# Internet of Things Maker Den Lab Guide



Windows 10 IoT Core

Internet of Things

Maker Den Lab Guide

Document Version 2.1

|  |  |
| --- | --- |
| Social | **Twitter** #makerden #iot #raspberrypi2 #windows10 |
| Document Authors | Dave Glover | [dglover@microsoft.com](mailto:dglover@microsoft.com) | @dglover Andrew Coates |[acoat@microsoft.com](mailto:acoat@microsoft.com) |@coatsy Fai Lai |[hoongfai@microsoft.com](mailto:hoongfai@microsoft.com) |@faister |
| Document location | <https://github.com/MakerDen/IoT-Maker-Den-Documentation-and-Resources> |
| Source Code Location | <https://github.com/MakerDen/IoT-Maker-Den-Windows-for-IoT> |
| Disclaimer | All care has been taken to ensure the accuracy of this document. No liability accepted. |
| Copyright | You are free to reuse and modify this document and associated software. |

## Introduction

Welcome to the Internet of Things Maker Den Lab where you will get firsthand experience with hardware prototyping and deploying code to a Raspberry Pi 2 running Windows 10 IoT Core.

## Goal

The goal of the Maker Den is to familiarise you with some of the components and technologies associated with the Internet of Things (IoT). Along the way, you will experience wiring circuits, deploying code, and streaming sensor data to Microsoft Azure.

## Getting Started

If you are setting up your own Maker Den then all source code and documentation is available at <https://github.com/MakerDen/Maker-Den-Documentation-and-Resources-FezHat>.

## Time Required

The lab will take approximately 15 minutes to complete. You are more than welcome to stay longer and delve a little deeper.

## Spread the Word

Be sure to spread the word about the Internet of Things Maker Den on Twitter. Use hash tags #makerden #iot #raspberrypi #windows10

## Lab Hardware

The following components are used for the Maker Den.

|  |  |
| --- | --- |
| [**Raspberry Pi 2**](https://www.raspberrypi.org/)  These labs are built on the Raspberry Pi running Windows 10 IoT Core.  You can find out more about Windows 10 IoT Core at <http://dev.windows.com/iot>. | http://www.raspberrypi.org/wp-content/uploads/2015/01/Pi2ModB1GB_-comp.jpeg |
| [**GHI electronics FEZ HAT**](https://www.ghielectronics.com/catalog/product/500)  The FEZ HAT Key Features:   * On-Board Analog Input and PWM chips. * Two DC Motor Drivers, suitable for building small robots. * Terminal Blocks for wiring in DC motors without the need for soldering. * Two Servo Motor Connections. * Two Multi Color LEDs, connected to PWM for thousands of colors. * Single Red LED. * Temperature Sensor. * Accelerometer. * Light Sensor. * Two user buttons. * Terminal block with 2x Analog, 2x Digital I/O, 2x PWM and power. * Female headers with SPI, I2C, 3x Analog, 3x PWM. * Dedicated power input for driving the servo motors and DC motors. * No Soldering required, completely assembled and tested. | https://www.ghielectronics.com/img/www/products/500-0_large.jpg |

## Experiments

* There are five Maker Den experiments to get you started with Windows 10 IoT Core and Microsoft Azure IoT Services.
* All the source code can be referenced from the Source Code folder on the Desktop.
* This user guide and an architectural overview of the Maker Den can be found in the Documents folder on the Desktop.
* Be sure to check out the [Windows 10 IoT Core Doc, Tutorials and Samples](http://ms-iot.github.io/content/en-US/win10/StartCoding.htm). There is a link to this page in the Desktop Documents folder.
* For the self-sufficient adventurous types, the Windows 10 IoT Core Node.Js and Python developer tools have been installed. Reference the [Windows 10 IoT Core Doc, Tutorials and Samples](http://ms-iot.github.io/content/en-US/win10/StartCoding.htm) and the [Explorer HAT Pro GPIO-pins](https://github.com/pimoroni/explorer-hat/blob/master/documentation/GPIO-pins.md) for more information.

### Resetting the Labs

* STEP 1: Ensure Visual Studio is closed.
* **STEP 2:** Double click the **ResetLabs.bat** file on your desktop. This will copy the source code from a GitHub repository and launch Visual Studio with the solution opened.

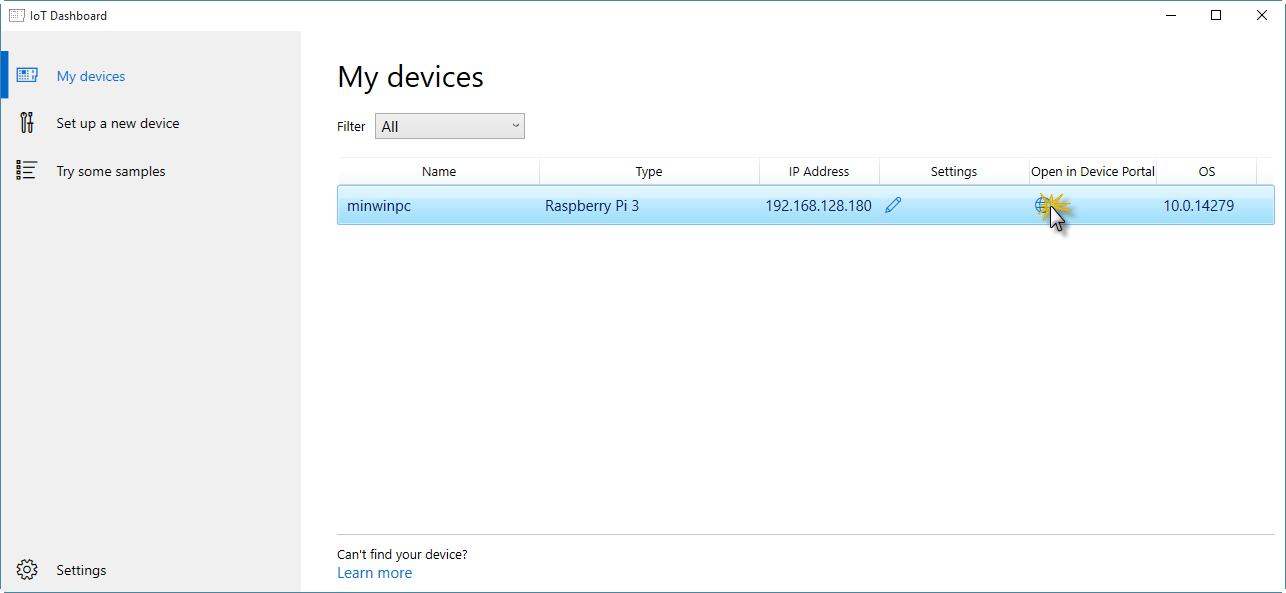
EXPERIMENT 1: Connecting and configuring your device

The Raspberry Pi will be connected to the development PC through a wired Ethernet connection. This connection is used both for deployment and debugging as well as passing through internet requests from the Raspberry Pi when [Internet Connection Sharing](http://ms-iot.github.io/content/en-US/win10/ConnectToDevice.htm) is enabled on the PC.

1. Identifying the device using IoT Core Dashboard

In this task, you'll connect to your device and explore the web management interface.

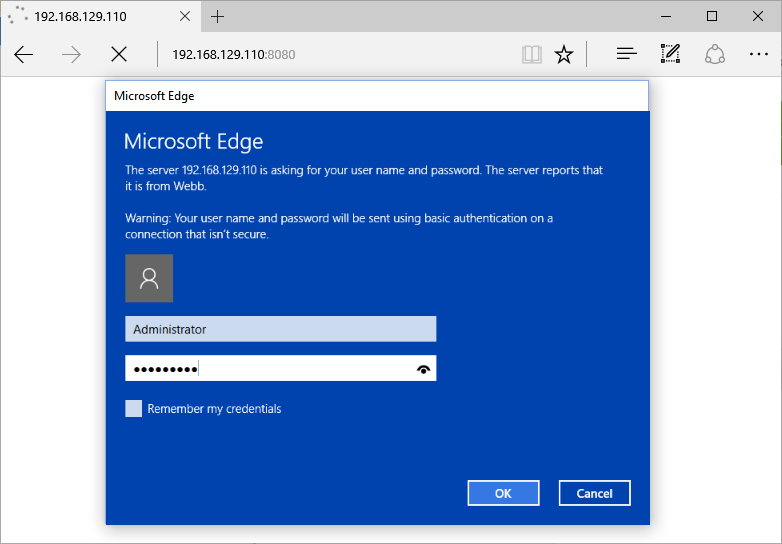
Launch the Windows 10 IoT Core Dashboard, go to My devices and click the Open in Device Portal icon of your device name.

[](https://github.com/gloveboxes/IoT-Camp-2016/blob/master/Module1-IntroWindows10IoTCore/Images/ex1task1-watcher.png?raw=true)

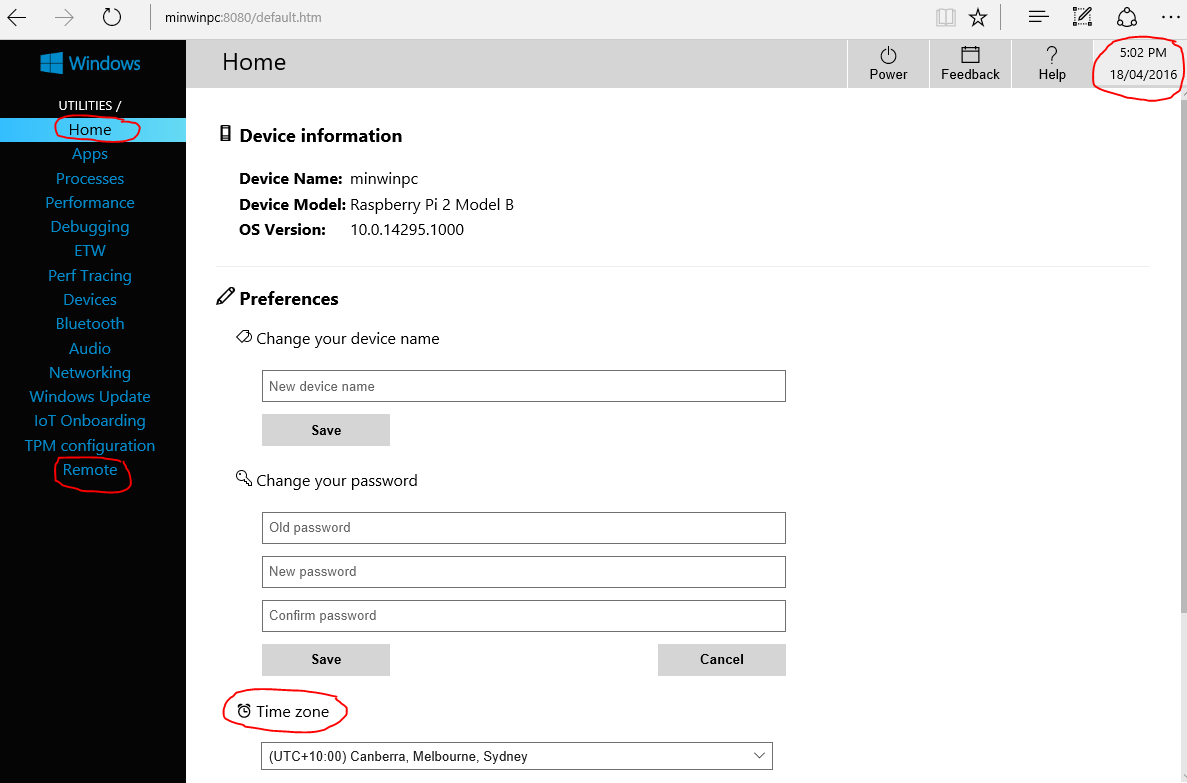
If your device does not show up in the list it is almost certainly because the network connection between your PC and the Raspberry Pi is public and Device Discovery is not enabled. See [How to change Windows 10 network location from Public to Private](https://tinkertry.com/how-to-change-windows-10-network-type-from-public-to-private).

