage.xaml folder) in the Solution Explorer and click on it. This is the file you are going to make a change in:



Scroll down a little to change the LED\_PIN from 5 to 47. Why? GPIO 47 is the onboard ACT status LED on the Raspberry Pi 2. In such a way, we don't have to bother about building a circuit on a breadboard with a resistor and a LED just yet:

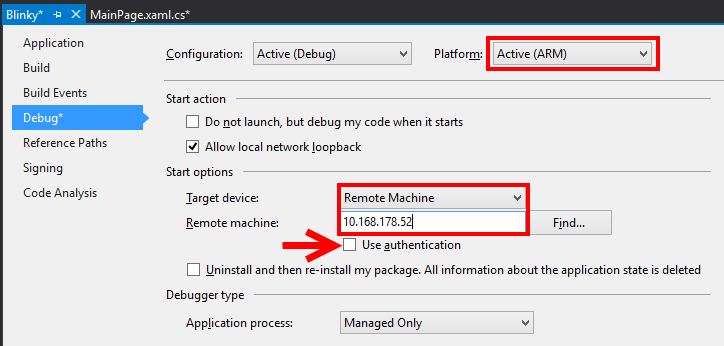


Now we need to configure the solution such that it will be deployed to a remote machine like our Raspberry Pi 2. Blinky will be deployed over the net by specifying the IP address of the Pi. Thus, right click on **Blinky (Windows Universal)** and then go to Properties:

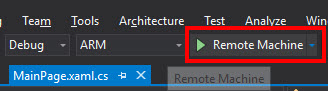


Under the Debug tab, change the **target device** to be a **remote machine**. Find out the IP address of your Raspberry Pi 2 using the Windows IoT Core Watcher or the Advanced IP Scanner.

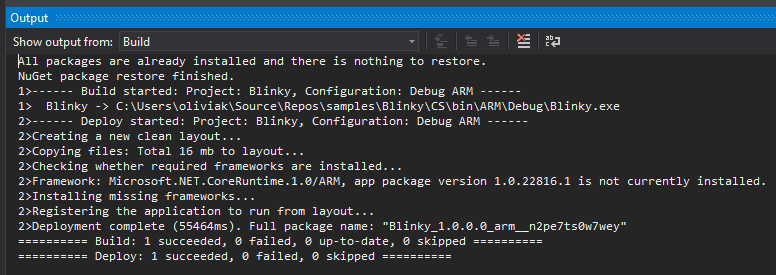
Make sure that the **platform** Blinky will be deployed on is set to **ARM**, and that **Use authentication** is **unchecked**.



You are good to go to run Blinky on your Raspberry Pi 2 - just click on the green play button!

