INTERNET OF THINGS DEN LAB GUIDE

Windows 10 IoT Core

Internet of Things Den with FEZ HAT Lab Guide

Document Version 3.1 Lite

This Lab assumes: -

1) An Azure account and an IoT Hub have been provisioned and you have been provided with Lab Supplement data
2) Alternatively, you followed the notes in the Windows IoT Core Lab Setup Guide and provision your own Azure account and IoT Hub and created your own Lab Supplement data.

Social	**Twitter** #iotden #iot #raspberrypi #windows10 #azure
Document Authors	Dave Glover | dglover@microsoft.com | @dglover Andrew Coates |acoat@microsoft.com |@coatsy Fai Lai |hoongfai@microsoft.com |@faister
Document location	https://github.com/MakerDen/Maker-Den-Documentation-and-Resources-FezHat
Source Code Location	https://github.com/MakerDen/Maker-Den-Windows-IoT-Core-FEZ-HAT
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

The goal of the IoT 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 deploying code, streaming sensor data to Microsoft Azure, aggregating data with Stream Analytics and reporting with Microsoft Power Bi.

GETTING STARTED

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

TIME REQUIRED

There are two sections to this lab. The first section is device centric and will take less than 15 minutes. Section 2 and beyond are more cloud centric and will take approximately an hour. You are more than welcome to stay longer and delve a little deeper.

SPREAD THE WORD

Be sure to spread the word about the IoT Den on Twitter. Use hash tags #iotden #iot #raspberrypi #windows10 #azure

LAB HARDWARE

The following components are used for the IoT Den.

[Raspberry Pi 2]

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.

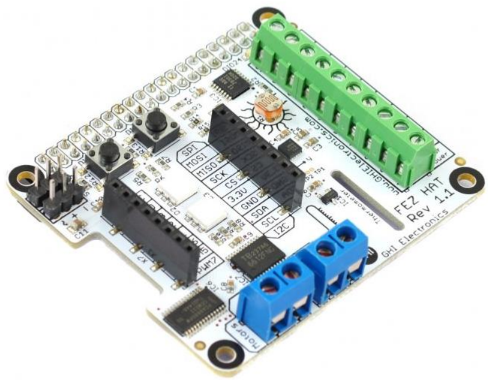
[GHI electronics FEZ HAT]

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.

Developer Guide

https://www.ghielectronics.com/docs/329/fez-hat-developers-guide



RESET THE LAB

⌦ **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.**

EXPERIMENTS

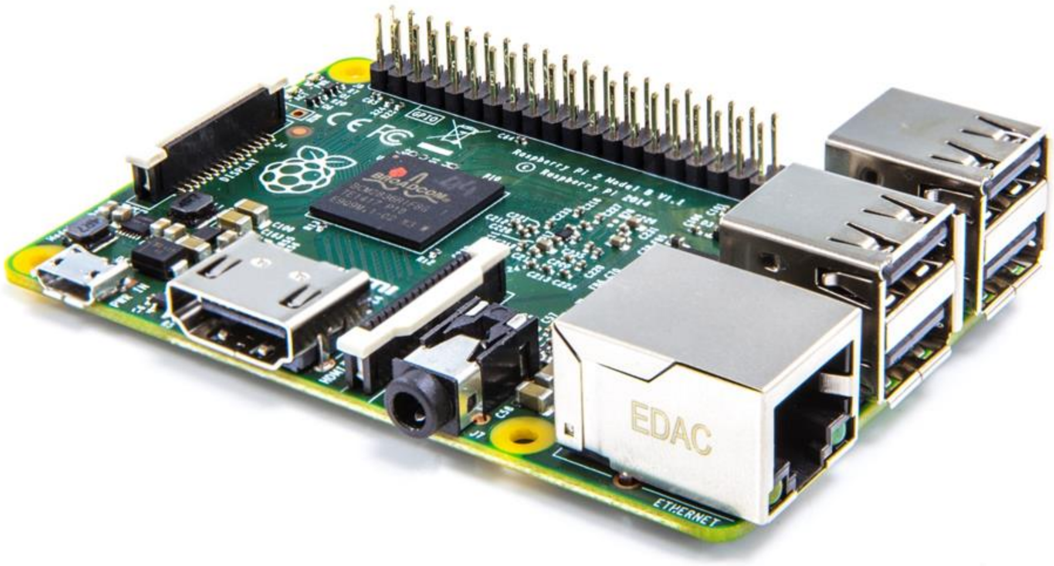
⌦ All the source code can be referenced from the Source Code folder on the Desktop.

⌦ This user guide can be found in the Documents folder on the Desktop.

⌦ Be sure to check out the Windows 10 IoT Core Doc, Tutorials and Samples. There is a link to this page in the Desktop Documents folder.

⌦ For the self-sufficient adventurous types, you can reference the Windows 10 IoT Core Doc, Tutorials and Samples and the GHI Electronics FEZ HAT developer resources for more information.

Section 1



Windows IoT Core development with Visual Studio

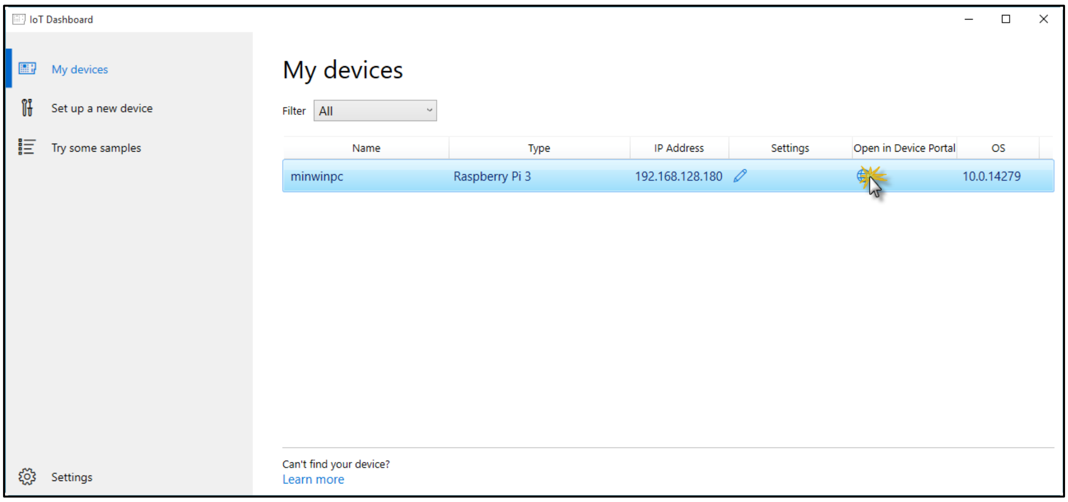
Connecting your device

EXPERIMENT 1: CONNECTING AND CONFIGURING YOUR DEVICE

The Raspberry Pi should be connected to the development PC via 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 is enabled on the PC.

☒ **STEP 1:** Press the Windows key and type "Windows 10 IoT Core Dashboard"[1] and run.

☒ **STEP 2:** Go to My devices[2] and click the Open in Device Portal icon for your chosen device.



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.

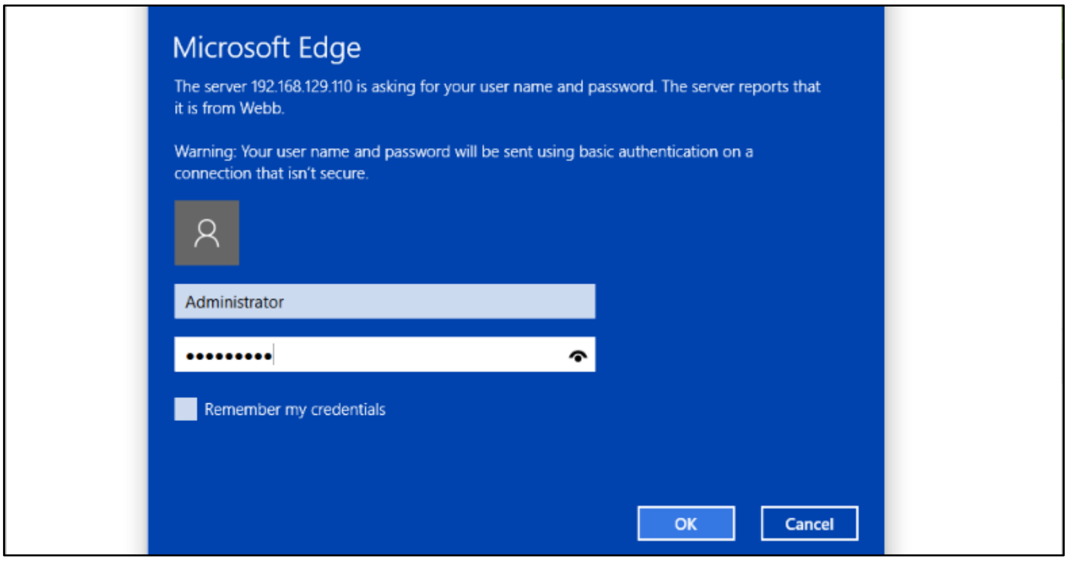
Alternatively, navigate to the default device url http://minwinpc:8080.

---

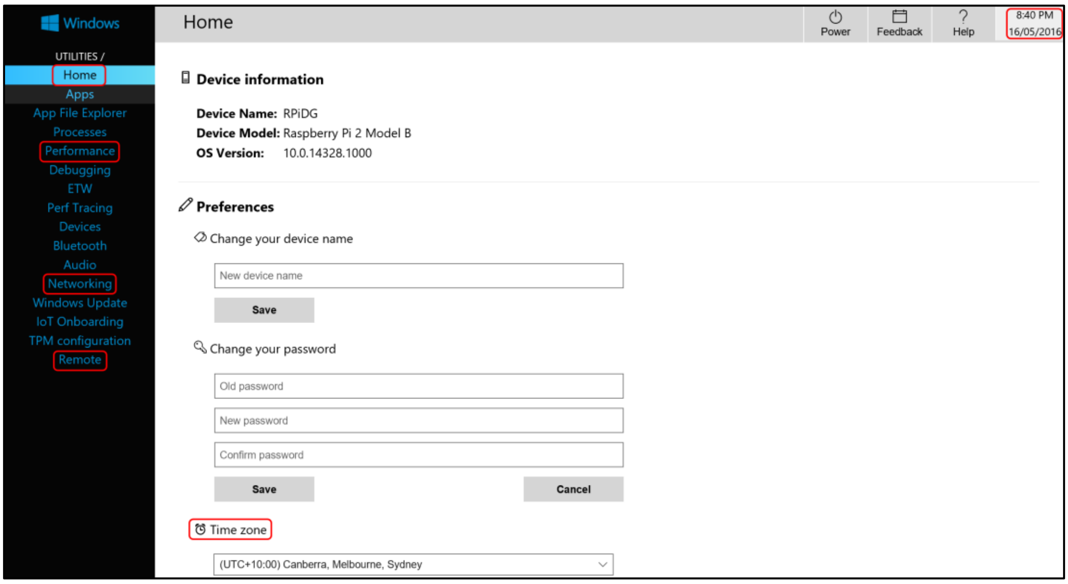
[1] You can download the Windows 10 IoT Core Dashboard from https://developer.microsoft.com/en-us/windows/iot/getstarted

[2] You can right mouse click a device for more options including copying the device IP Address, Name, and to start a PowerShell session.

⌧ **STEP 3:** Authenticate. The default credentials are Username: *Administrator* and Password: *p@ssw0rd*



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



⌧ **STEP 4:** Verify Device Configuration

- From the **Home** Tab verify the time zone, date and time are correct. If the device has the incorrect date or time, then refer to the troubleshooting section in the appendix.

- From the **Remote** tab verify that **Windows IoT Remote Server**[3] is enabled. If it is not, then enable it.
- Take a moment to explore the other tabs in the Windows Device Portal.

⌦**STEP 5:** Test Windows IoT Remote Client connection.

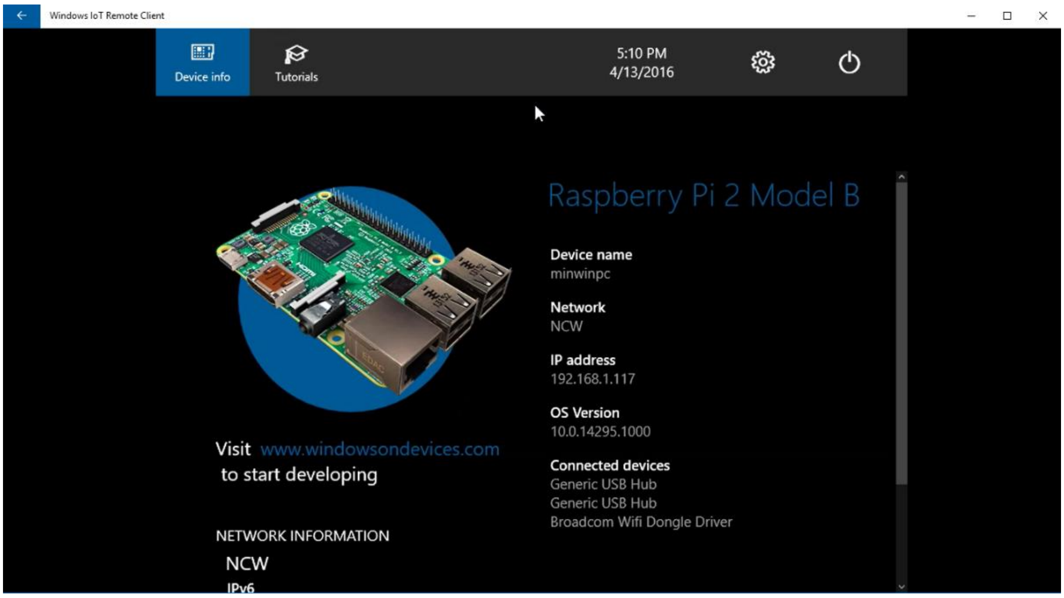
Press the Windows key and type "Windows IoT Remote Client" [4] and run.