Alternatively, open a web browser and browse to the default device url [http://minwinpc:8080](http://minwinpc:8080/).

In the credentials dialog, use the default username and password. Username: *Administrator* Password: *p@ssw0rd*

[](https://github.com/gloveboxes/IoT-Camp-2016/blob/master/Module1-IntroWindows10IoTCore/Images/ex1task1-device-portal-credentials.png?raw=true)

**Windows Device Portal** should launch and display the web management home screen!

[](https://github.com/gloveboxes/IoT-Camp-2016/blob/master/Module1-IntroWindows10IoTCore/Images/ex1task1-device-portal.png?raw=true)

Windows Device Portal

1. Verify Device Configuration

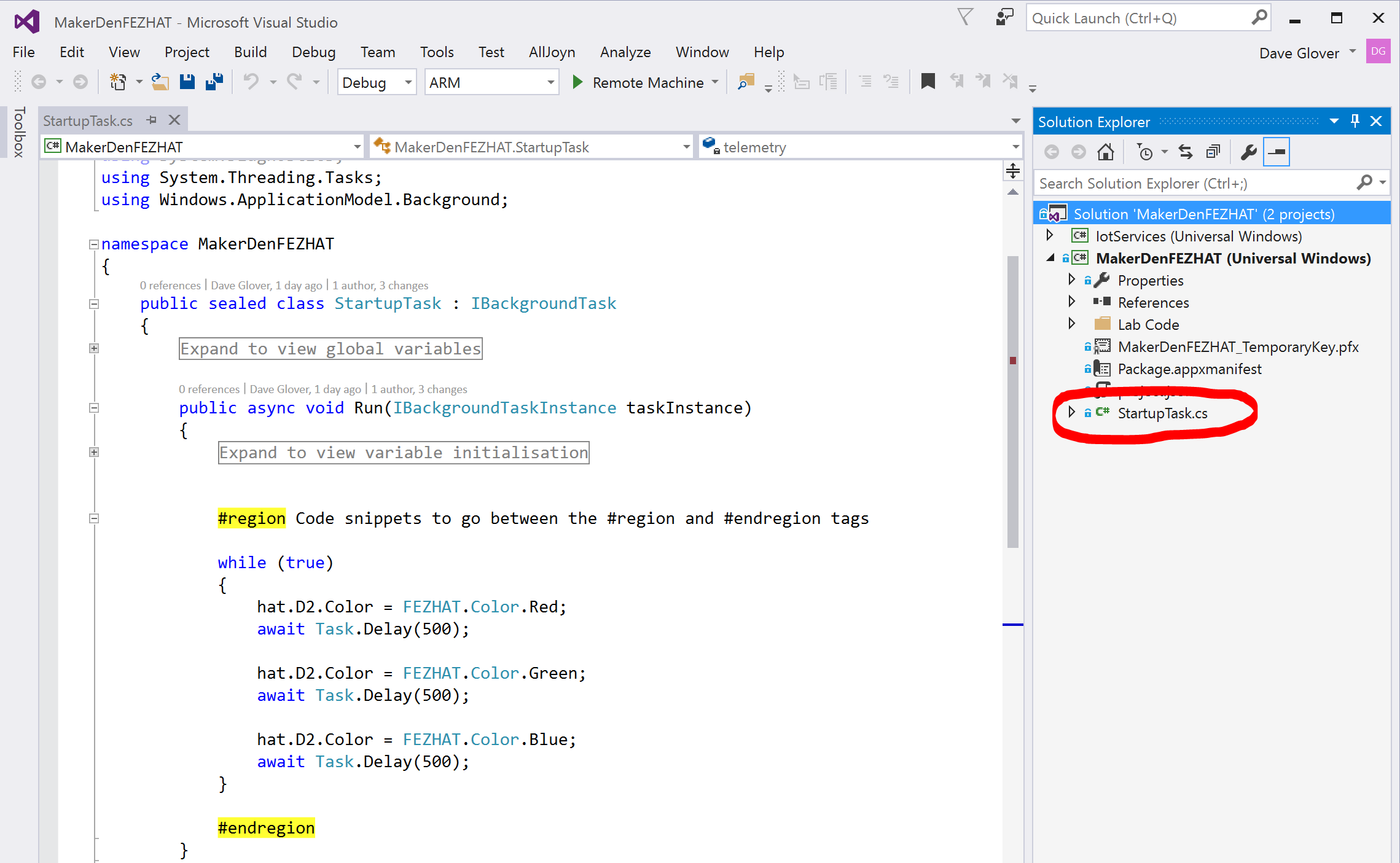
From the **Home** Tab verify the Time Zone, date and time are correct. If you change the Time zone you will need to reboot.

From the **Remote** tab verify that **Windows IoT Remote Server** enabled. If it is not, then enable it.

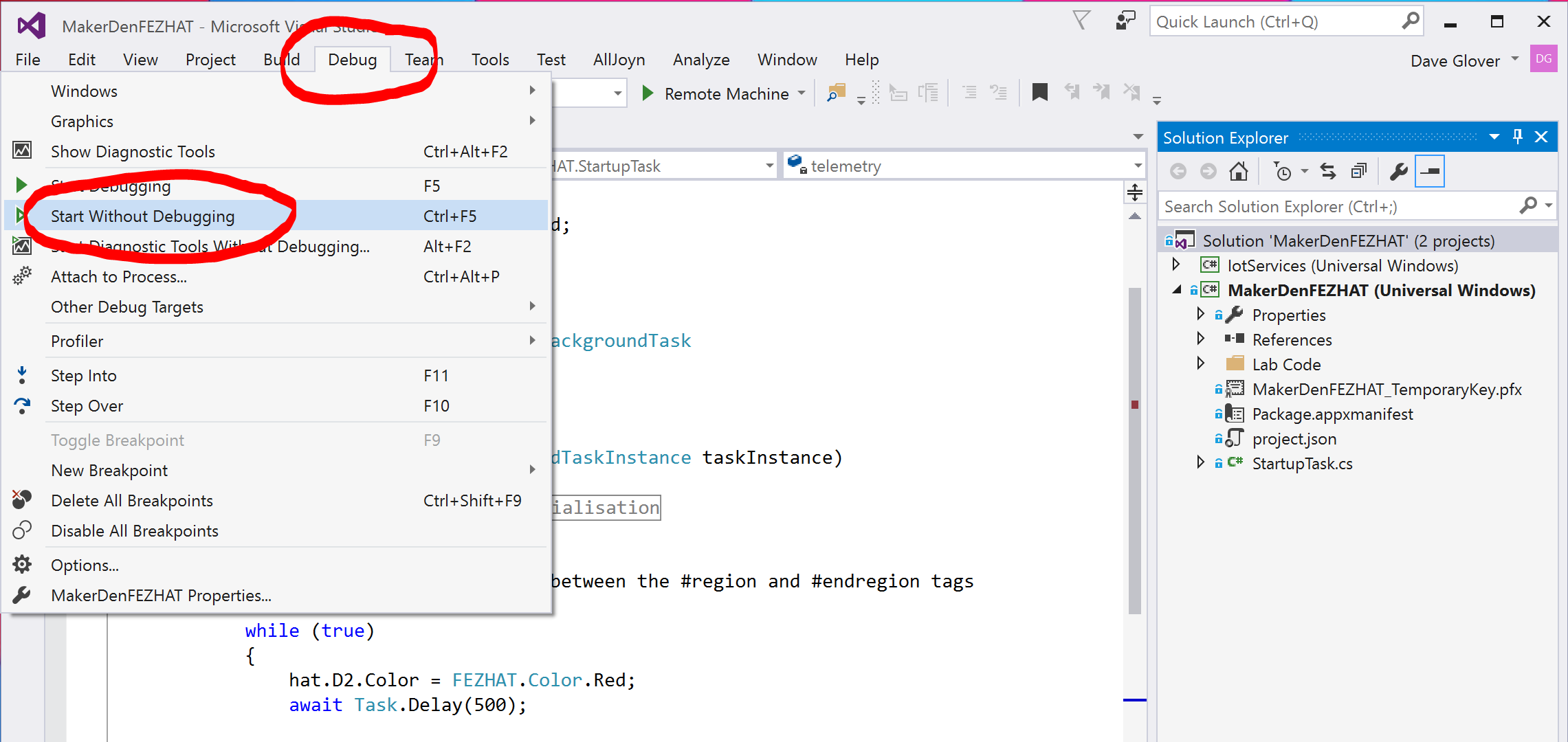
Experiment 1: Hello WORLD

Deploy your first experiment to ensure everything is setup correctly and to check Visual Studio is communicating with your Raspberry Pi.