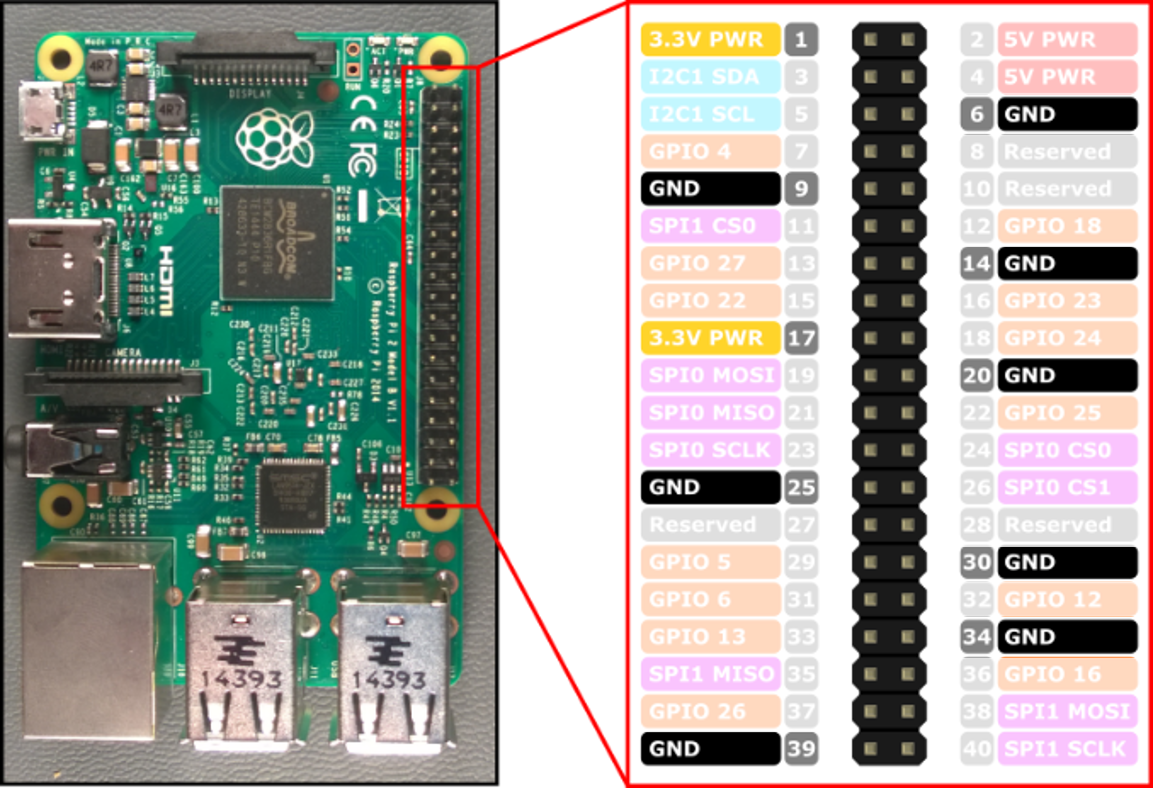


The output in Visual Studio as Blinky is deployed to the RP2 will be as follows:

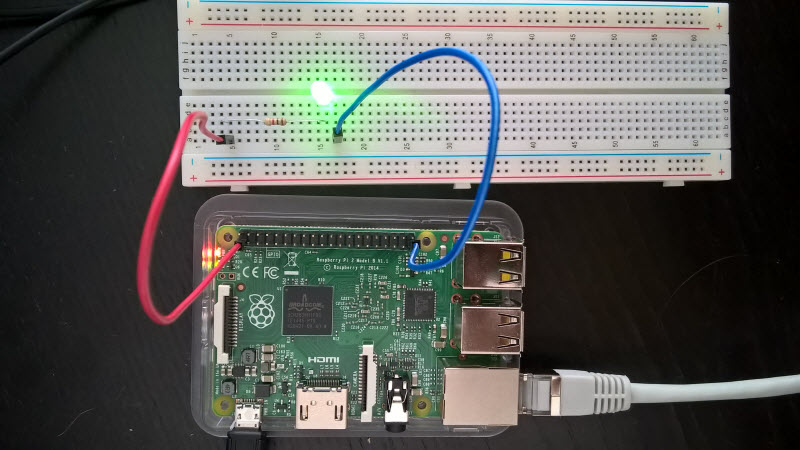


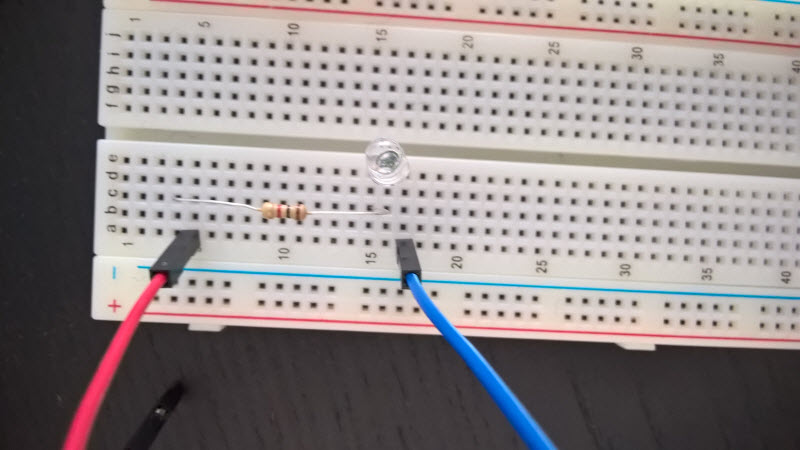
# 4. [Optional] Build the circuit

Before deploying Blinky again on your RP2 (now with a blinking LED), let's first build a working circuit. For checking the circuit, one only needs to connect the circuit with two GPIO pins of the RP2: power and ground.