⌦**STEP 6:** Select your device from the dropdown list.

Depending on the network setup you may 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.

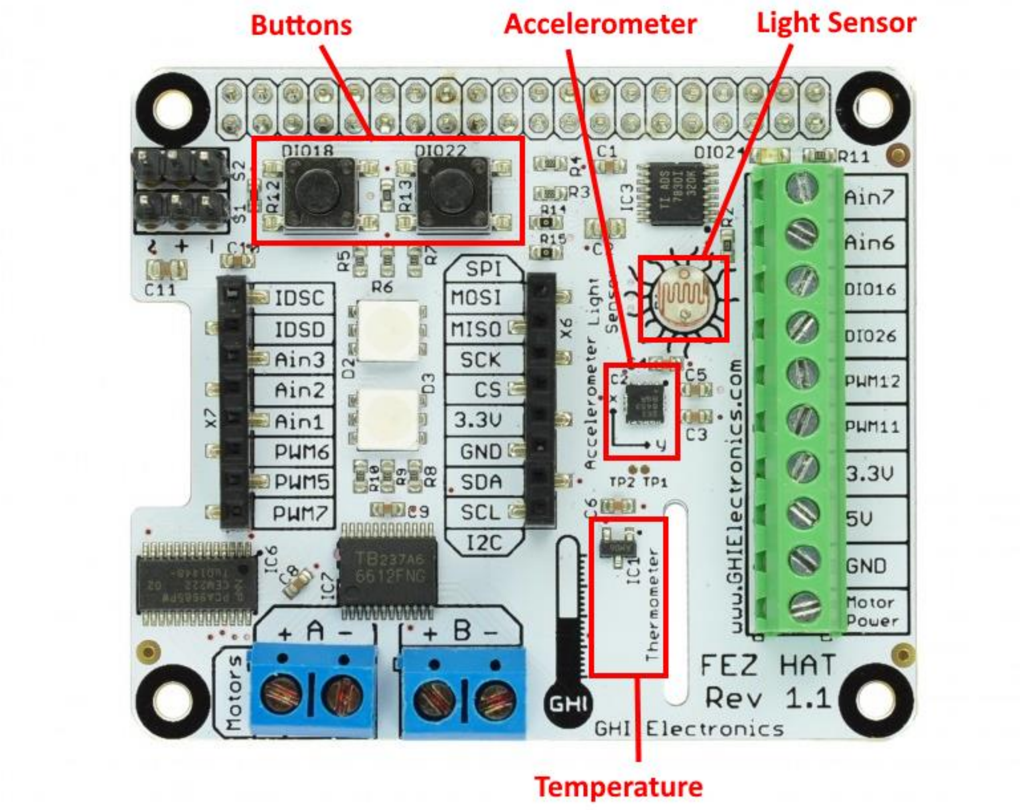
Minimise the remote client application when you have verified that it is working.



[3] The Windows IoT Remote Server does take additional CPU cycles on the Raspberry Pi so depending on what you are doing you may want to disable the Windows Remote Server from the Windows Device Portal.

[4] The Windows IoT Remote Client is available from the Windows Store.

Section 2



Windows IoT Core development with Visual Studio

Deploying your first "**Headless**" app

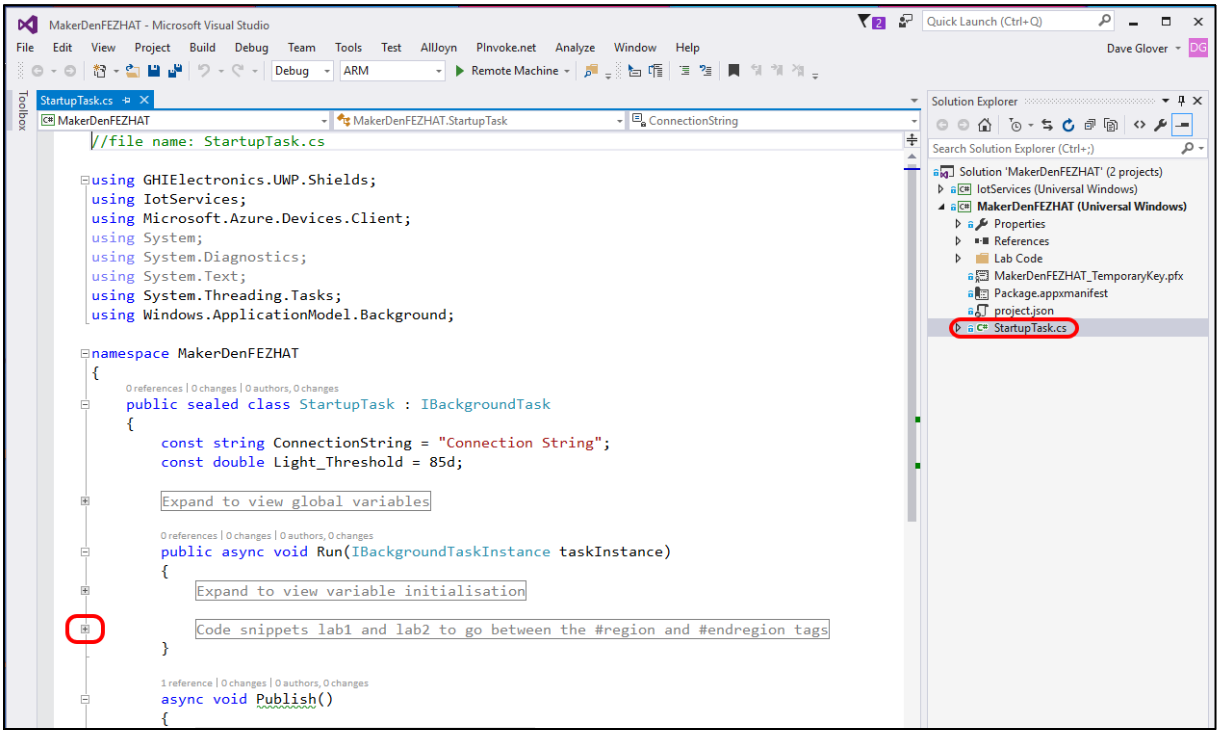
Sensing ambient light levels

Debugging

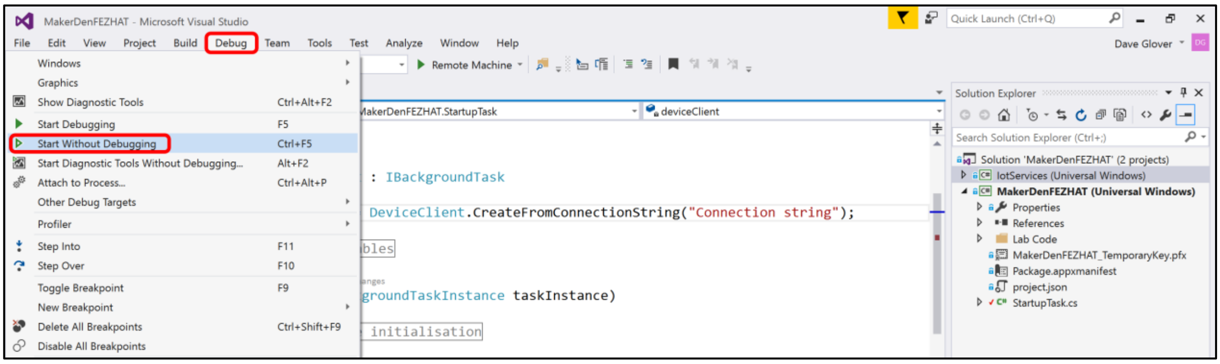
EXPERIMENT 2: HELLO WORLD

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

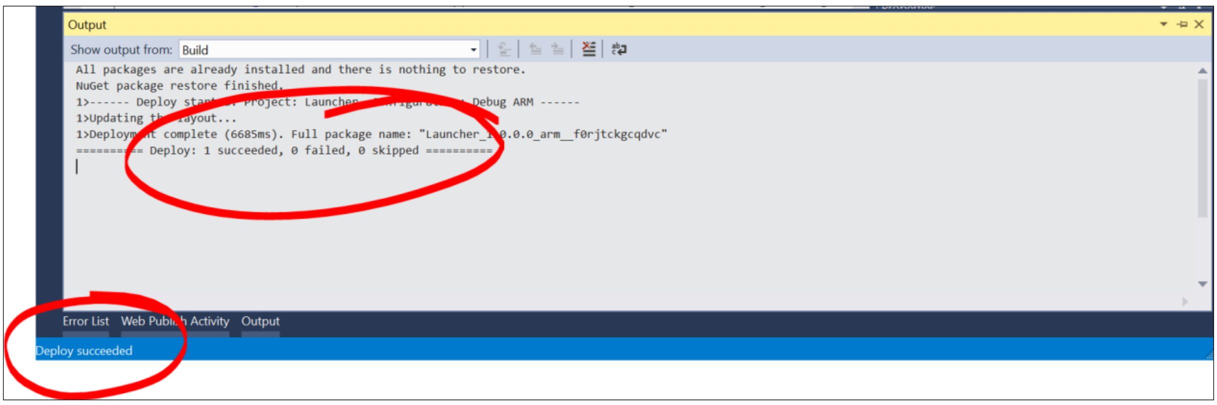
⌧ **STEP 1**: Expand the **MakerDen** project then double click the **StartupTask.cs** file to open it. You may need to expand the code regions by clicking the highlighted **+** symbol on the left hand side.



⌧ **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. It will take approximately 30 to 60 seconds to deploy.



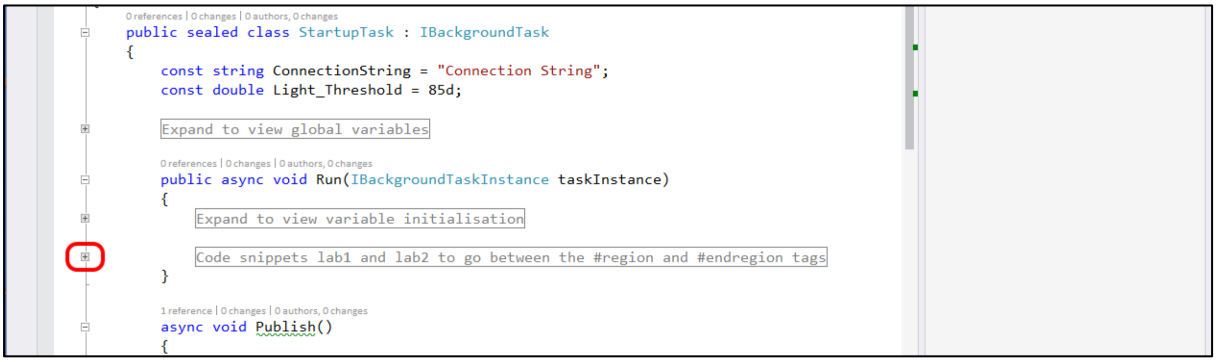
⌧ **STEP 4:** Check the LEDs on the FEZ HAT. You should see an LED alternating between Red, Green and Blue.