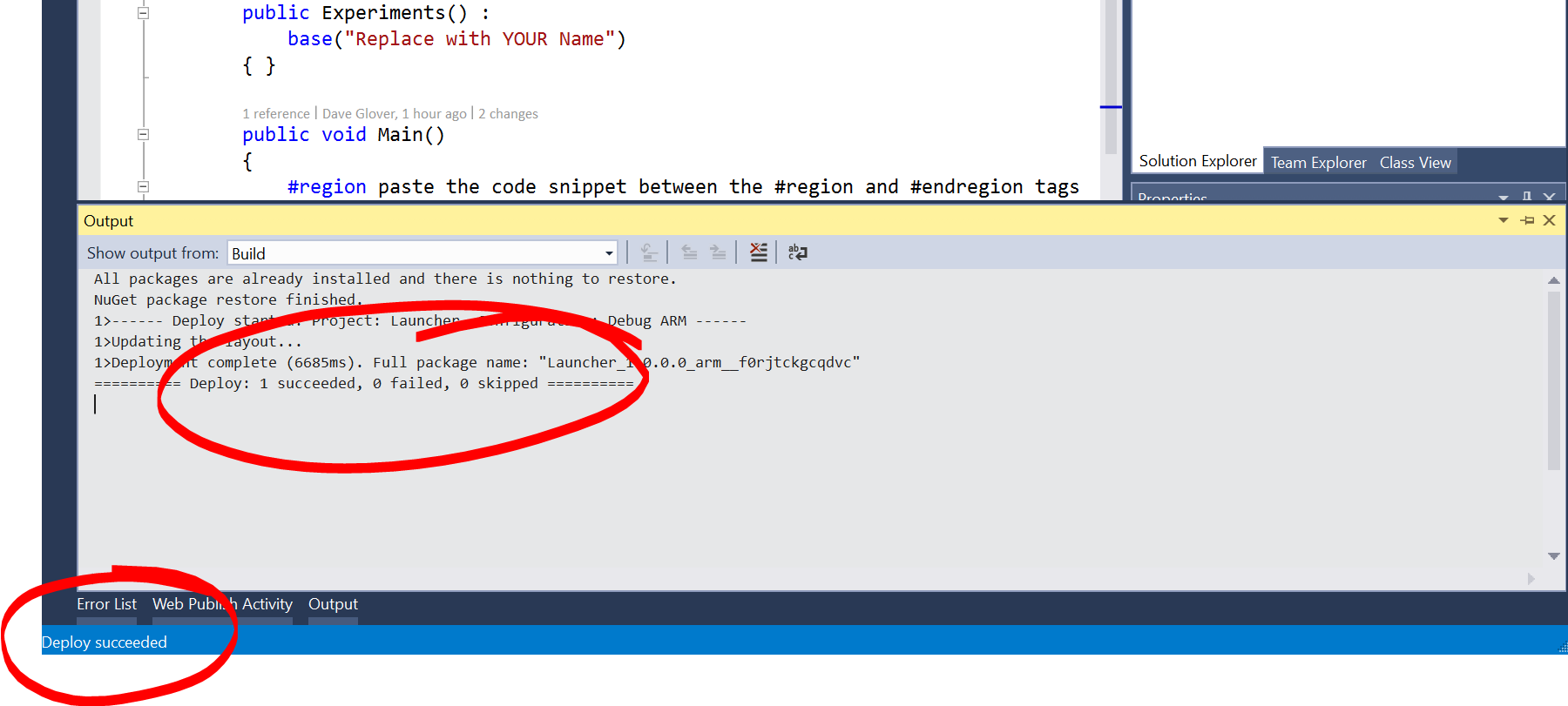
* **STEP 1**: Expand the **MakerDen** project then double click the **StartupTask.cs** file to open it.



* **STEP 2:** Deploy the solution to the Raspberry Pi. From the **Debug** menu select **Start Without Debugging** or from the keyboard press **Ctrl+F5.**



* **STEP 3:** Check that Visual Studio has successfully compiled and deployed the code by looking at the output window and the status bar.

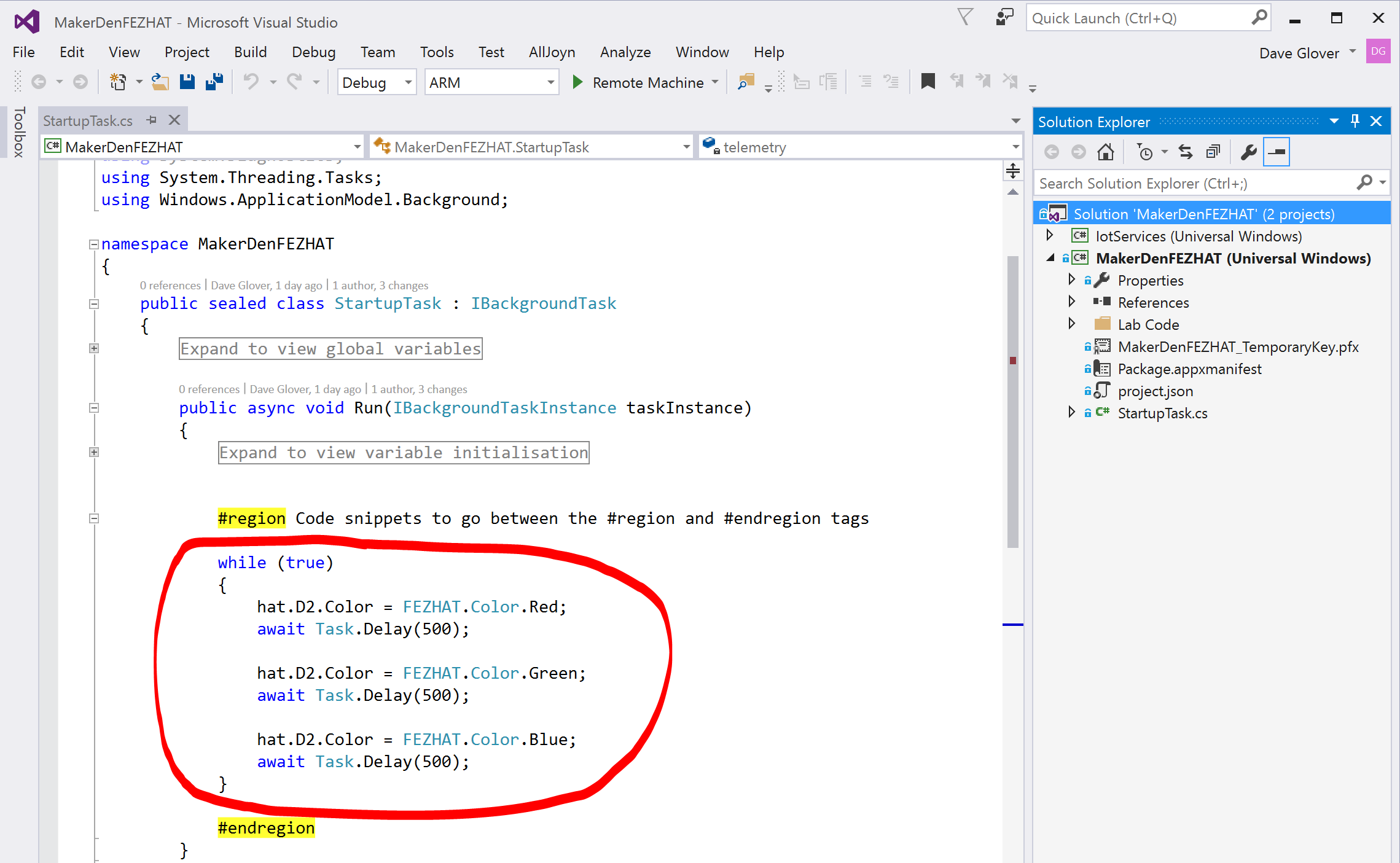


* **STEP 4:** Check the LED Matrix on the Raspberry Pi. You should see your name, the machine name and the IP Address scrolling on the LED Matrix display.
* **STEP 5:** Pat yourself on the back, you did it☺

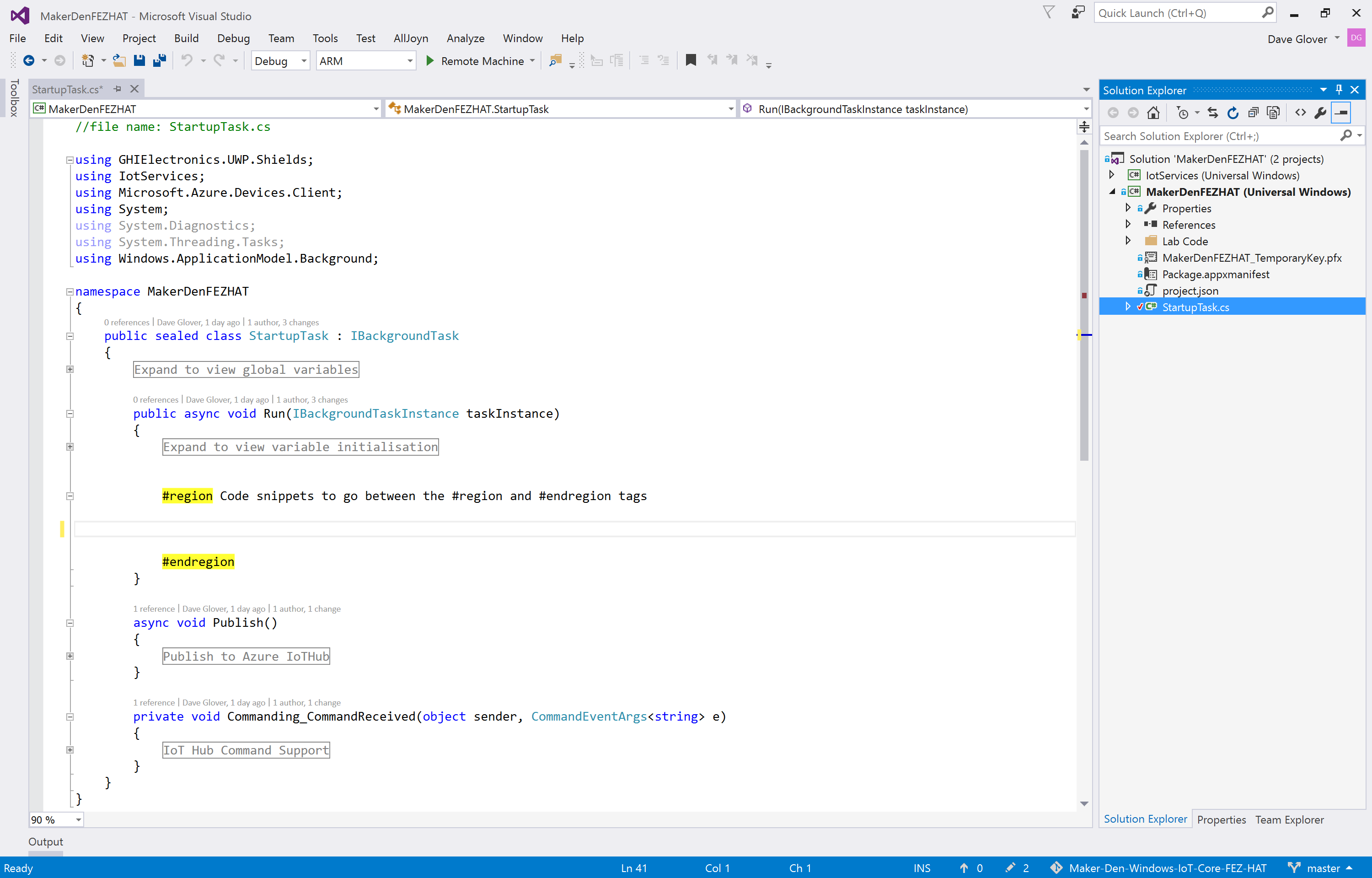
Experiment 2: Sensing the World

This lab reads the current light levels from the light sensor.