You can see the power pins in yellow (**3.3V PWR**), i.e. pins 1 and 17, and the ground pins coloured in black (**GND**), i.e. pins 6, 9, 14, 25, 30, 34 and 39).



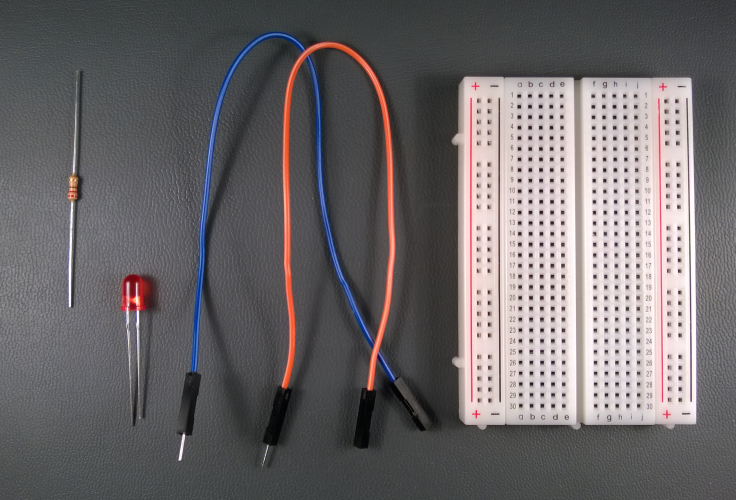


It is very similarly built as described below in the [next section](http://oliviaklose.com/hello-blinky/#blinkyled), with the difference that the blue connector wire is connected to pin 39 (ground) as opposed to pin 29 (GPIO 5).

# 5. [Optional] Blinky on the Raspberry Pi 2 + Breadboard

You’ll need a few components:

* a LED (whichever colour you like)
* a 220 Ω resistor
* a breadboard and a couple of connector wires



We will connect the one end of the LED to GPIO 5 (pin 29 on the expansion header) on the RPi2, the other end to the resistor, and the resistor to the 3.3 volt power supply from the RPi2. Note that the polarity of the LED is important. Make sure the shorter leg (-) is connected to GPIO 5 and the longer leg (+) to the resistor or it won’t light up.

And here is the pinout of the RPi2:

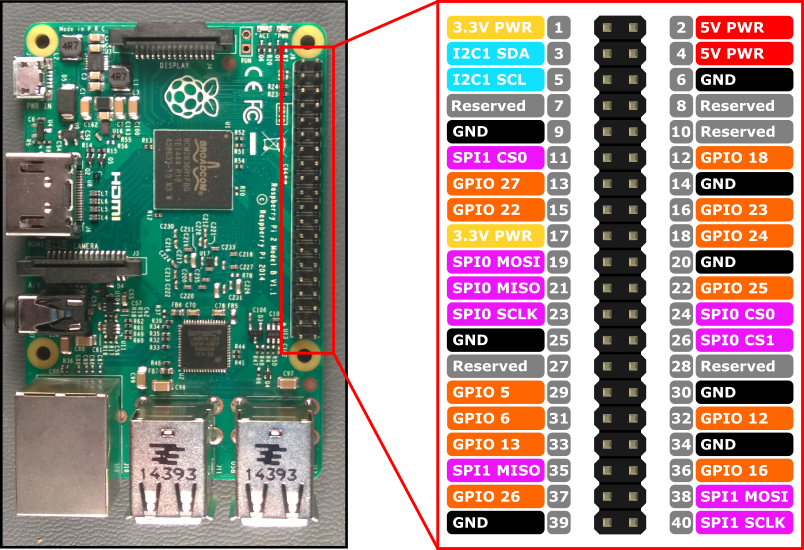


Image made with [*Fritzing*](http://fritzing.org/)

Here is an example of what your breadboard might look like with the circuit assembled:

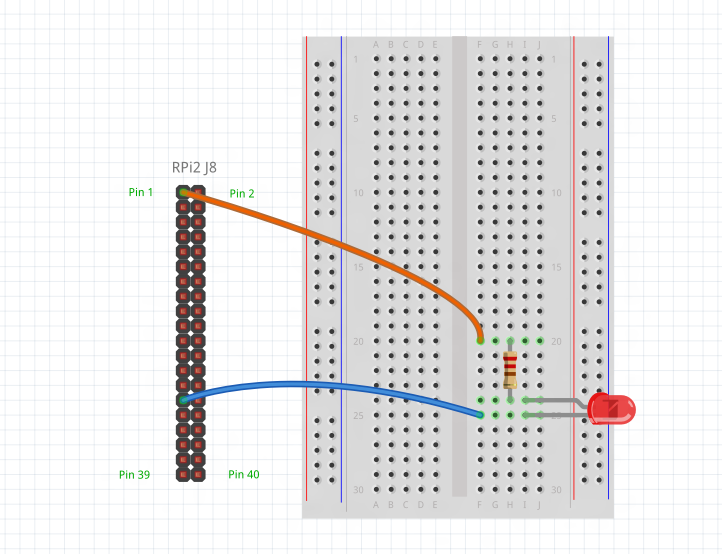
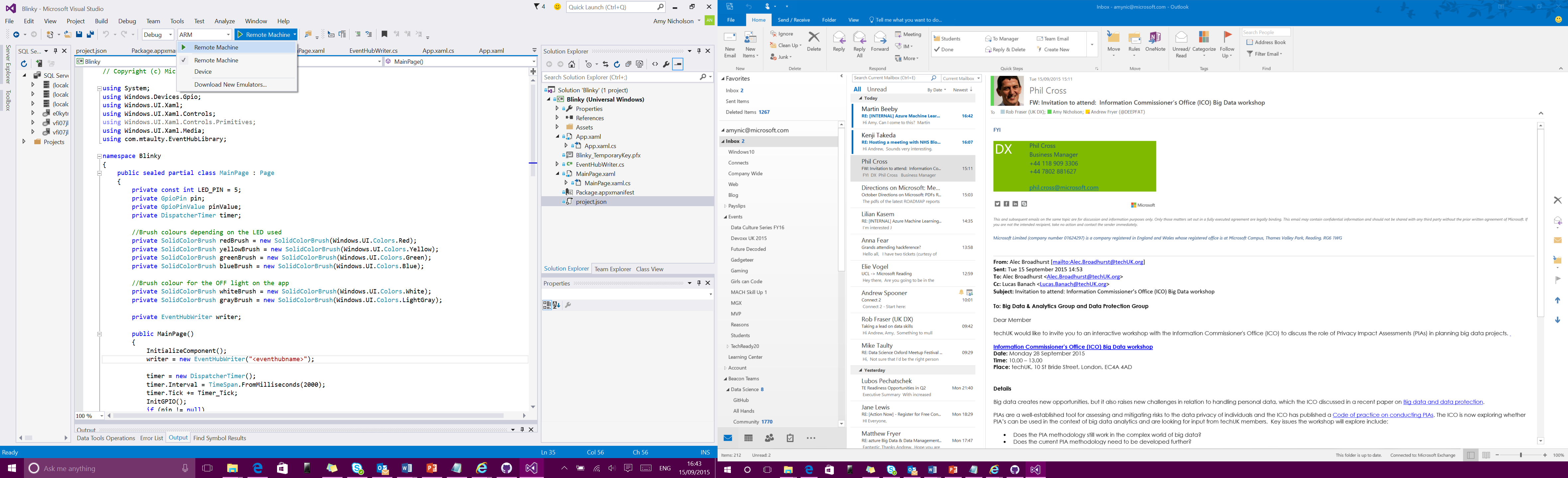


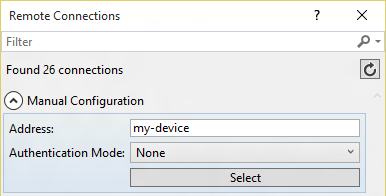
Image made with [*Fritzing*](http://fritzing.org/)