⌧ **STEP 5:** Pat yourself on the back, you did it☺

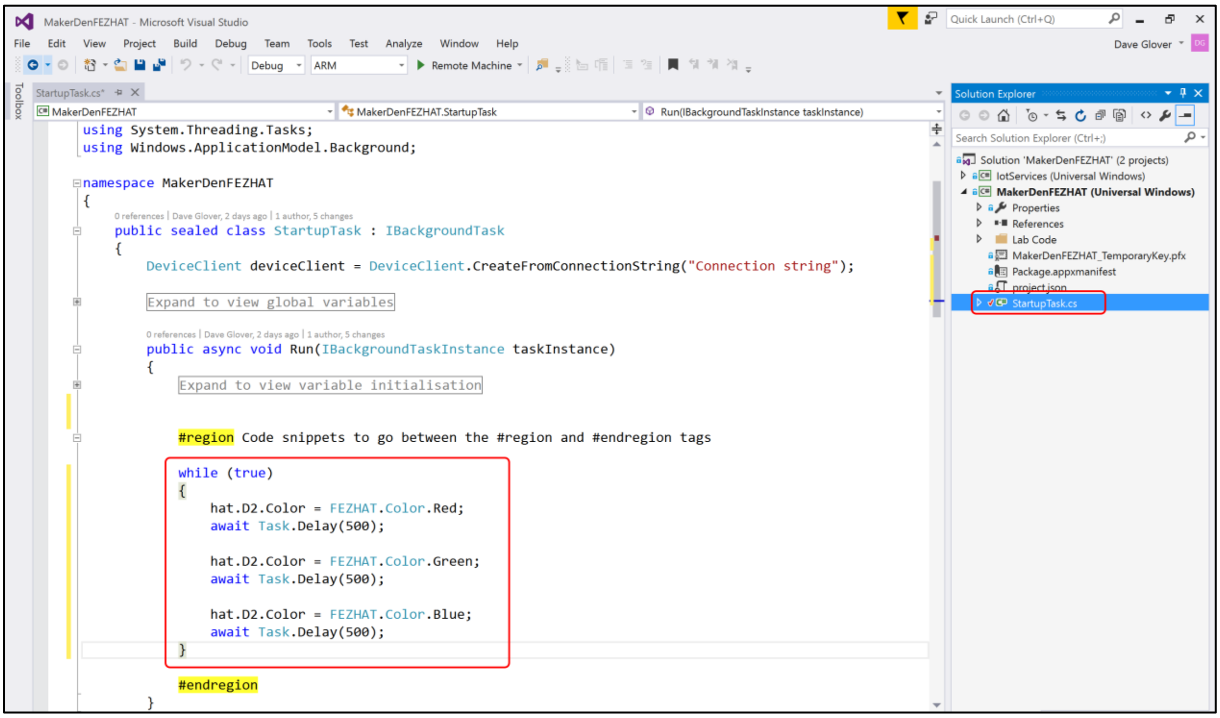
EXPERIMENT 3: SENSING THE WORLD

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

☒**STEP 1:** Review the code in the **StartupTask.cs** file. Look for the **#region** and **#endregion** tags. You may need to expand the code regions by clicking the highlighted **+** symbol on the left hand side.



☒**STEP 2:** Delete the code circled in red **inside** the #region tags.



⌦**STEP 3:** Ensure the cursor is between the **#region** tags, then a code snippet type **lab3** and press Tab twice **OR** type the following code.

```
while (true)
{
    var level = hat.GetLightLevel() * 100;

    if (level > Light_Threshold)
    {
        hat.D2.Color = FEZHAT.Color.Blue;
    }
    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.

```csharp
//file name: StartupTask.cs

using GHIElectronics.UWP.Shields;
using IotServices;
using Microsoft.Azure.Devices.Client;
using System;
using System.Diagnostics;
using System.Text;
using System.Threading.Tasks;
using Windows.ApplicationModel.Background;

namespace MakerDenFEZHAT
{
    public sealed class StartupTask : IBackgroundTask
    {
        const string ConnectionString = "Connection String";
        const double Light_Threshold = 85d;

        Expand to view global variables

        public async void Run(IBackgroundTaskInstance taskInstance)
        {
            Expand to view variable initialisation

            #region Code snippets lab2 and lab3 to go between the #region and #endregion tags

            while (true)
            {
                var level = hat.GetLightLevel() * 100;

                if (level > Light_Threshold)
                {
                    hat.D2.Color = FEZHAT.Color.Blue;
                }
                else
                {
                    hat.D2.Color = FEZHAT.Color.Red;
                }

                await Task.Delay(500);
            }

            #endregion
        }

        async void Publish()
        {
            #region Snippet lab6 - Publish to Azure IoT Hub

            #endregion
        }

        private void Commanding_CommandReceived(object sender, CommandEventArgs<string> e)
        {
            #region Snippet lab9 - IoT Hub Command Support

            #endregion
        }
    }
}
```

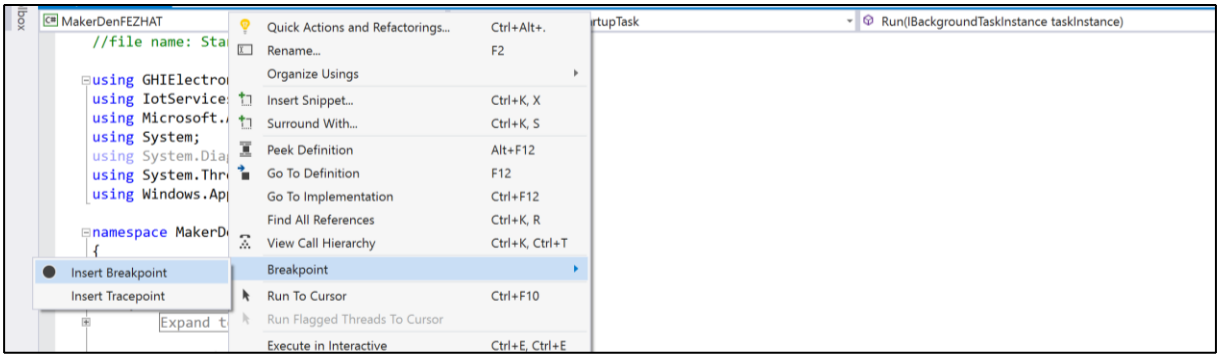
☒ **STEP 5:** Deploy the solution to the Raspberry Pi. From the **Debug** menu select **Start Without Debugging** or from the keyboard press **Ctrl+F5.**

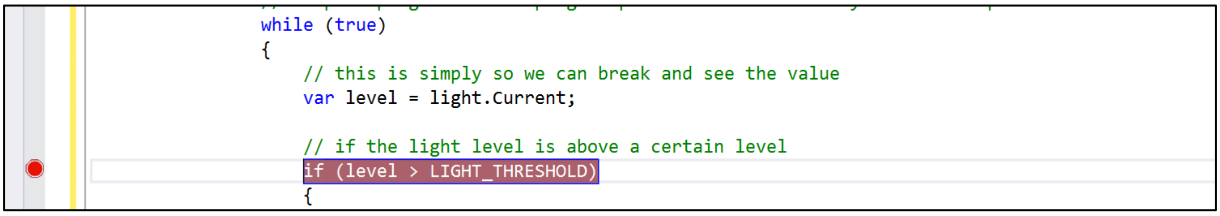
☒ **STEP 6:** Hover your hand over the light sensor and observe the LED alternate between blue and red depending on the ambient light levels.

EXPERIMENT 4: 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 and Windows IoT Core.

- Right-click on the line that **reads** `if (level > Light_Threshold)`
- Choose Breakpoint, then Insert Breakpoint.





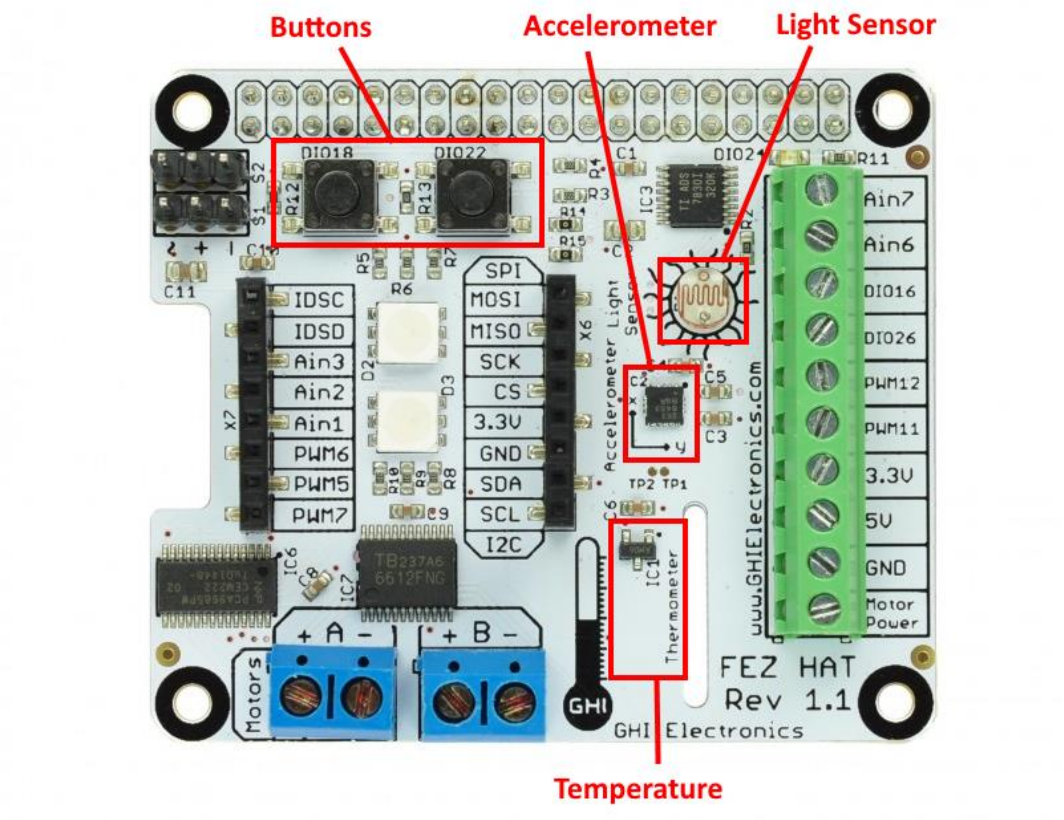
⌦ **STEP 2:** From the **Debug** menu select **Start Debugging** or on the keyboard press **F5** and wait for the solution to deploy and for Visual Studio to hit the breakpoint.

⌦ **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 the LED change colour depending on ambient light levels.

⌦ **Step 5**: Press Shift-F5 to stop debugging.

Section 3 Optional



Windows IoT Core development with Visual Studio

Deploying your first “**Headed**” app

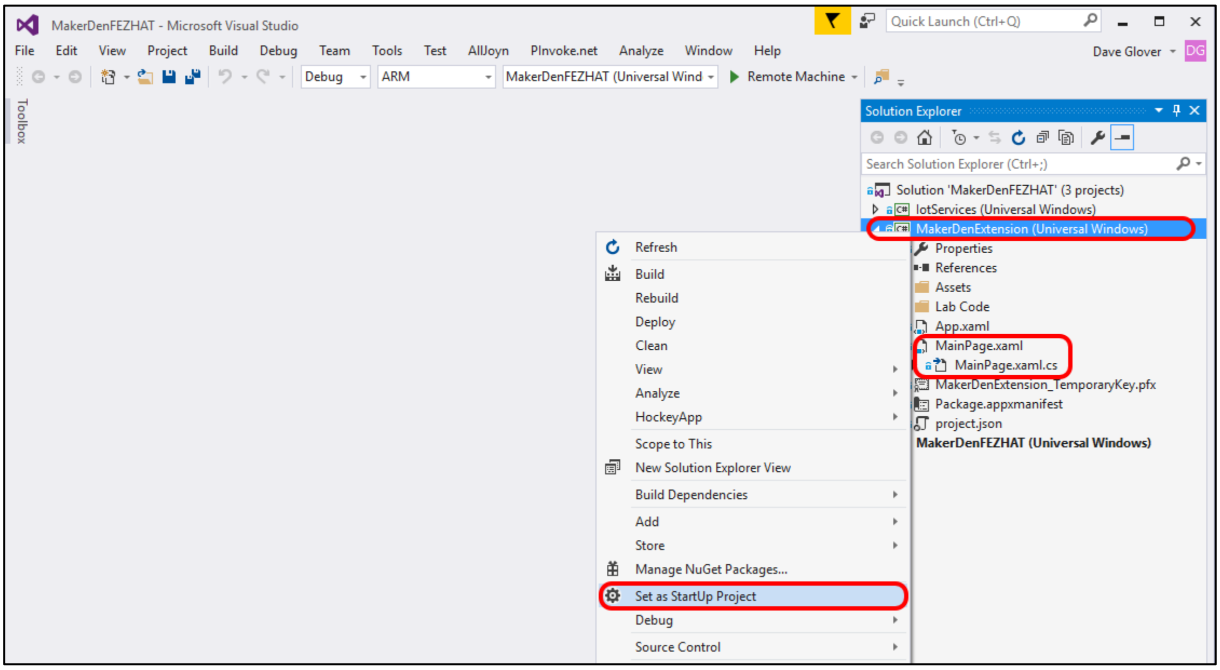
Interacting with Sensors

EXPERIMENT 5 EXPLORING APPS AND SENSORS

There are two styles of applications you can run on Windows IoT Core – Headed and Headless.

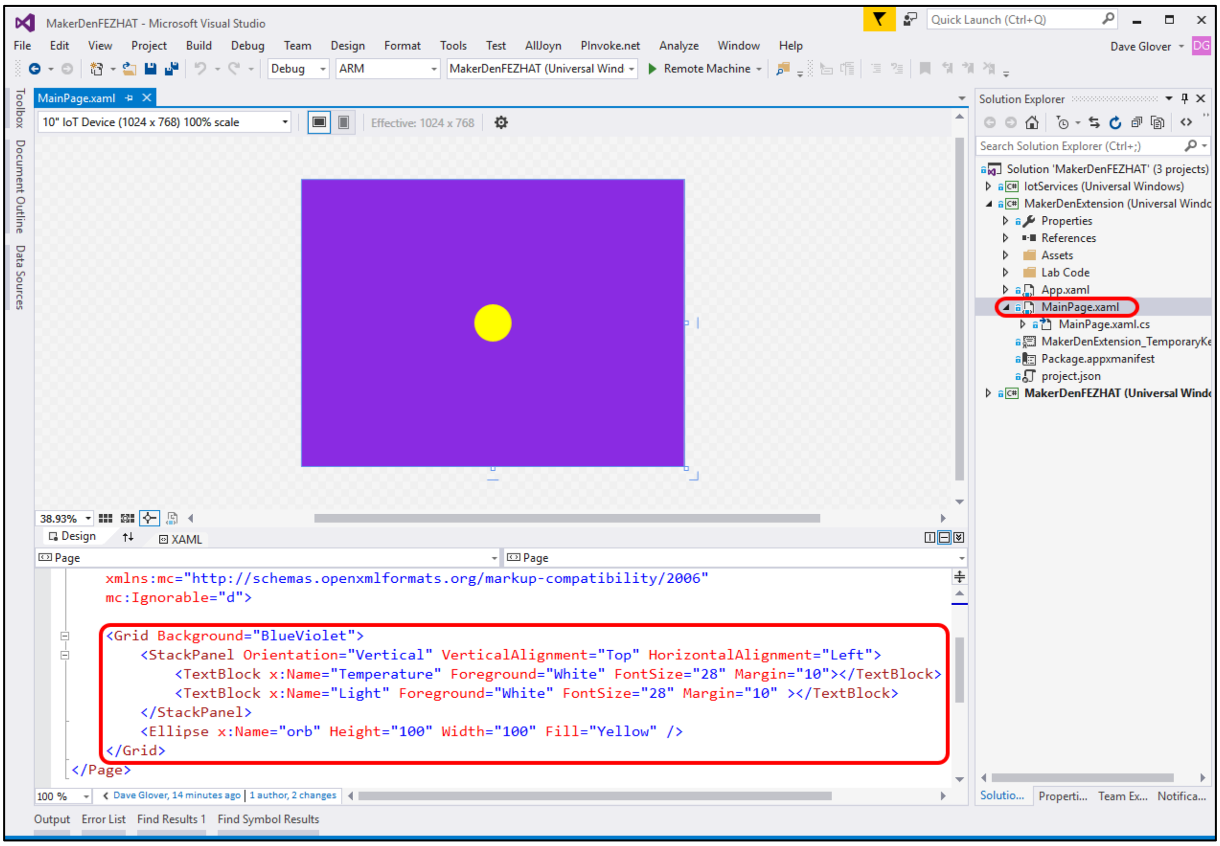
This section focuses on “Headed” apps. These are apps with a user interface that can display output on a screen (or via the Windows IoT Remote Client) and can use a mouse and keyboard. The app runs in the foreground and only one “Headed” app can be active at any one time.

⌦ **STEP 1:** Set the **MakerDenExtension** as the Startup Project. Right mouse click **MakerDenExtension** in the Solution Explorer and select “**Set as Startup Project**”



☒ **STEP 2:** Double click on **MainPage.xaml** to open the file. It will take a moment to load.

☒ **STEP 3:** Review the XAML markup that describes the User Interface, the MainPage.xaml page should look like the following.



⌦ **STEP 4:** Double click on **MainPage.xaml.cs** to open the file. Locate the **Setup** method in the MainPage.xaml.cs file. Between the **#region Lab4b** tags type **lab4b** and press Tab twice **OR** type the following code.