* **STEP 1:** Review the code in the **StartupTask.cs** file. Look for the **#region** and **#endregion** tags.
* **STEP 2:** Delete the code circled in red **inside** the #region tags.



Your “StartupTask.cs” file should look like the screenshot below after you have deleted the code. If it doesn’t look the same then **Ctrl+Z** to undo the changes you made and try again.



* **STEP 3:** Type the following code between the #region tags **OR** using a code snippet type **lab2** and press Tab twice.

while (true)

{

var level = hat.GetLightLevel() \* 100;

if (hat.GetLightLevel() \* 100 > LIGHT\_THRESHOLD)

{

hat.D2.Color = FEZHAT.Color.Green;

}

else

{

hat.D2.Color = FEZHAT.Color.Red;

}

await Task.Delay(500);

}

* **STEP 4:** Your “StartupTask.cs” file should like look like the following. If not, **Ctrl+Z** and try again.

//file name: StartupTask.cs

using GHIElectronics.UWP.Shields;

using IotServices;

using Microsoft.Azure.Devices.Client;

using System;

using System.Diagnostics;

using System.Threading.Tasks;

using Windows.ApplicationModel.Background;

namespace MakerDenFEZHAT

{

public sealed class StartupTask : IBackgroundTask

{

Expand to view global variables

public async void Run(IBackgroundTaskInstance taskInstance)

{

Expand to view variable initialisation

#region Code snippets to go between the #region and #endregion tags

while (true)

{

var level = hat.GetLightLevel() \* 100;

if (hat.GetLightLevel() \* 100 > LIGHT\_THRESHOLD)

{

hat.D2.Color = FEZHAT.Color.Green;

}

else

{

hat.D2.Color = FEZHAT.Color.Red;

}

await Task.Delay(500);

}

#endregion

}

async void Publish()

{

#region Publish to Azure IoTHub

#endregion

}

private void Commanding\_CommandReceived(object sender, CommandEventArgs<string> e)

{

#region IoT Hub Command Support

#endregion

}

}

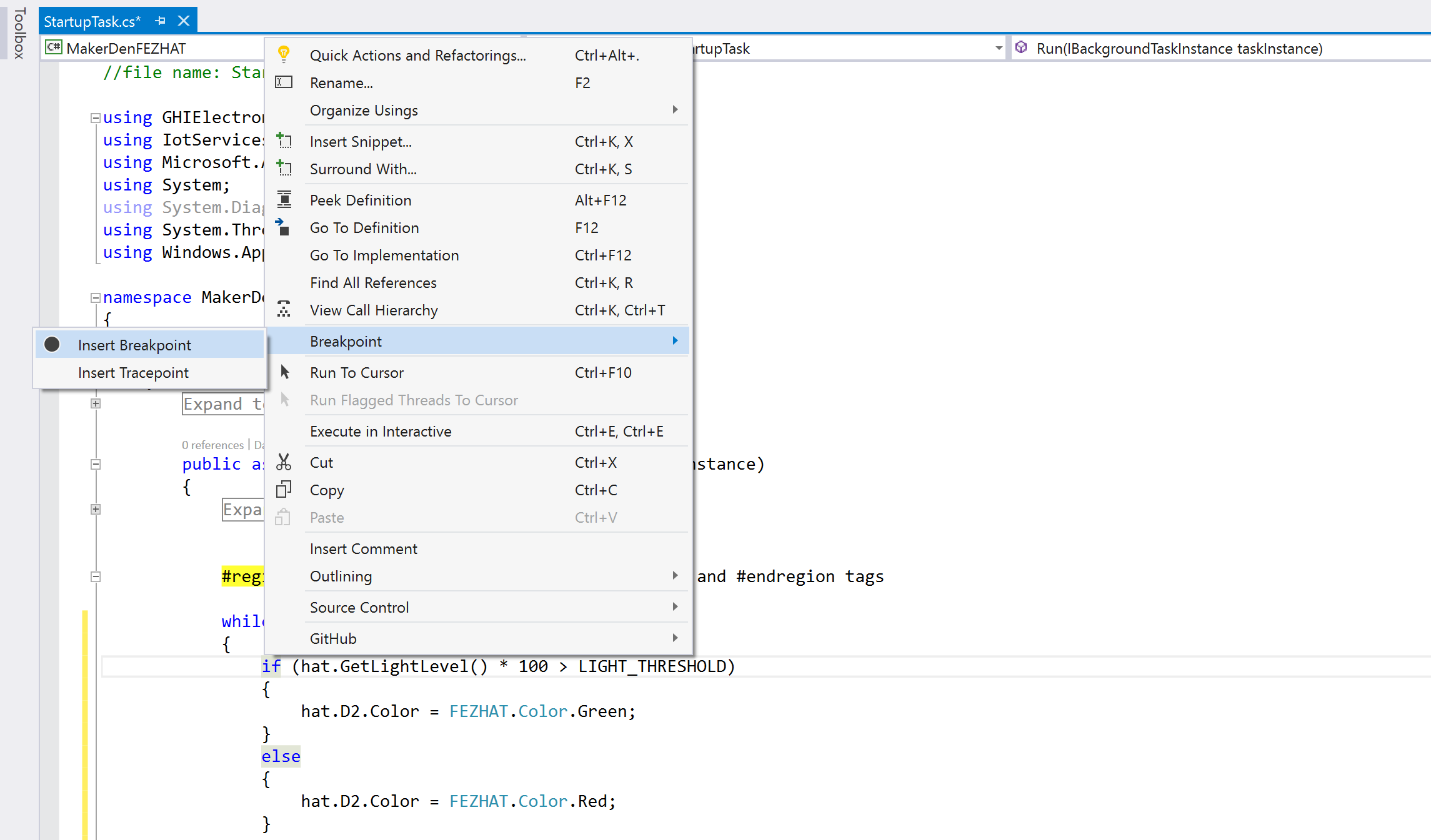
}

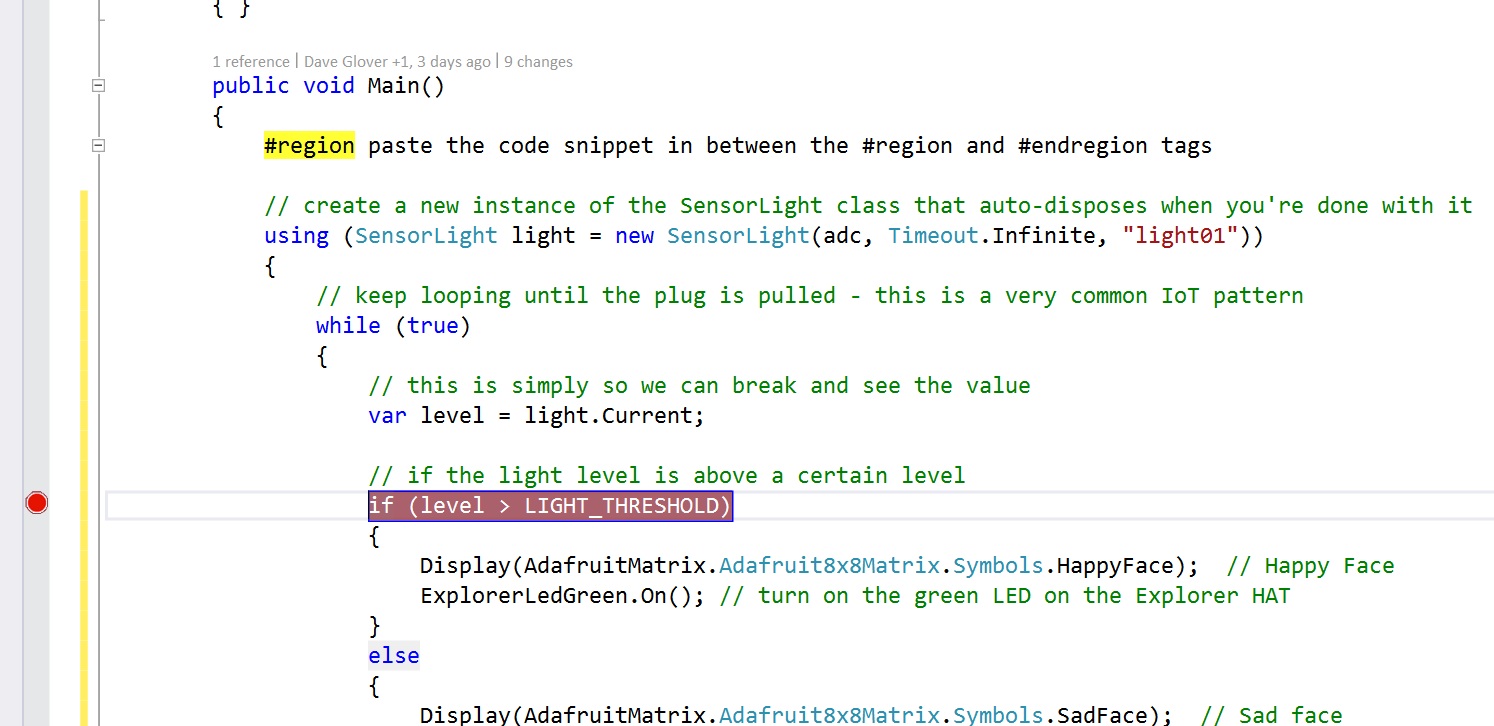
* **STEP 5:** Deploy the solution to the Raspberry Pi. From the **Debug** menu select **Start Without Debugging** or from the keyboard press **Ctrl+F5.**
* **STEP 6:** Hover your hand over the light sensor and observe that the face changes and the green LED turns off when it gets dark.

Experiment 3: Remote Debugging

* **STEP 1:** Next, set a break point to see how easy it is to debug directly on the device. This is a unique capability provided by Visual Studio.  
    
  Right-click on the line that reads if (level > LIGHT\_THRESHOLD)

Choose Breakpoint, then Insert Breakpoint.





* **STEP 2:** From the **Debug** menu select **Start Debugging** or on the keyboard press **F5** and wait for the solution to deploy and for Visual Studio to hit the breakpoint.
* **STEP 3:** Hover the cursor over the variable “level” and Visual Studio will display its current value.
* **STEP 4:** While holding your hand over the light sensor, press F5 a couple of times to continue and observe that the face changes and the green LED turns off when it gets dark.
* **Step 5**: Press Shift-F5 to stop debugging.

Experiment 4: Connecting to Azure IOT HUB

This lab connects the Raspberry Pi 2 to Azure cloud services. The Maker Den supports three cloud modes.