# 6. Deploy your app

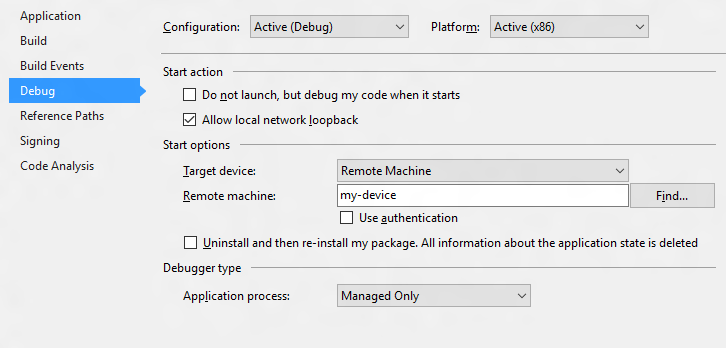
1. With the application open in Visual Studio, set the architecture in the toolbar dropdown. You’re building for Raspberry Pi 2, so select ARM.
2. Next, in the Visual Studio toolbar, click on the Local Machine dropdown and select Remote Machine



1. At this point, Visual Studio will present the **Remote Connections** dialog. If your device has a unique name (see IoT Core Watcher), you can enter it here (in this example, we’re using **my-device**). Otherwise, use the IP address of your Windows IoT Core device. After entering the device name/IP select None for Windows Authentication, then click **Select**.



1. You can verify or modify these values by navigating to the project properties (select **Properties** in the Solution Explorer) and choosing the Debug tab on the left:



When everything is set up, you should be able to press F5 from Visual Studio. The Blinky app will deploy and start on the Windows IoT device, if you have a screen you should see the LED blink in sync with the simulation on the screen.



Congratulations! You controlled one of the GPIO pins on your Windows IoT device.

# 7. Let’s look at the code

The code for this sample is pretty simple. We use a timer, and each time the ‘Tick’ event is called, we flip the state of the LED.

## 7.1 Let’s look at the code

Here is how you set up the timer in C#:

public MainPage()

{

// ...

this.timer = new DispatcherTimer();

this.timer.Interval = TimeSpan.FromMilliseconds(500);

this.timer.Tick += Timer\_Tick;

this.timer.Start();

// ...

}

private void Timer\_Tick(object sender, object e)

{

FlipLED();

}

## 7.2 Initialise the GPIO pin

To drive the GPIO pin, first we need to initialize it. Here is the C# code (notice how we leverage the new WinRT classes in the Windows.Devices.Gpio namespace):

using Windows.Devices.Gpio;

private void InitGPIO()

{

var gpio = GpioController.GetDefault();

// Show an error if there is no GPIO controller

if (gpio == null)

{

pin = null;

GpioStatus.Text = "There is no GPIO controller on this device.";

return;

}

pin = gpio.OpenPin(LED\_PIN);

// Show an error if the pin wasn't initialized properly

if (pin == null)

{

GpioStatus.Text = "There were problems initializing the GPIO pin.";

return;

}

pin.Write(GpioPinValue.High);

pin.SetDriveMode(GpioPinDriveMode.Output);

GpioStatus.Text = "GPIO pin initialized correctly.";

}

Let’s break this down a little:

* First, we use GpioController.GetDefault() to get the GPIO controller.
* If the device does not have a GPIO controller, this function will return null.
* Then we attempt to open the pin by calling GpioController.OpenPin() with the LED\_PIN value.
* Once we have the pin, we set it to be off (High) by default using the GpioPin.Write() function.
* We also set the pin to run in output mode using the GpioPin.SetDriveMode() function.

## 7.3 Modify the state of the GPIO pin

Once we have access to the GpioOutputPin instance, it’s trivial to change the state of the pin to turn the LED on or off.

To turn the LED on, simply write the value GpioPinValue.Low to the pin:

this.pin.Write(GpioPinValue.Low);

and of course, write GpioPinValue.High to turn the LED off:

this.pin.Write(GpioPinValue.High);

Remember that we connected the other end of the LED to the 3.3 Volts power supply, so we need to drive the pin to low to have current flow into the LED.