```
private async void Setup()
{
    #region Lab4b Code to go between the #region and #endregion tags

    myTransformGroup.Children.Add(myTranslate);
    orb.RenderTransform = myTransformGroup;

    this.hat = await FEZHAT.CreateAsync();

    timer = new DispatcherTimer();
    this.timer.Interval = TimeSpan.FromMilliseconds(100);
    this.timer.Tick += this.UpdateOrb;
    this.timer.Start();

    #endregion
}
```

⌦ **STEP 5:** Locate the **UpdateOrb** method in the MainPage.xaml.cs file. Ensure the cursor is between the **#region Lab4c and Lab4d Code** tags, then using a code snippet type **lab4c** and press Tab twice **OR** type the following code.

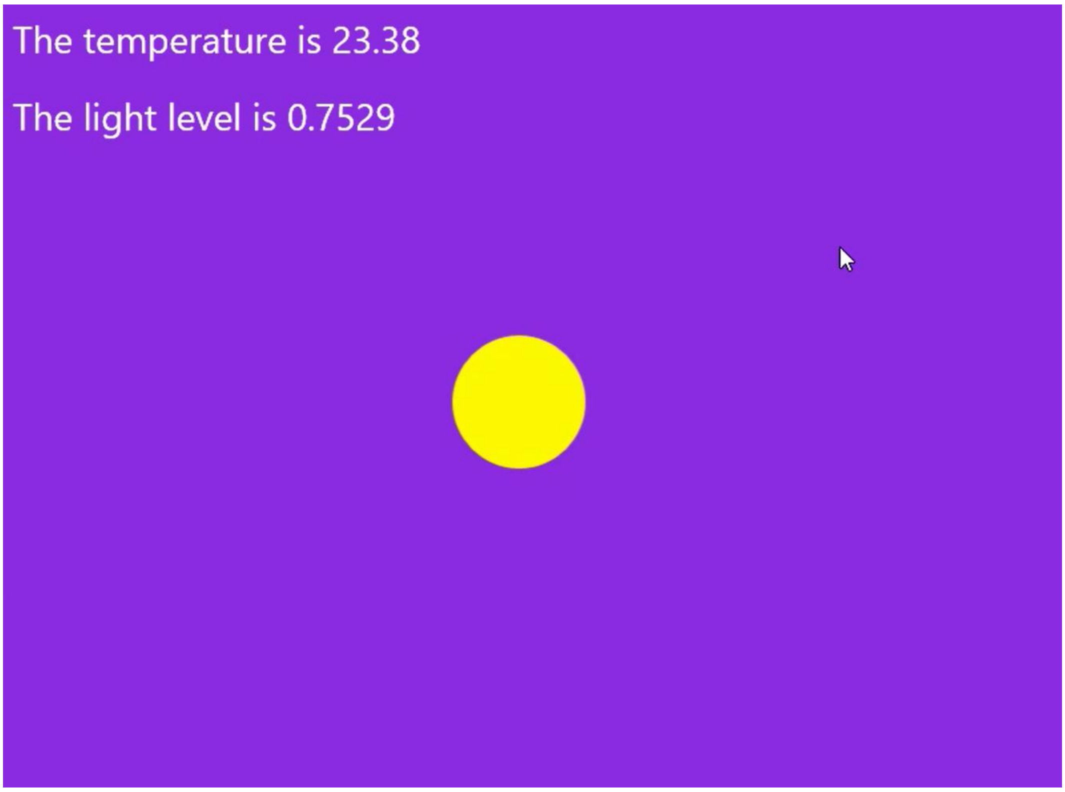
```
Temperature.Text = string.Format("The temperature is {0:N2}", hat.GetTemperature());
Light.Text = string.Format("The light level is {0:N4}", hat.GetLightLevel());

if (hat.IsDIO18Pressed())
{
    computedColour = Colors.DeepPink;
}
if (hat.IsDIO22Pressed())
{
    computedColour = Colors.Lime;
}

orb.Fill = new SolidColorBrush(computedColour);

orb.UpdateLayout(); // update the orb with the new colour and position
```

⌦ **STEP 6:** Deploy the solution to the Raspberry Pi. From the **Debug** menu select **Start Without Debugging** or from the keyboard press **Ctrl+F5.**

⌧**STEP 7: Maximise** the Windows IoT Remote Client from the Windows Taskbar.

You should have minimised the Windows IoT remote Client after you verified that it was working in the first section. But in case you closed it, then press the Windows key and type "Windows IoT Remote Client" and run. Then select your device from the dropdown list.

When your application has started on the Raspberry Pi it should look like the following image.



⌧ **STEP 9:** Press the buttons on the Fez HAT and observe the orb colour changes then hover your hand over the Raspberry Pi and observe the light level value changes.

⌧ **STEP 10:** For a bit of fun, modify the **UpdateOrb** method and type the highlighted line as below. This will change the brightness of the orb on the screen based on the ambient light levels.

```
if (hat.IsDIO18Pressed())
{
    computedColour = Colors.DeepPink;
}
if (hat.IsDIO22Pressed())
{
    computedColour = Colors.Lime;
}

computedColour.A = (byte)(255 * hat.GetLightLevel());  // change the orb brightness

orb.Fill = new SolidColorBrush(computedColour);

orb.UpdateLayout(); // update the orb with the new colour and position
```

⌧ **STEP 11:** Deploy the solution to the Raspberry Pi. From the **Debug** menu select **Start Without Debugging** or from the keyboard press **Ctrl+F5.**

⌧ **STEP 12:** Maximise the Windows IoT Remote Client and observe that when you place your hand over the Raspberry Pi the brightness of the org will changed based on the ambient light levels.

EXPERIMENT 6: WHICH WAY IS UP WITH THE FEZ HAT ACCELEROMETER

⌦ **STEP 1:** Modify the **UpdateOrb** method so the colour and position of the orb change based on data from the accelerometer built into the Fez HAT.

Remove all the code between the **#Region Lab4c and Lab4d** Code tags. Then using a code snippet type **lab4d** and press Tab twice.

```
private void UpdateOrb(object sender, object e)
{
    #region Lab4c and Lab4d Code to go between the #region and #endregion tags

    Temperature.Text = string.Format("The temperature is {0:N2}", hat.GetTemperature());
    Light.Text = string.Format("The light level is {0:N4}", hat.GetLightLevel());

    hat.GetAcceleration(out x, out y, out z);

    computedColour = ComputeColour(x, y, z);

    orb.Fill = new SolidColorBrush(computedColour);

    ComputeOrbPosition(x, y);