1. Azure IoT Hub (<https://azure.microsoft.com/services/iot-hub/>),
2. Azure Event Hubs (<https://azure.microsoft.com/services/event-hubs>)
3. MQTT ([Mosquitto.org](http://mosquitto.org/) service running in an Azure VM)

* **STEP 1:** Delete the code between the #region tags
* **STEP 2:** Type the following code between the #region tags **OR** using a code snippet type **lab4** and press Tab twice.

using (SensorMgr lightSensor = new SensorMgr(light))

using (SensorMgr tempSensor = new SensorMgr(temp))

{

lightSensor.OnAfterMeasurement += OnAfterMeasurement;

lightSensor.OnBeforeMeasurement += OnBeforeMeasure;

lightSensor.OnAfterMeasurement += SetLEDMatrixBrightness;

tempSensor.OnAfterMeasurement += OnAfterMeasurement;

tempSensor.OnBeforeMeasurement += OnBeforeMeasure;

DisplayTemperature().Wait();

}

* **STEP 3**: Enable IoT Hub CloudMode. Change the line that reads

Initialise("Replace with Your Name", CloudMode.None);

To

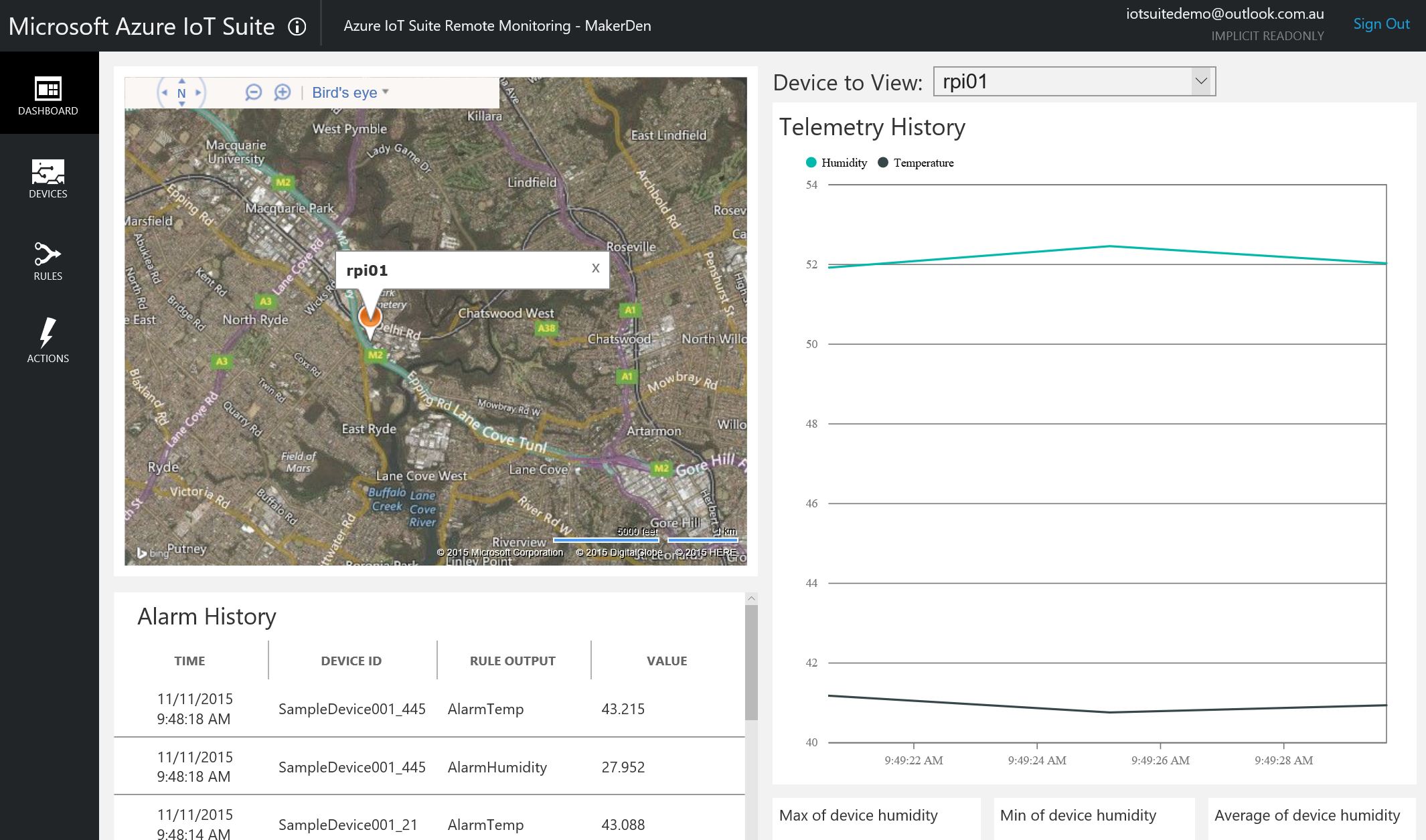
Initialise("Replace with Your Name", CloudMode.IoT\_Hub);

* **STEP 4:** Deploy the solution to the Raspberry Pi. From the **Debug** menu select **Start without Debugging** or from the keyboard press **Ctrl+F5** and wait for the solution to deploy.
* **STEP 5:** Open a web browser and go to the URL <http://aka.ms/makerdeniothub>. This is the Remote Monitoring web dashboard that is part of one of the preconfigured solutions in [Azure IoT Suite](https://www.azureiotsuite.com).
* **STEP 6:** **Sign In** using this Microsoft Account:

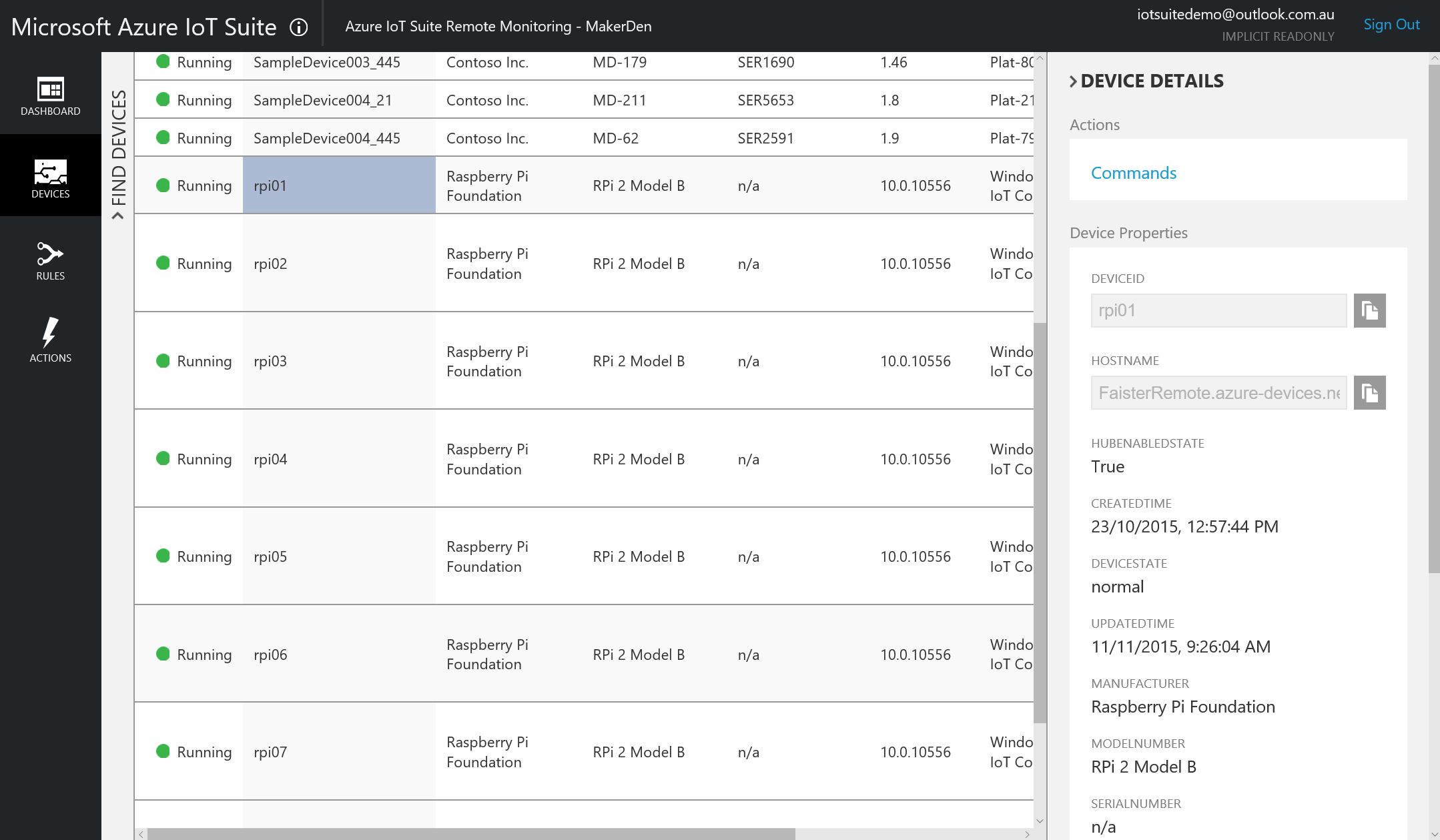
Email: [iotsuitedemo@outlook.com.au](mailto:iotsuitedemo@outlook.com.au)

Password: IoThub2015

* **STEP 7:** In the “Device to View” drop down list, **select** your device ID. This is the same as the hostname of your Raspberry Pi 2 running Windows 10 IoT Core. Once you have selected the right device to view, you will see the Telemetry History graph, and 3 gauges for device humidity (NOTE: only temperature reading is read from the device. Humidity values are randomly generated). You may zoom in the map to locate your device’s pin drop.



* **STEP 8:** Touch your finger on the temperature sensor and observe the temperature changing.
* **STEP 9: Click DEVICES** in the left navigation bar. This brings you to the Devices List. **Select** your device. **Click DEVICE DETAILS** in the right-hand corner of the dashboard. You will find the device metadata associated to this device. This is stored within a DocumentDB store which is also preconfigured as part of the Azure IoT Suite Remote Monitoring solution.



Experiement 4: Streaming telemetry data to Azure IoT hub

* **STEP 1:**

public async void Run(IBackgroundTaskInstance taskInstance)

{

Expand to view variable initialisation

deviceClient = DeviceClient.CreateFromConnectionString("device connection string");

#region Code snippets to go between the #region and #endregion tags

while (true)

{

var level = hat.GetLightLevel() \* 100;

if (level > LIGHT\_THRESHOLD)

{

hat.D2.Color = FEZHAT.Color.Green;

}

else

{

hat.D2.Color = FEZHAT.Color.Red;

}

await Task.Delay(500);

}

#endregion

}

* **STEP 2:** Type the following code in the **Publish** method between the **#region Publish to Azure IoTHub** tags **OR** using a code snippet type **lab4** and press Tab twice.

try // Exception handling if problem streaming telemetry to Azure IoT Hub

{

hat.D3.Color = publishColor; // turn on publish indicator LED

var temperature = hat.GetTemperature(); // read temperature from the FEZ HAT

var light = hat.GetLightLevel(); // read light level from the FEZ HAT

var json = telemetry.ToJson(temperature, light, 0, 0); //serialise to JSON

var content = new Message(json);

await deviceClient.SendEventAsync(content); //Send telemetry data to IoT Hub

}

catch { telemetry.Exceptions++; }

finally { hat.D3.TurnOff(); }

* **STEP 3:** Your completed Publish method should look like this

async void Publish()

{

#region Publish to Azure IoTHub

try // Exception handling if problem streaming telemetry to Azure IoT Hub

{

hat.D3.Color = publishColor; // turn on publish indicator LED

var temperature = hat.GetTemperature(); // read temperature from the FEZ HAT

var light = hat.GetLightLevel(); // read light level from the FEZ HAT

var json = telemetry.ToJson(temperature, light, 0, 0); //serialise to JSON

var content = new Message(json);

await deviceClient.SendEventAsync(content); //Send telemetry data to IoT Hub

}

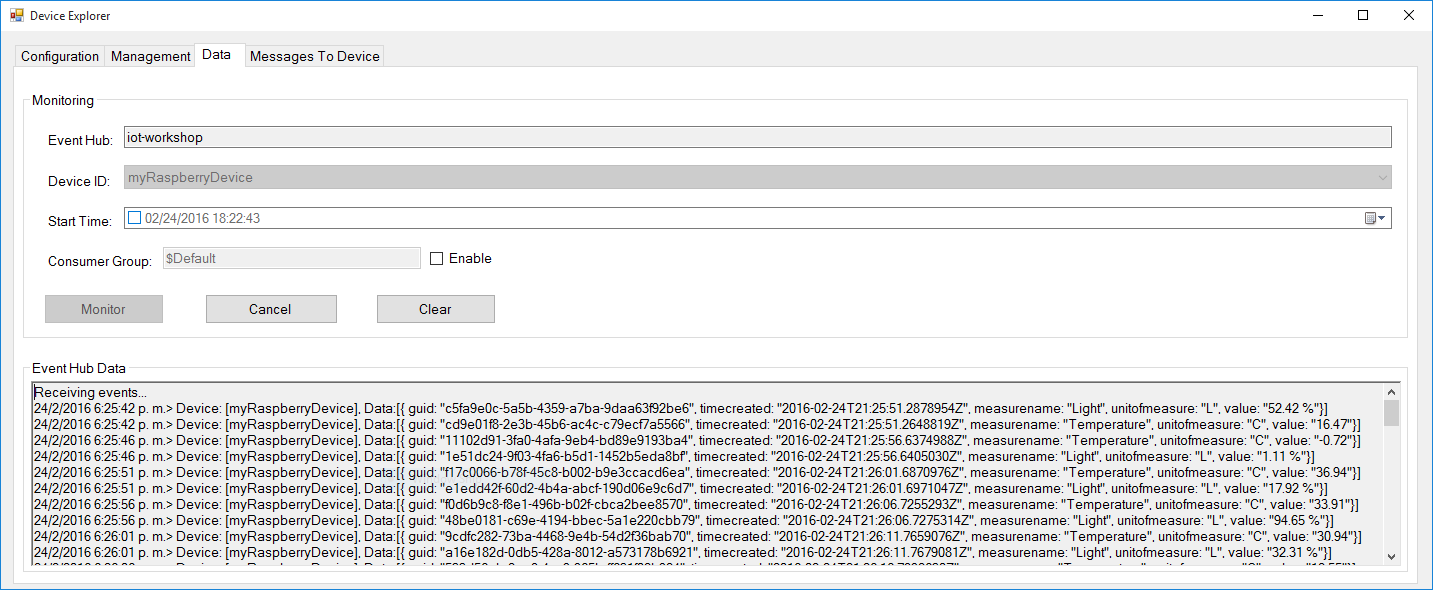
catch { telemetry.Exceptions++; }

finally { hat.D3.TurnOff(); }

#endregion

}

* **STEP 4:** Deploy the solution to the Raspberry Pi. From the **Debug** menu select **Start without Debugging** or from the keyboard press **Ctrl+F5** and wait for the solution to deploy.

[](https://github.com/gloveboxes/IoT-Camp-2016/blob/master/Module2-WindowsIoTCorePi2FezHat-IoTHubs/Images/monitoring-messages-sent.png?raw=true)

**Note**: If you navigate back to your IoT Hub blade in the Azure Portal, it may take a couple minutes before the message count is updated to reflect the device activity under **Usage**.

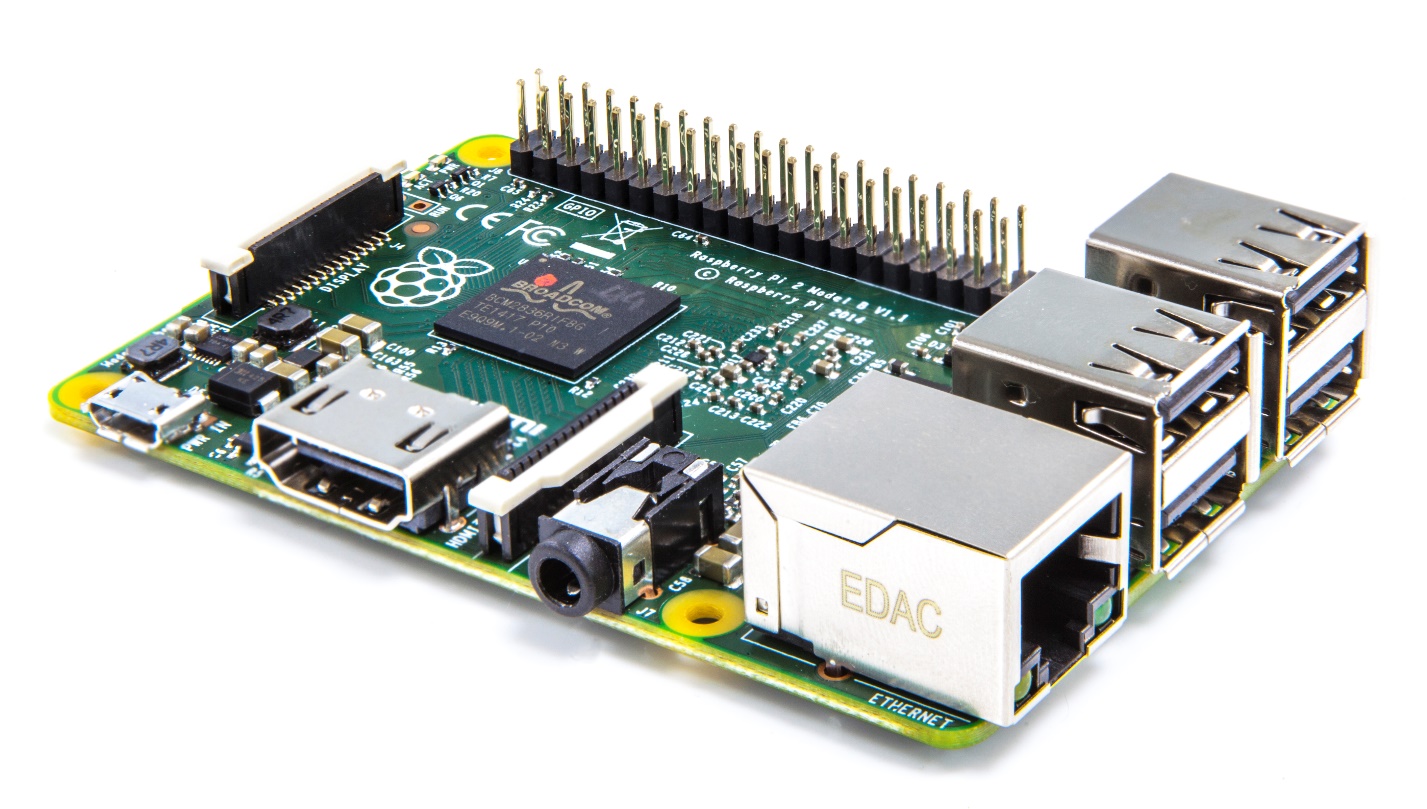
Experiment 5: Not so Fast Cowboy

Congratulations, you have successfully completed the Maker Den Experience. You have done some hardware prototyping, deployed a Universal Windows App to a Raspberry Pi 2. You have streamed data to Microsoft Azure, ingested telemetry using Azure IoT Hub and visualised data with the [Azure IoT Suite Remote Monitoring](https://github.com/Azure/azure-iot-remote-monitoring/wiki) solution. Optionally you also streamed data over MQTT and visualised on the IoT Dashboard.

Please complete the following steps before you leave.

* **STEP 1**: Close Visual Studio
* **STEP 2**: Leave all the components in the breadboard and the Raspberry Pi running and streaming data to Azure.
* **STEP 3:** **Take 30 seconds to complete the Maker Den Evaluation at** <http://aka.ms/ignite2015makerden> (there is a shortcut on the desktop)
* **STEP 4:** Make sure you get scanned to be in the draw for a Raspberry Pi 2.

WIN!



All the documentation and software for the Maker Den is available at <http://www.github.com/makerden>

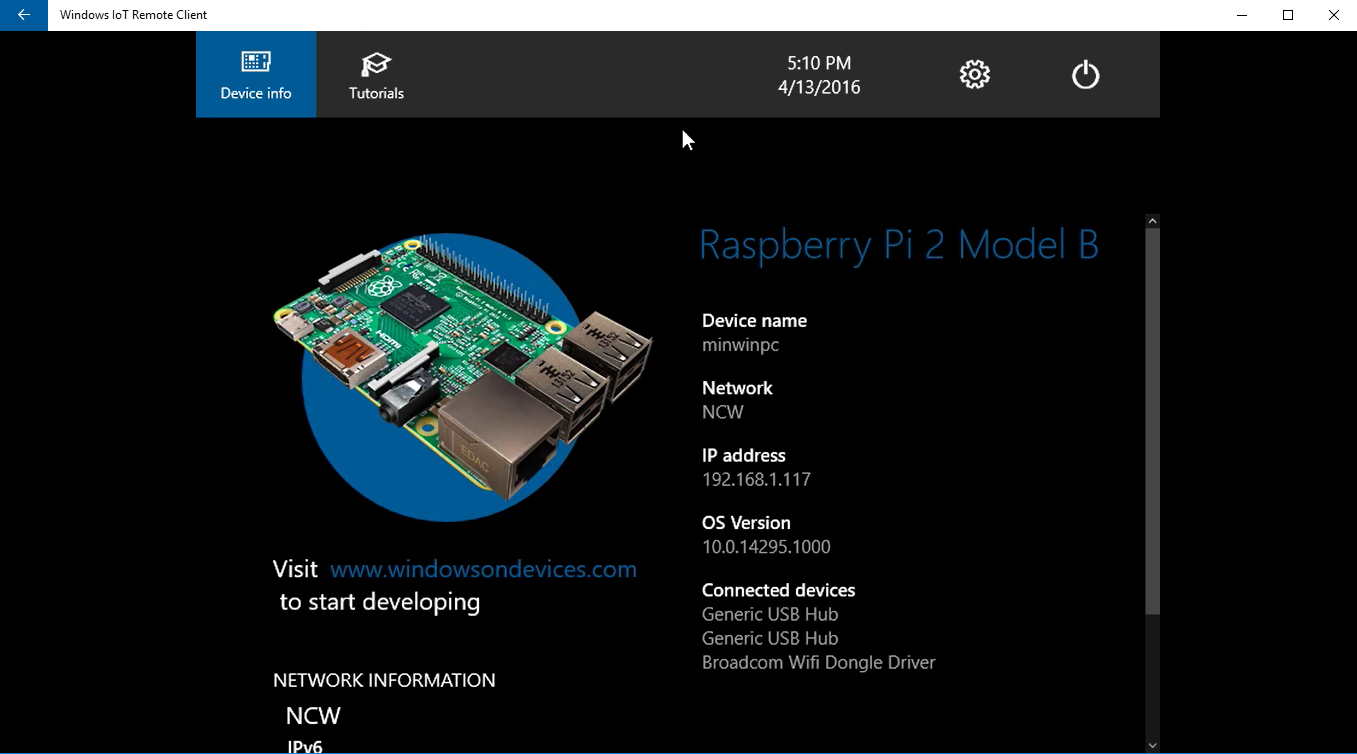
1. Verify the Windows IoT Remote Client connection