    orb.UpdateLayout(); // update the orb with the new colour and position

    #endregion
}
```

⌦ **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:** Maximise the Windows IoT Remote Client. Gently pick up the Raspberry Pi and tilt it backwards and forwards, left and right and observer the colour and position of the orb changes.

Section 4

Microsoft Azure Cloud Development

Registering a device with IoT Hub

Secure Data Streaming

Secure Command and Control

EXPERIMENT 5: REGISTERING YOUR DEVICE WITH AZURE IOT HUB

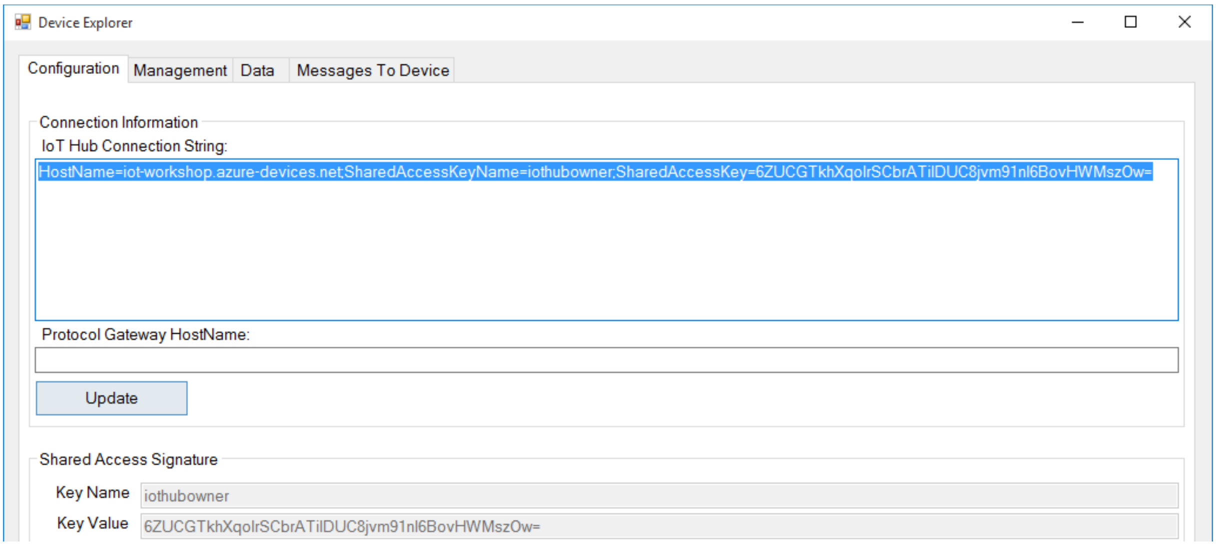
This experiment assumes that you have either been provided with or created your own Azure account and IoT Hub Service. See “[Windows IoT Core Lab Setup Guide]” in the Lab GitHub repository.

You must register your device in order to be able to send and receive information from the Azure IoT Hub. This is done by registering a Device Identity in the IoT Hub.

⌧ **STEP 1:** Press the Windows key and type “Device Explorer” [5] and run the app.

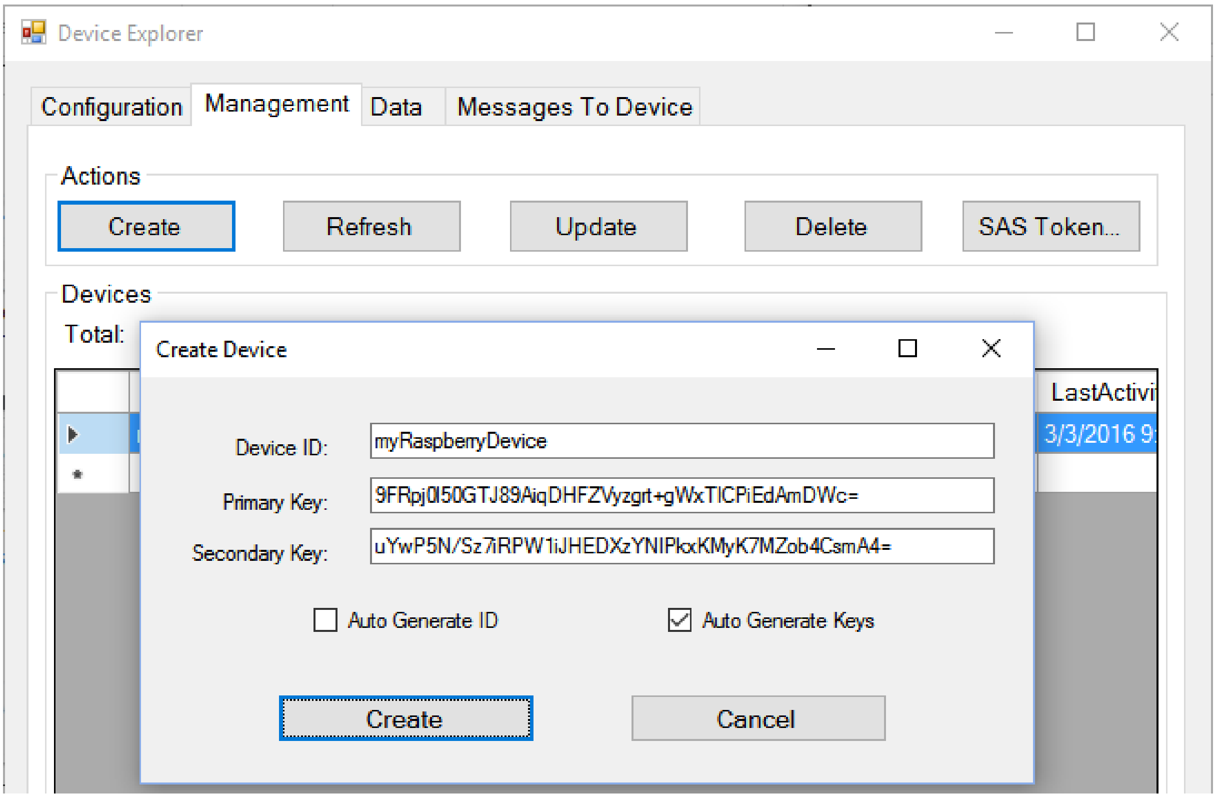
If “Device Explorer” is not installed, then install it from https://github.com/Azure/azure-iot-sdks/releases (Scroll down for SetupDeviceExplorer.msi).

⌧ **STEP 2:** Paste the **IoT Hub Connection String** provided to you in the Lab Supplement into the IoT Connection String field and click **Update**.



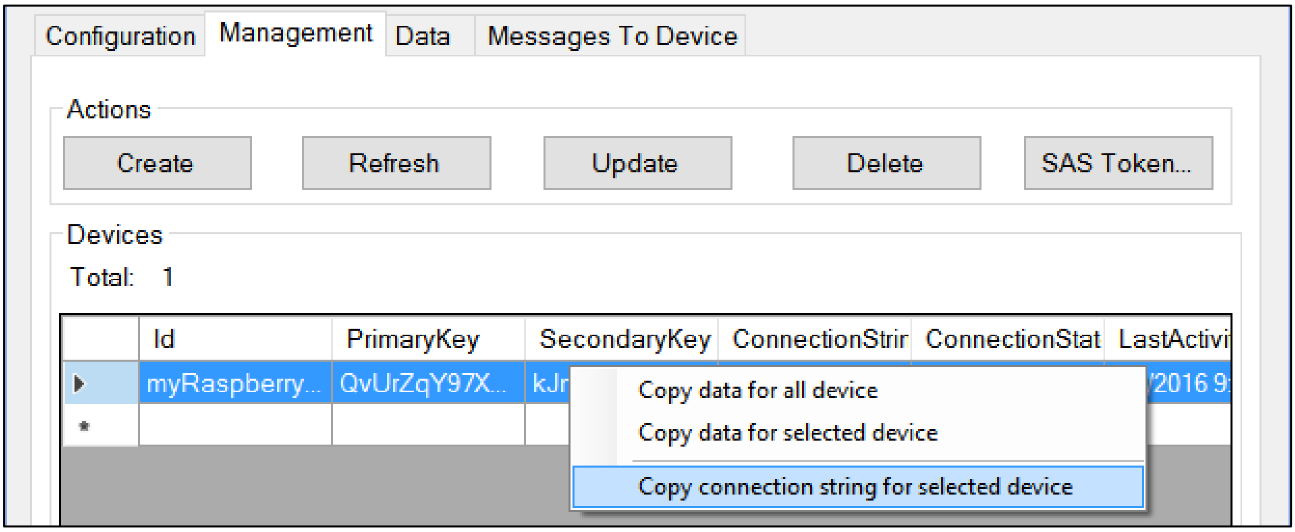
⌧ **STEP 3:** Go to the **Management** tab and click on the **Create** button. The Create Device popup will be displayed. Fill the **Device ID** field with a new id for your device. For example, MyRPi01, then click on Create.

[5]The Device Explorer is an Open Source sample. In production you would integrate device provisioning into your solution. See Get started with Azure IoT Hub for .NET.



⌧**STEP 4:** Once the device identity is created, it will be displayed in the grid. Right click on the identity you just created, select **Copy connection string for selected device**, the connection string will be copied to the clipboard.

This unique connection string allows a device to authenticate and communicate securely with Azure IoT Hub.



EXPERIEMENT 6: STREAMING TELEMETRY DATA TO AZURE IOT HUB

⌦ **STEP 1:** Return to Visual Studio

⌦ **STEP 2:** Ensure debugging has stopped (Shift-F5) otherwise you will not be able to make any edits to the StartUpTask.cs file.

⌦ **STEP 3:** Paste the Connection String you previously copied from the Device Explorer into the highlighted area in the StartUpTask.cs file.

```
namespace MakerDenFEZHAT
{
    public sealed class StartupTask : IBackgroundTask
    {
        const string ConnectionString = "Connection String";
        const double Light_Threshold = 85d;

        #region Expand to view global variables

        public async void Run(IBackgroundTaskInstance taskInstance)
```

⌦ **STEP 4:** Locate the **Publish** method in the StartupTask.cs file. Between the **#region Lab 6** tags type **lab6** and press Tab twice **OR** type the following code.

```
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() * 100;  // 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(); }
```

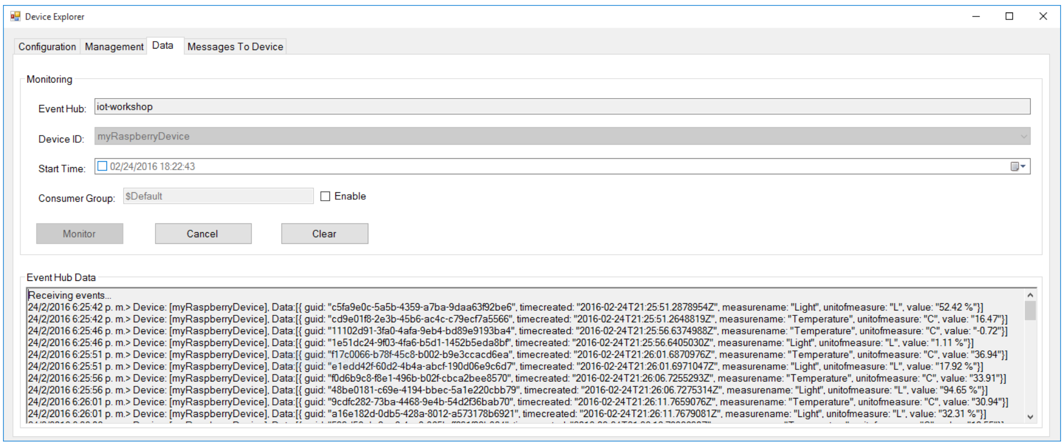
The Publish method is responsible for reading sensors and streaming data to Azure IoT Hub. The Publish method is called by the Telemetry class every 10 seconds.

⌧**STEP 5:** Your completed Publish method should look like this.

```
async void Publish()
{
    #region Snippet lab6 - Publish to Azure IoT Hub

    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() * 100;  // 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 6:** 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 7:** Press the Windows key and type “Device Explorer” and run the app. Navigate to the Data tab and select your device from the dropdown and click on Monitor.



EXPERIMENT 6A: CONTROLLING A DEVICE FROM AZURE IOT HUB

Azure IoT Hub is a service that enables reliable and secure bi-directional communications[6] between millions of IoT devices and an application back end.

In this experiment we will send cloud-to-device messages to your device to command it to change the colour of one of the FEZ HAT LEDs. For the experiment Device Explorer will serve as the back end.

⌧ **STEP 1:** Locate the **Command_Processing** method in the StartupTask.cs file. Between the **#region Lab 9** tags type **lab9** and press Tab twice **OR** type the following code.

```
#region Snippet lab9 - IoT Hub Command Support

while (true)
{
    try
    {
        Message receivedMessage = await deviceClient.ReceiveAsync();
        if (receivedMessage == null)
        {
            await Task.Delay(2000);
            continue;
        }

        await deviceClient.CompleteAsync(receivedMessage);

        string command =
Encoding.ASCII.GetString(receivedMessage.GetBytes()).ToUpper();
        if (string.IsNullOrEmpty(command) ||
telemetry.SetSampleRateInSeconds(command)) { continue; }

        switch (command[0])
        {
            case 'R':
                publishColor = FEZHAT.Color.Red;
                break;
            case 'G':
                publishColor = FEZHAT.Color.Green;
                break;
            case 'B':
```

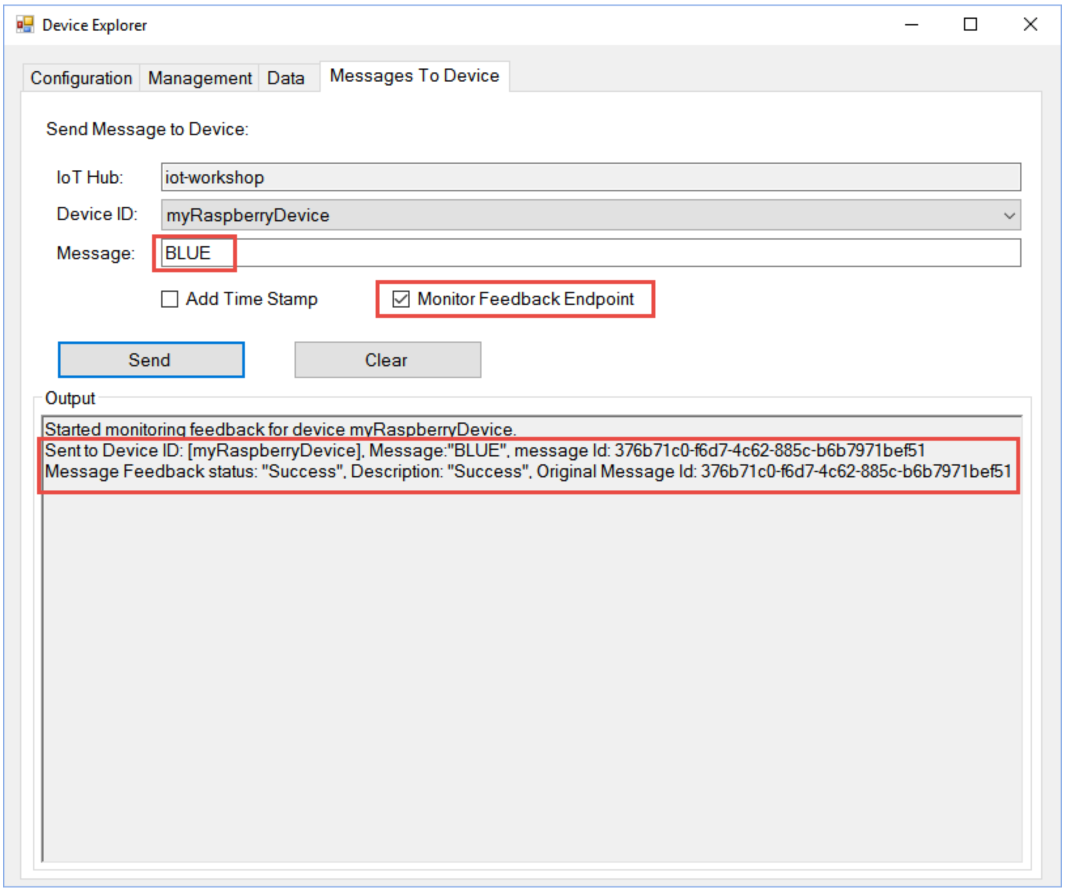
[6] Azure IoT Hub supports a number of protocols including AMQP, HTTPS and MQTT.

```
                publishColor = FEZHAT.Color.Blue;
                break;
            case 'Y':
                publishColor = FEZHAT.Color.Yellow;
                break;
            case 'M':
                publishColor = FEZHAT.Color.Magneta;
                break;
            default:
                Debug.WriteLine("Unrecognized command: {0}", command);
                break;
        }

        hat.D3.Color = publishColor;
    }
    catch { } // just keep going
}

#endregion
```