Press the Windows key and type “Windows IoT Core Remote Client” and run the app.

It is likely that you will need to enter the IP address of your Raspberry Pi. Get the address of the device from the **Windows 10 IoT Core Dashboard**.

This will take a moment to connect. When it does you will see the video output of the Raspberry Pi remoted to your desktop.

Minimize the remote client application when you have verified that it is working.

[](https://github.com/gloveboxes/IoT-Camp-2016/blob/master/Module1-IntroWindows10IoTCore/Images/windows-iot-remote-client.png?raw=true)

Exercise 2: Create and deploy "Hello World" UWP

A **Universal Windows Platform (UWP)** app has the potential to run on any Windows-powered device like a Raspberry Pi device with Windows 10 IoT core.

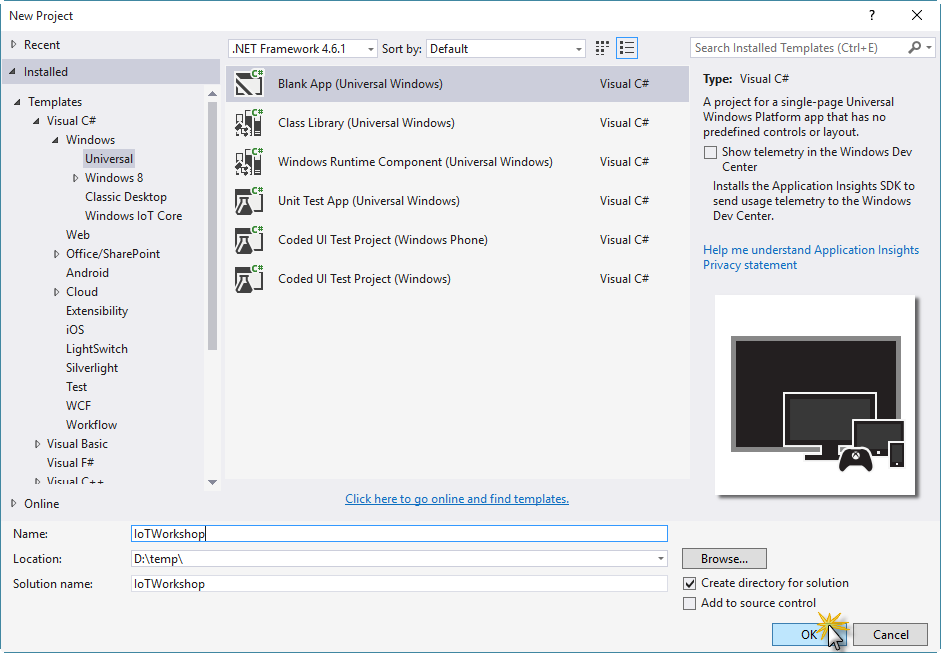
**Windows IoT Core** can be configured for either **headed** or **headless** mode. The difference between these two modes is the presence or absence of any form of UI. By default, Windows 10 IoT Core is in headed mode and runs the default startup app which displays system information like the computer name, connected devices, and IP addresses.

In this exercise, you'll create and deploy a UWP app for headed mode using a XAML view to display a TextBlock and a button that updates the TextBlock content.

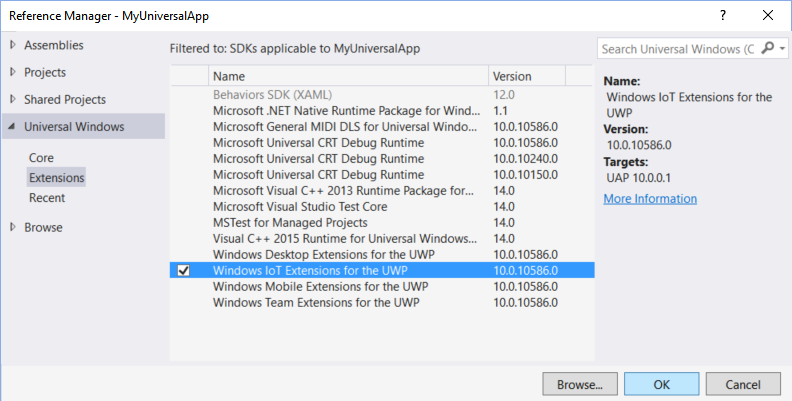
1. Creating a Hello World UWP app

In this task, you'll use the Universal project template to create a Blank App. Then you'll add some UI elements and verify the app by debugging the application on your local PC.

1. Start Visual Studio 2015 and create a new project File > New Project...
2. In the installed templates tree, navigate to Visual C# > Windows > Universal and select the template Blank App (Windows Universal). Enter the name "IoTWorkshop".

[](https://github.com/gloveboxes/IoT-Camp-2016/blob/master/Module1-IntroWindows10IoTCore/Images/ex2task1-new-blank-app.png?raw=true)

1. Click OK to create the project.
2. Add a reference to the Windows IoT extension SDK. Right-click the References entry under the project, select Add Reference then navigate the resulting dialog to Universal Windows > Extensions > Windows IoT Extensions for the UWP, check the box, and click OK.

[](https://github.com/gloveboxes/IoT-Camp-2016/blob/master/Module1-IntroWindows10IoTCore/Images/ex2task1-IoT-SDK-ref.png?raw=true)

1. *Add Windows IoT extension SDK*
2. From Solution Explorer, select the MainPage.xaml file.
3. Locate the <Grid> tag within the XAML section of the designer, and add the following markup inside the grid. This will add an input and a button that you'll see in the design surface.

<StackPanel HorizontalAlignment="Center" VerticalAlignment="Center">

<TextBox x:Name="Message" Text="Hello, World!" Margin="10" IsReadOnly="True"/>

<Button x:Name="ClickMe" Content="Click Me!" Margin="10" HorizontalAlignment="Center"/>

</StackPanel>

1. Double click the Click Me! button in the design surface to generate the click method in the *MainPage.xaml.cs* file.
2. Add the following line to the ClickMe\_Click method:

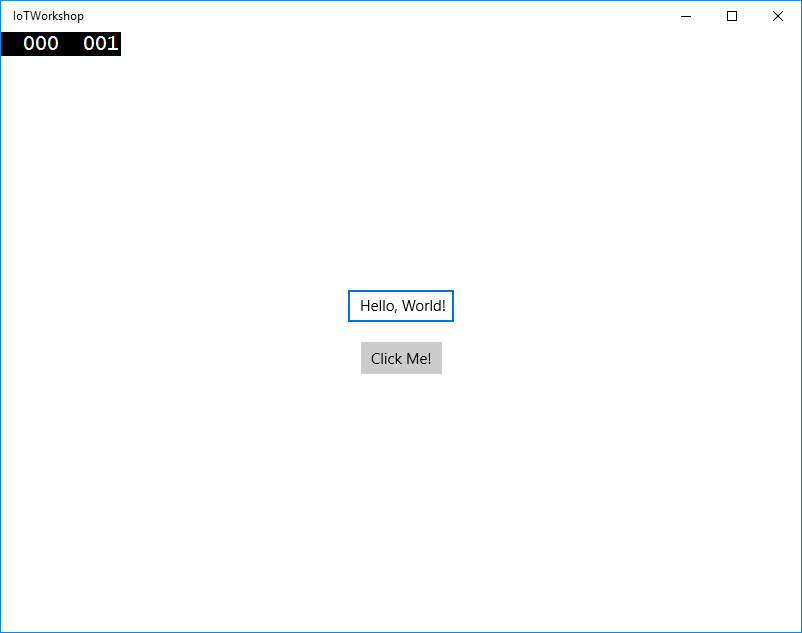
private void ClickMe\_Click(object sender, RoutedEventArgs e)

{

this.Message.Text = "Hello, Windows IoT Core!";

}

1. Press F5 to build and run the app locally (since this is a Universal Windows Platform (UWP) application, you can test the app on your Visual Studio machine as well).

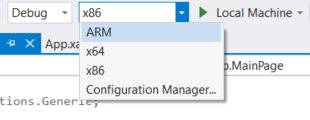
[](https://github.com/gloveboxes/IoT-Camp-2016/blob/master/Module1-IntroWindows10IoTCore/Images/ex2task1-hello-world-local.png?raw=true)

1. Click the **Click Me!** button to verify that the input message changes and close the app after you're done verifying it.
2. Deploying your app to the device

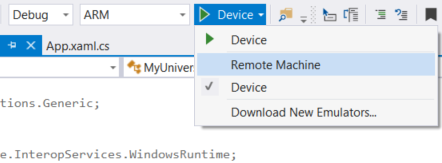
Now that you validated the app in your local PC, you can deploy it to the Raspberry Pi device by using remote debugging.

In order to use remote debugging, your IoT Core device must first be connected to same local network as your development PC.

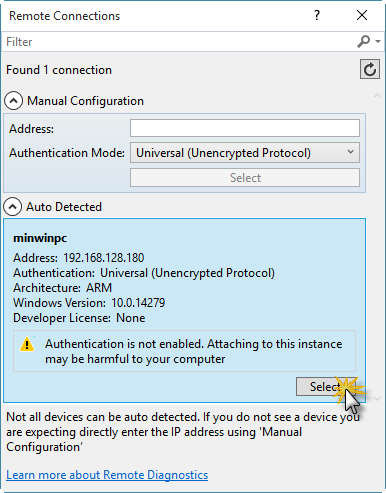
1. In Visual Studio, continue with the same solution that you created in the previous task and select the **ARM** architecture in the toolbar dropdown.

[](https://github.com/gloveboxes/IoT-Camp-2016/blob/master/Module1-IntroWindows10IoTCore/Images/ex2task2-arm.png?raw=true)

1. Next, click the **Device** dropdown and select **Remote Machine**.

[](https://github.com/gloveboxes/IoT-Camp-2016/blob/master/Module1-IntroWindows10IoTCore/Images/ex2task2-run-remote-machine.png?raw=true)

1. In the **Remote Connections** dialog, click your device name within the **Auto Detected** list and then click **Select**. Not all devices can be auto detected, if you don't see it, enter the IP address using the **Manual Configuration**. After entering the device name/IP, select **Universal (Unencrypted Protocol)** Authentication Mode, then click **Select**.

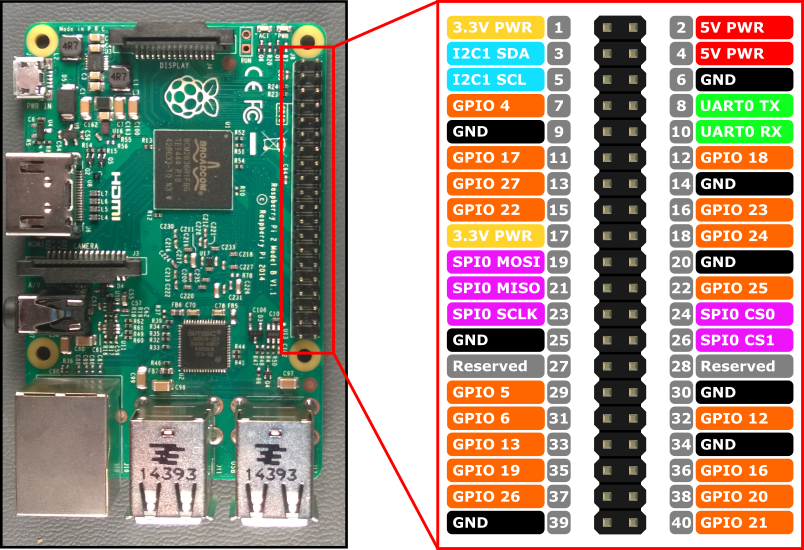
[](https://github.com/gloveboxes/IoT-Camp-2016/blob/master/Module1-IntroWindows10IoTCore/Images/ex2task2-remote-connections-dialog.png?raw=true)

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.
2. Build and deploy your app to your device by selecting **Build > Rebuild Solution and Build > Deploy Solution**.
   1. If there are any missing packages that you did not install during setup, Visual Studio may prompt you to acquire those now.
3. Hit **F5** to deploy and run the app. You can insert breakpoints and debug in the remote device.
4. Restore the "Windows IoT Remote Client" app and you should see the output of your app running on the Raspberry Pi.

Exercise 3: Using Windows.Devices.Gpio

The **Windows.Devices.Gpio** namespace includes APIs for direct access to the IO pins on the device. Through this API, you can access digital and analog IO, I2C, SPI, and more.

The individual GPIO (General Purpose IO) pins on the Raspberry Pi may be addressed from code using the UWP GPIO APIs. We will use one of these pins to toggle an LED. You will access the GPIO pins using their logical GPIO number, not the the physical pin number. This is consistent with how other APIs and operating systems work on the Raspberry Pi.

[](https://camo.githubusercontent.com/711bd59989d320a08150416c29480d5ec20be253/687474703a2f2f6d732d696f742e6769746875622e696f2f636f6e74656e742f696d616765732f50696e4d617070696e67732f5250325f50696e6f75742e706e67)

1. **Programming the Device IO using the GPIO Controller**

Of course, you can hook an LED and resistor directly to one of the pins using a breadboard and jumper wires (in fact, we encourage you to try that during the Open Hack), but to keep things simple, we'll toggle the red LED on the GHI FEZ HAT that is already on your Raspberry Pi.

To do this, we'll create a new project.

1. In Visual Studio 2015, create a new **C# Blank App (Universal Windows)**. Name it anything you want. We named ours **IoTHelloBlinky**.
2. When prompted for the SDK version, set the minimum and max SDK version to 10586.
3. You'll use a ToggleButton to turn the LED on and off. In the MainPage.xaml XAML view, place the following markup inside the opening and closing Grid tags.

<ToggleButton x:Name="ToggleLed" Content="Toggle LED" FontSize="40"

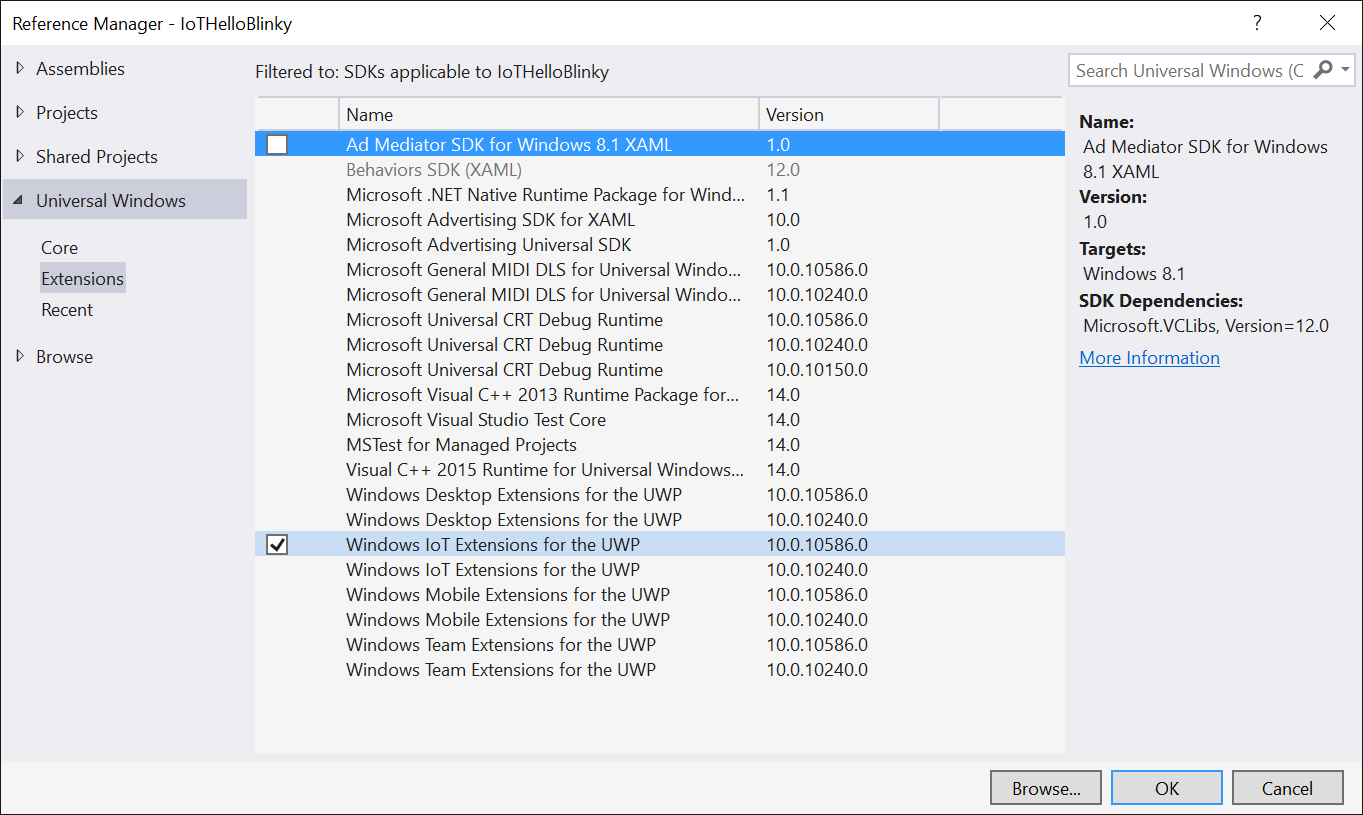
Padding="15"

HorizontalAlignment="Center" VerticalAlignment="Center"

Checked="ToggleLed\_Checked"

Unchecked="ToggleLed\_Unchecked" />

1. Next, you need some code to light up the LED. However, before that, you'll need to add a reference to the IoT UWP extension library to get access to the **Windows.Devices.Gpio** namespace. As before, use the **Project - Add Reference menu** to add the extension. Be sure to check it in the dialog, not just select it. If you have more than one version listed, select the one with the highest number. In a real application, you'll want to keep this in sync with the version of Windows on the device so that you have access to all of the latest features.

[](https://github.com/gloveboxes/IoT-Camp-2016/blob/master/Module1-IntroWindows10IoTCore/Images/add-iot-extension.png)

1. Now that you have the extension SDK in place, you can add the code. Open the MainPage.xaml.cs code-behind file and add the following namespace to the using statements at the top of the file

using Windows.Devices.Gpio;

1. Next, add the code to initialize the GPIO controller and pin. The Red LED on the FEZ HAT is connected directly to GPIO 24 on the Raspberry Pi. What this code does is get the default GPIO Controller (which maps to a device driver in Windows), and opens the Red LED Pin for output. This is detailed in the [FEZ HAT schematic](http://www.ghielectronics.com/downloads/schematic/FEZ_HAT_SCH.pdf). Finally, it writes a **low** value to the pin to turn the LED off.

public MainPage()

{

this.InitializeComponent();

InitializeGpio();

}

private const int LED\_PIN = 24;

private GpioPin \_pin;

private void InitializeGpio()

{

var controller = GpioController.GetDefault();

if (controller != null)

{

\_pin = controller.OpenPin(LED\_PIN);

\_pin.SetDriveMode(GpioPinDriveMode.Output);

\_pin.Write(GpioPinValue.Low);

}

else

{

System.Diagnostics.Debug.WriteLine("Target device has no GPIO controller");

}

}

1. At this point, we have the pin opened and set to the right mode. The final step is to actually toggle the pin's state when the toggle button is pressed. Add the following event handler code to the same code-behind file.

private void ToggleLed\_Checked(object sender, RoutedEventArgs e)

{

if (\_pin != null)

\_pin.Write(GpioPinValue.High);

}

private void ToggleLed\_Unchecked(object sender, RoutedEventArgs e)

{

if (\_pin != null)

\_pin.Write(GpioPinValue.Low);

}

1. The code is complete. Now follow the same deployment steps you used in the previous exercise to deploy the app to the Raspberry Pi. (First, set the target to **ARM**, then select the **Remote Machine**).
2. Again from the "Windows IoT Remote Client" and using the mouse, click the button on and off and look at the red LED on the board. If you want to see the debugging in action, place a breakpoint in both of the event handlers and step through when you click the button.