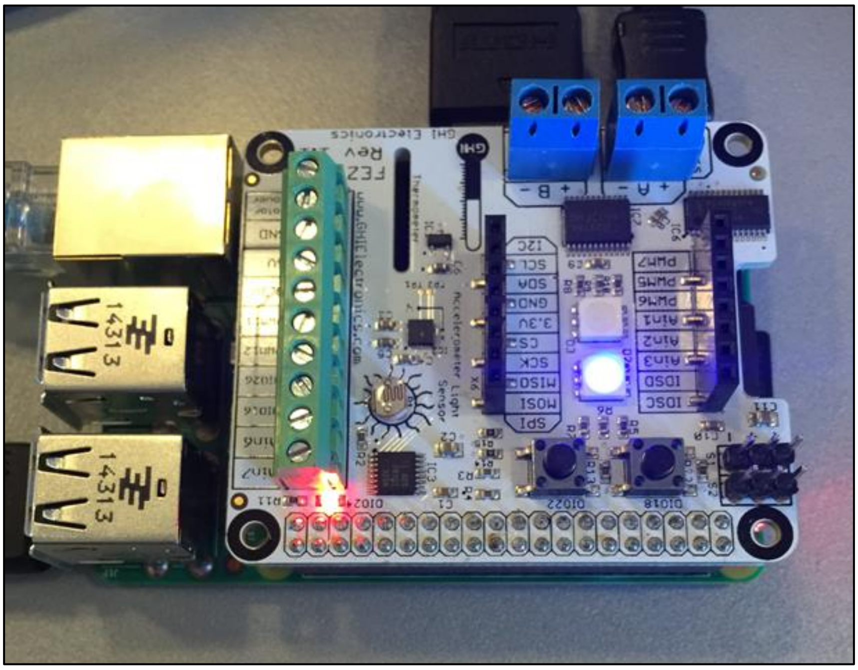
☒ **STEP 2:** Deploy the solution to the Raspberry Pi. From the **Debug** menu select **Start without Debugging** or from the keyboard press **Ctrl+F5** and wait for the solution to deploy.

☒ **STEP 3:** From Device Explorer select the Messages to Device Tab, select your device from the Device ID: dropdown and in the Message field type a colour. Valid colours are Red, Green, Blue or Yellow – or just the first letter of a colour.

☒ **STEP 4:** Enable **Monitor Feedback Endpoint** and click Send.



After a few seconds the message will be processed by the device and the LED will turn on in the colour you selected. The feedback will also be reflected in the Device Explorer screen after a few seconds.



Section 5

Azure Stream Analytics

Gain real-time insights from devices, sensors, infrastructure, and applications

EXPERIMENT 7: INSIGHT AND IOT DATA

You have used the Device Explorer to view data streamed to the Azure IoT Hub. However, there are many ways to gain insight from the data including Stream Analytics, Power Bi, [Azure IoT Suite Remote Monitoring](), and of course your own custom solution.

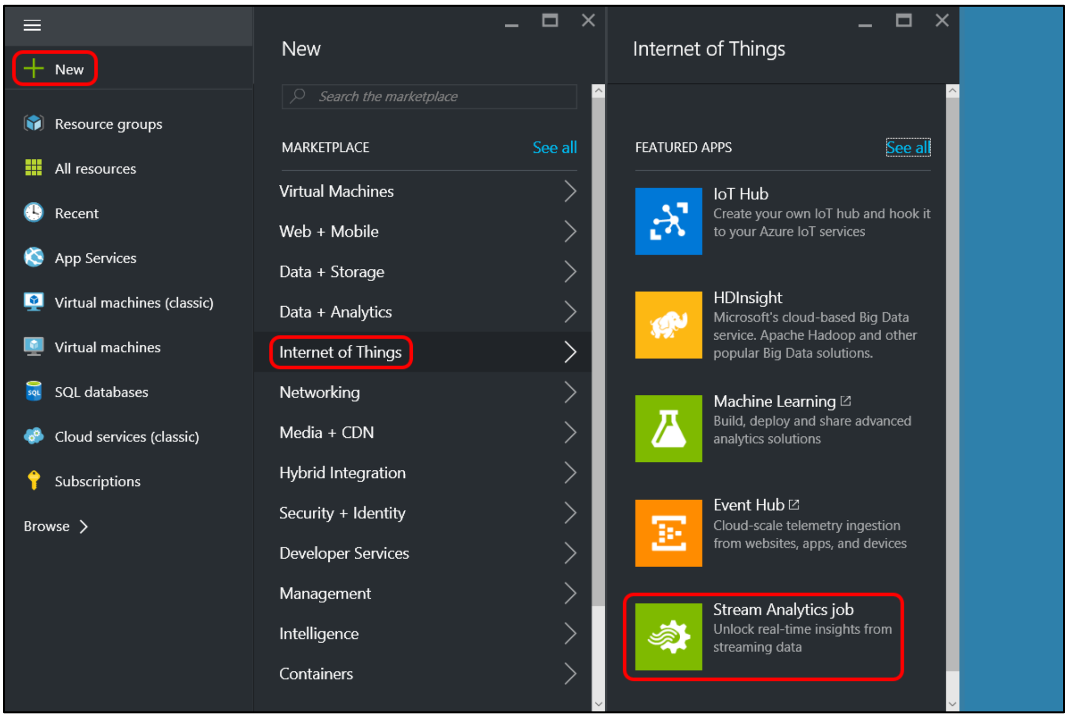
In the following section you will use Azure Stream Analytics in combination with Microsoft Power BI to consume IoT data and to generate meaningful reports.

CREATE A STREAM ANALYTICS JOB

Before the information can be delivered to **Power BI**, it must be processed by a **Stream Analytics Job**. To do so, an input, output and query for the job must be defined. As the Raspberry devices are sending information to an IoT Hub, it will be set as the input for the job.

⌦ **STEP 1:** Login to the Azure Management Portal at http://portal.azure.com. The Azure **email** and **password** credentials are provided in the Lab Supplement.

⌦ **STEP 2:** From the Azure Management Portal, click New -> Internet of Things -> Stream Analytics job.



☒ **STEP 3:** Define the Stream Analytics job then click Create. It will take approximately 30 seconds to create the job.

Complete the form

Job Name

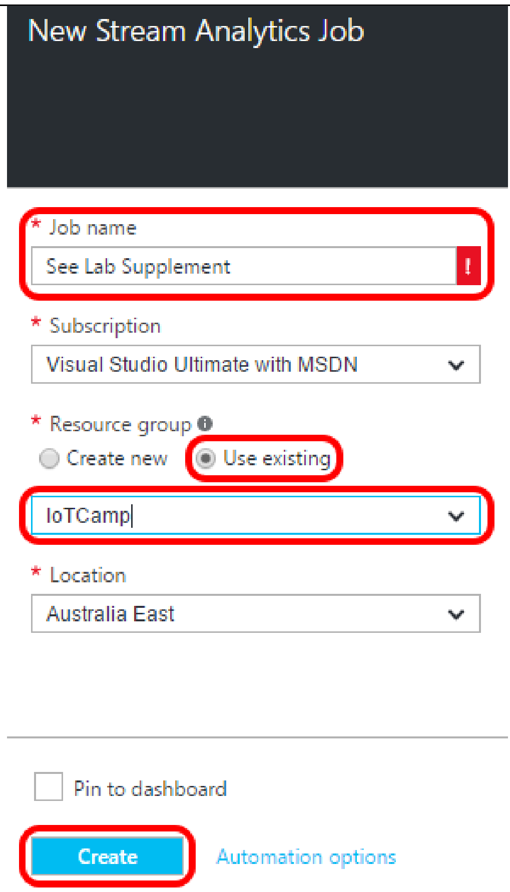
Provided in the Lab Supplement

Resource Group

Use existing IoTCamp

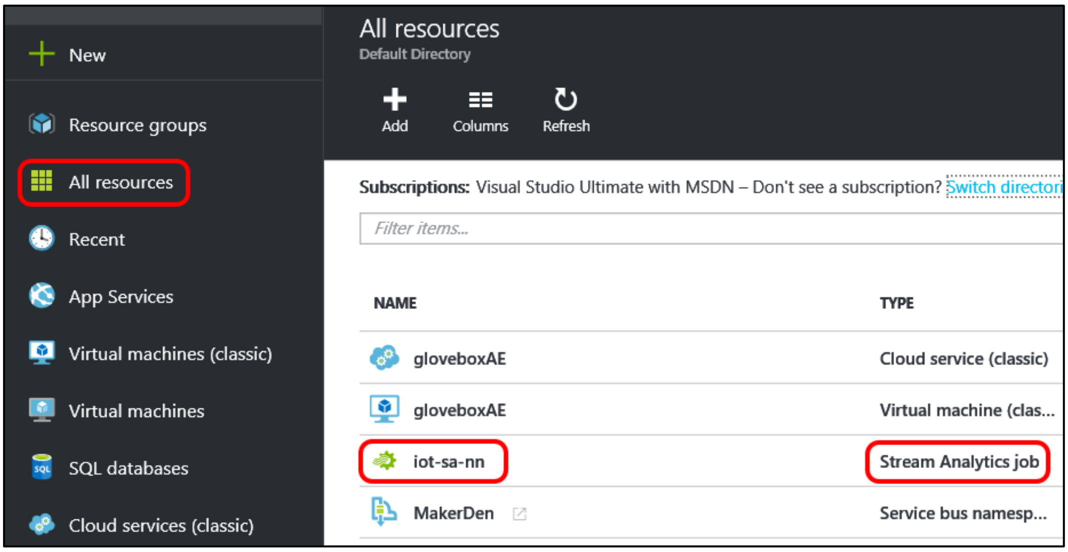
Location

Suggest selecting the region closest to your IoT Hub



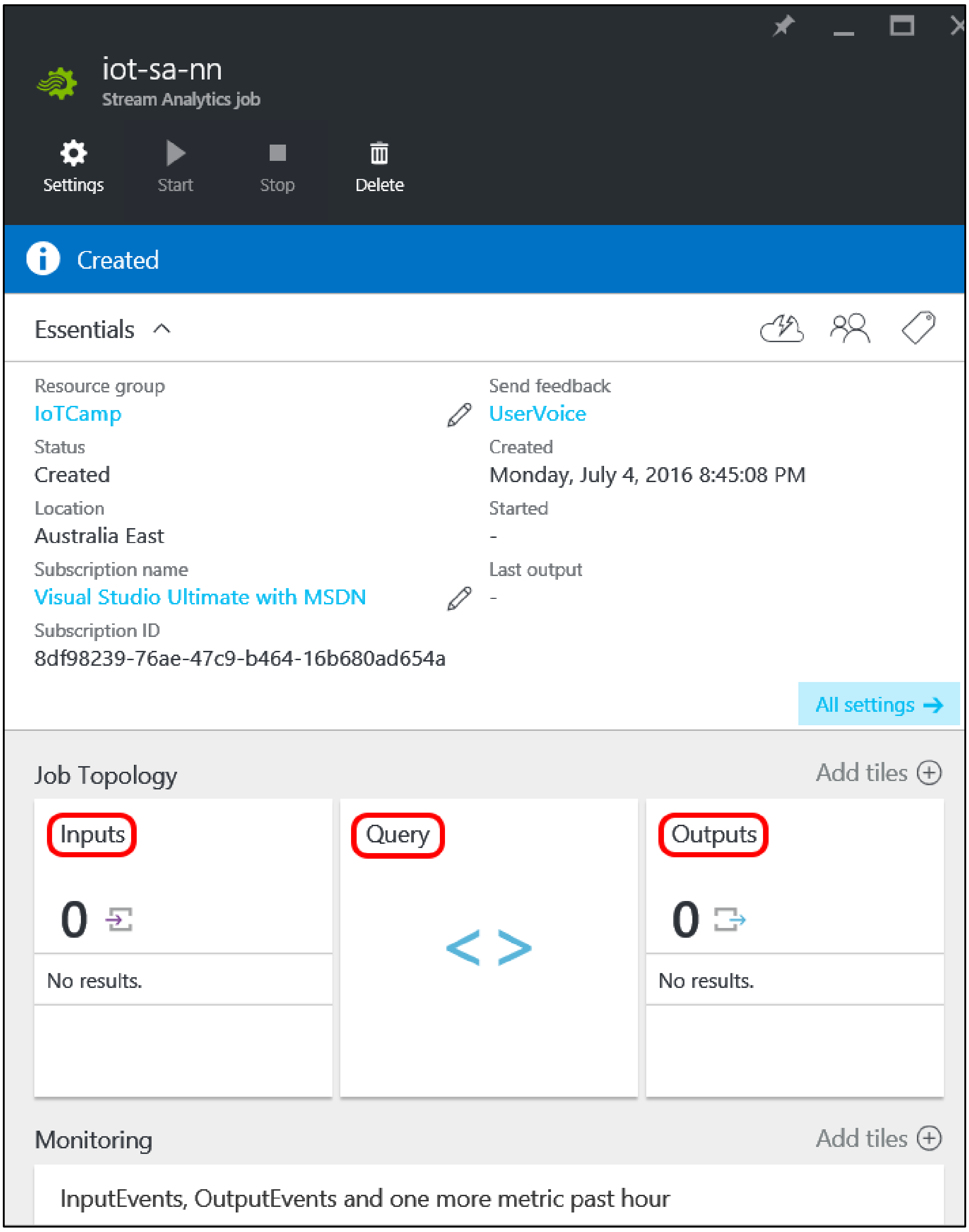
☒ **STEP 4:** Open the newly created Stream Analytics job configuration blade.

Click **All resources** -> **Your newly created Stream Analytics Job.**



☒ **STEP 5:** Familiarise yourself with the Stream Analytics configuration blade. Note the Inputs, Query and Output zones.

As you can see, the Start button is disabled since the job is not configured yet.



STREAM ANALYTICS - CONFIGURE NEW INPUT

An Input defines the data source for the Stream Analytics job.

⌦ **STEP 1:** From the Stream Analytics Configuration blade select **Inputs -> Add -> then specify parameters -> Create -> Close the Inputs blade**.

Settings for New Input Blade

Input alias
TelemetryHub

Source
IoT hub

Subscription
Provide IoT hub settings manually

IoT Hub
Provided in the Lab Supplement

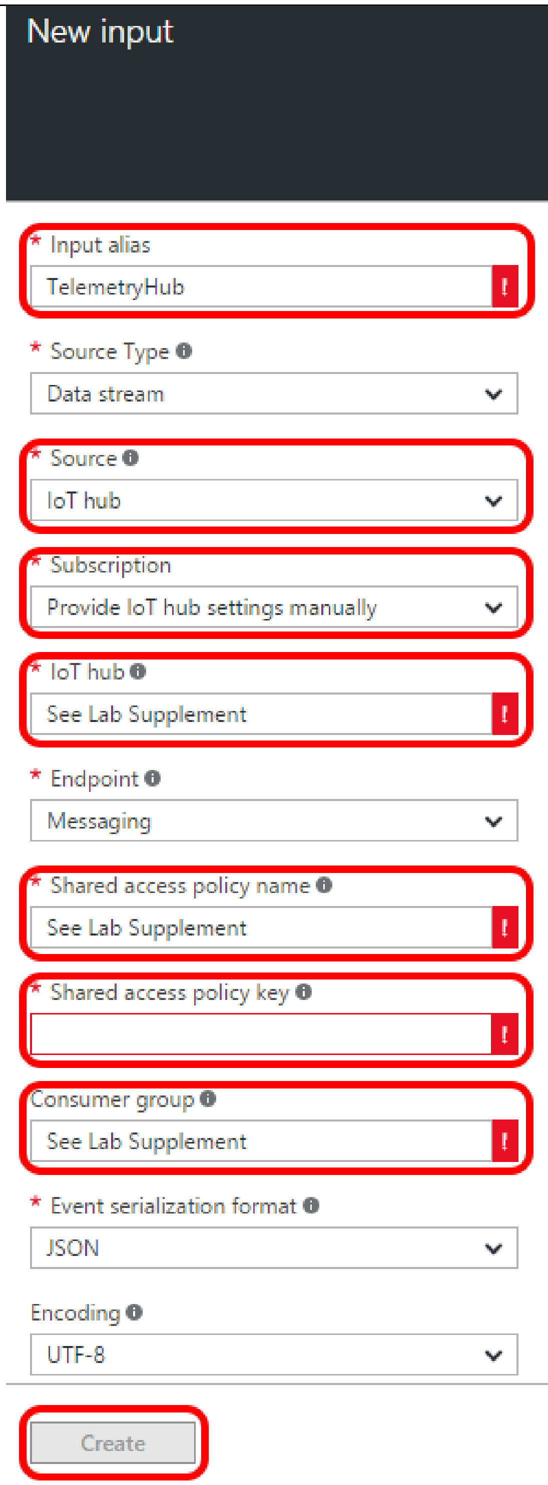
Shared access policy name
iothubowner

Shared access policy key
Provided in the Lab Supplement

Consumer group
Provided in the Lab Supplement

Click Create
It will take a moment or two to create the Input stream.

Finally close the Inputs Blade



STREAM ANALYTICS – CONFIGURE NEW OUTPUT

An Output defines the output destination for the Stream Analytics job.

⌧ **STEP 1:** From the Stream Analytics Configuration blade select **Outputs -> Add -> then specify parameters -> Create -> Close the Outputs blade**.

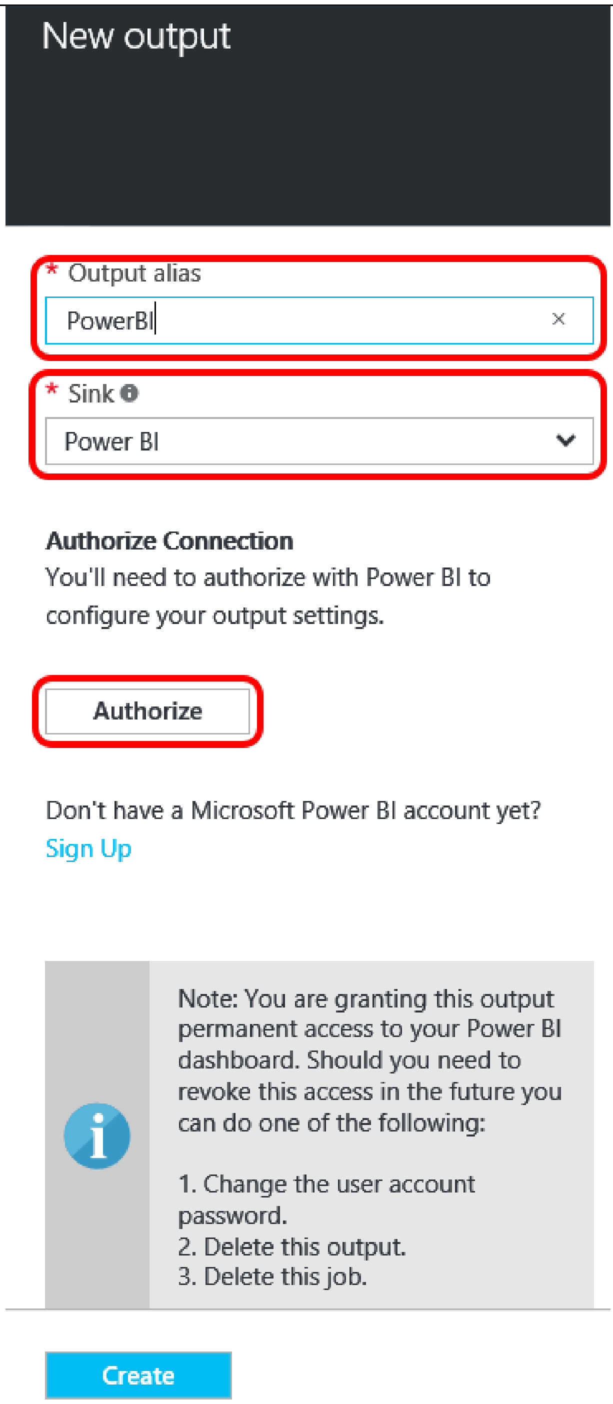
Settings for New Output Blade

Output alias
PowerBI

Sink
Select the **POWER BI**

Authorize
The Power Bi credentials are provided in the Lab Supplement.

Click Authorise and you will be redirected to the Microsoft login page to authorise output to your Power BI account



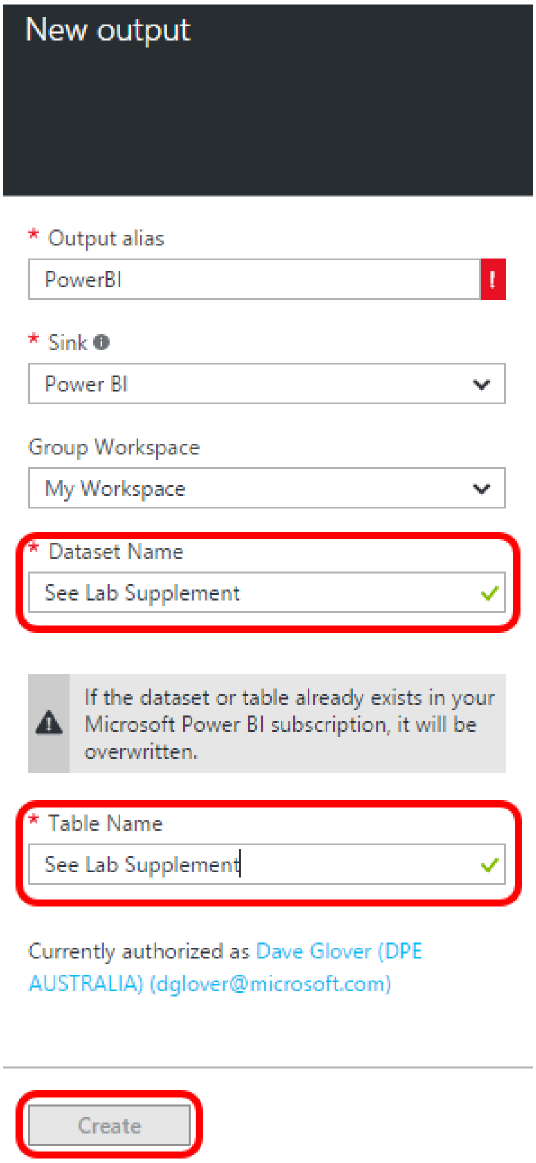
☒ **STEP 2:** Once Power BI is authorised continue setting the remaining configuration fields.

<u>Dataset Name</u>
Provided in the Lab Supplement

<u>Table Name</u>
Telemetry

<u>Click Create</u>
It will take a moment or two to create the Output stream.

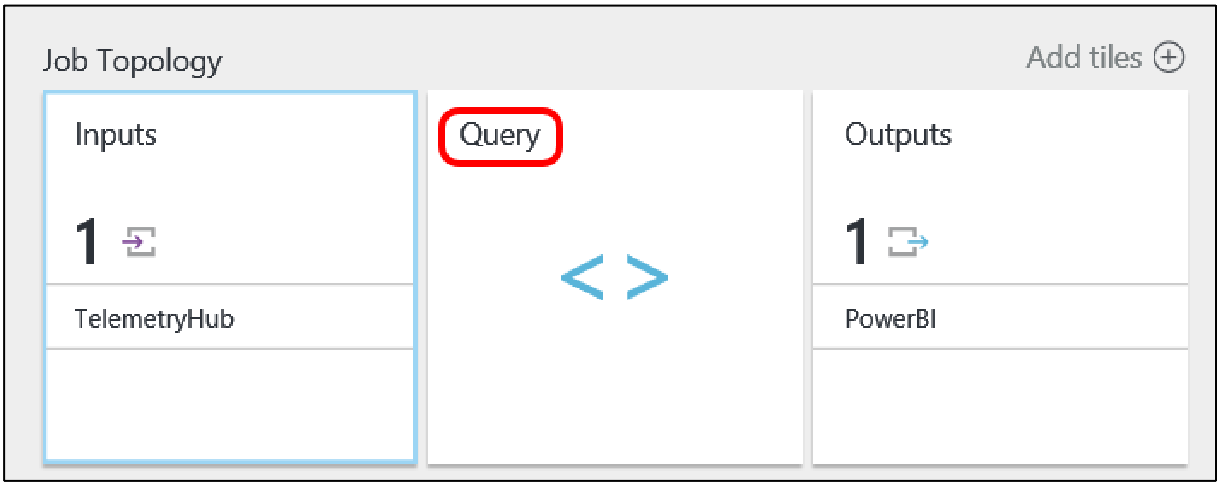
<u>Finally close the Outputs Blade</u>



STREAM ANALYTICS - QUERY CONFIGURATION

Now that the job's inputs and outputs are configured, the Stream Analytics Job needs to know how to transform the input data into the output data source. To do so, you will create a new Query.

⌦**STEP 1:** From the Stream Analytics Configuration blade click **Query**.



⌦**STEP 2**: Replace the default query with the following Stream Analytics Query.

Note: You can copy and paste this query from the Query.txt file found in the Stream Analytics folder on your desktop.

```
SELECT
    iothub.connectiondeviceid deviceid,
    Geo AS GeoLocation,
    Max(DateAdd(Hour, 10, EventEnqueuedUtcTime)) AS TimeCreated, -- AU EST UTC + 10
    Avg(Celsius) AS Temperature,
    AVG(Humidity) AS Humidity,
    AVG(Light) AS Light,
    AVG(HPa) AS AirPressure
INTO
    [PowerBI]
FROM
    [TelemetryHUB] TIMESTAMP BY EventEnqueuedUtcTime
GROUP BY
    iothub.connectiondeviceid, Geo,
    TumblingWindow(Second, 30)
```

The query takes input data from TelemetryHUB and aggregates into 30 second chunks and writes to the output PowerBI.

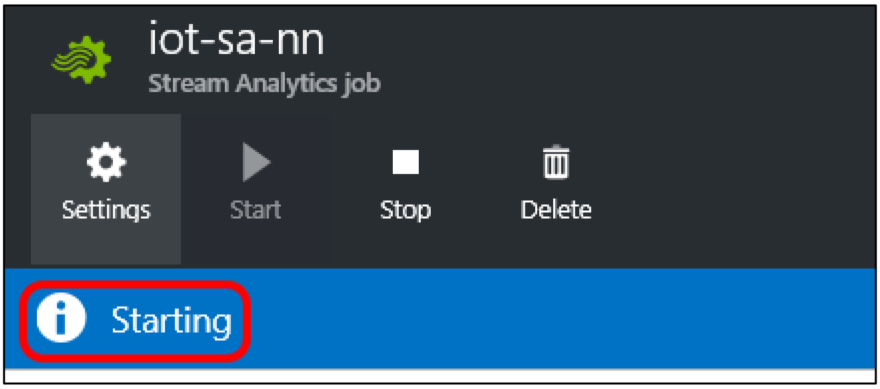
⌦**STEP 3:** Click the **SAVE** button and confirm.

⌦**STEP 4:** Close the Query blade.

STARTING THE STREAM ANALYTICS JOB

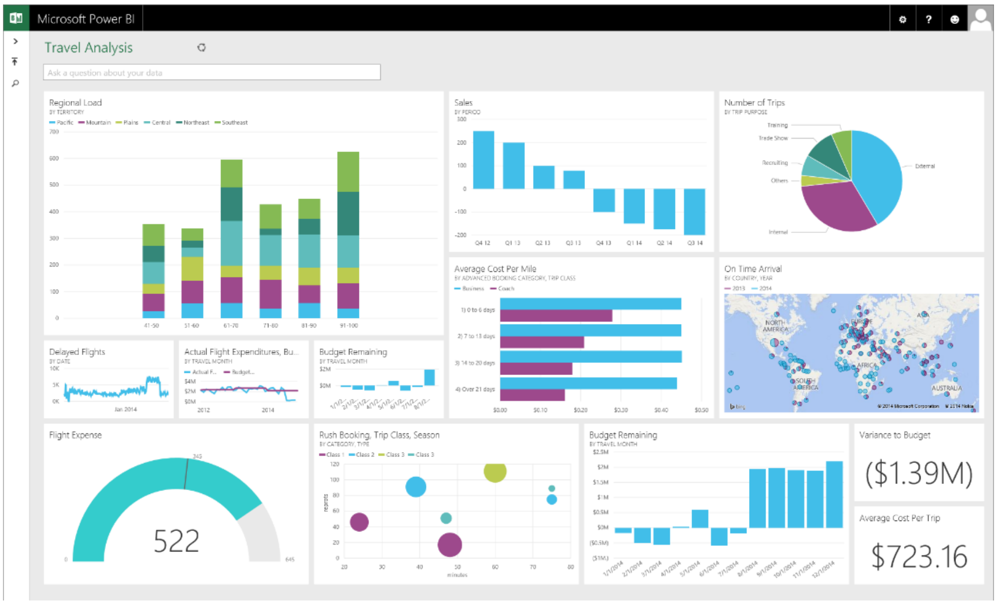
The job is configured and it now needs to be started.

⌦**STEP 1**: From the Stream Analytics configuration blade click **Start** -> **Now** -> **Start**. Allow for 30 to 60 seconds for the job to enter "**Running**" mode.



Once the job starts and it is receiving data from your IoT device it will create the Power BI datasource associated with the given subscription.

Section 6



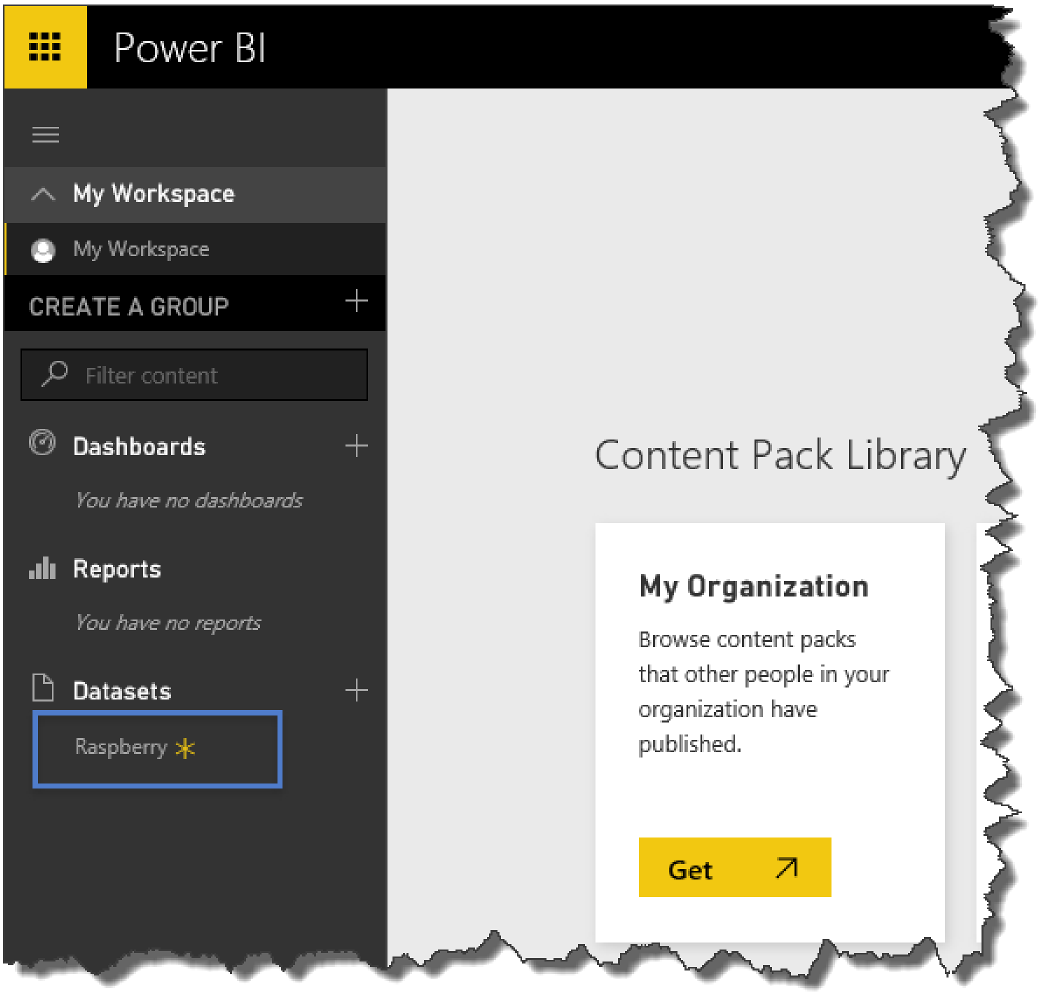
Microsoft Power BI

Transform your data with rich visuals for you to collect and organize

EXPERIEMENT 8: SETTING UP THE POWER BI DASHBOARD

⌦**STEP 1:** Navigate to Power Bi (www.powerbi.com) and authenticate. Click the Hamburger to expand the navigation pane.

The Steam Analytics[7] job needs to run for a few minutes before it appears in the navigation pane.



[7]The Power BI dataset will only be created if the job is running and if it is receiving data from the IoT Hub input. If there is no Raspberry dataset then check the Universal App is running on the Raspberry Pi and it is streaming data to Azure. To verify the Stream Analytics job is receiving and processing data you can check the Azure Management Stream Analytics monitor.

⌧ **STEP 2:** Click on the **datasource name that you created** and start defining the report.

DEFINING A POWER BI REPORT

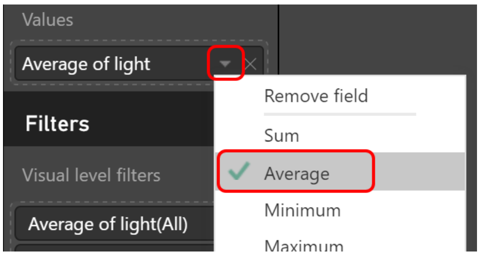
The Report designer will be opened showing the list of fields available for the selected datasource and the different visualizations supported by the tool.

⌧ **STEP 1:** Select Line Chart from the Visualizations and ensure it is selected in the designer.

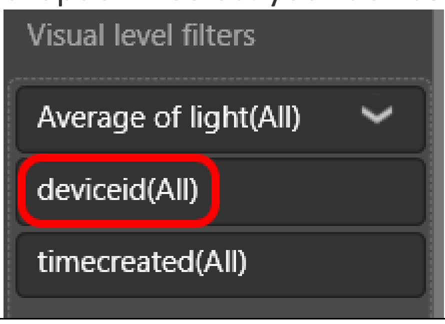
⌧ **STEP 2:** Drag and drop the following fields from the Fields section

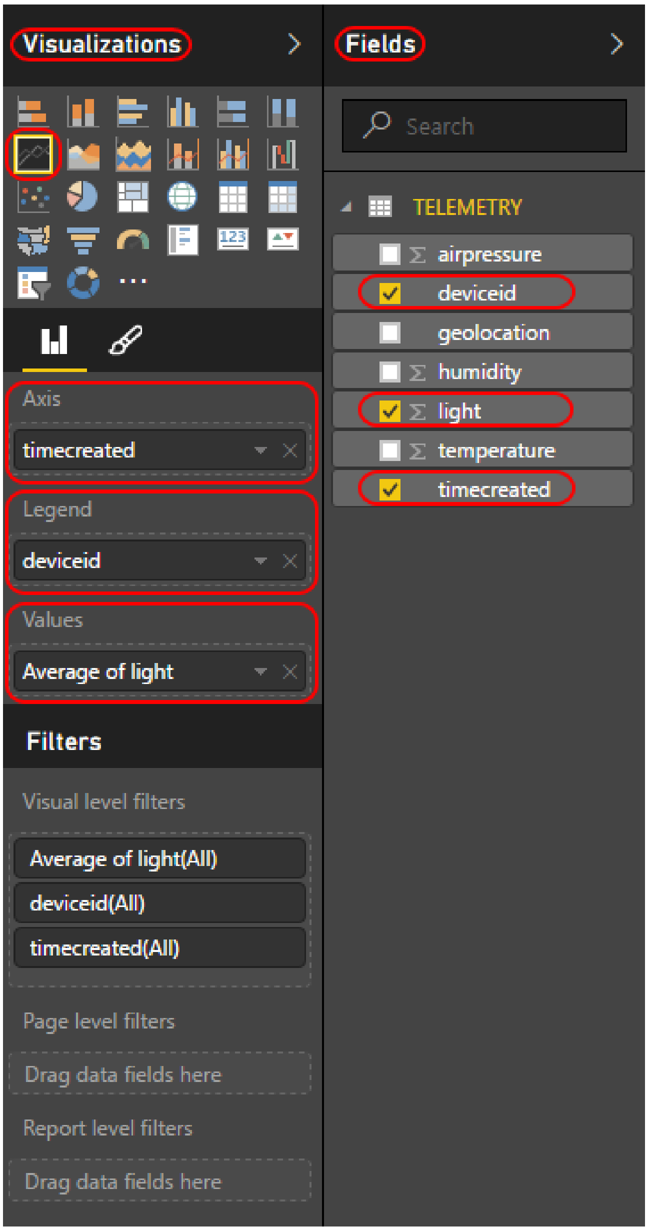
1) deviceid -> Legend
2) light -> Values
3) timecreated -> Axis

⌧ **STEP 3:** Select **Average of light** from the Values dropdown menu.



⌧ **STEP 4:** From the deviceid(All) dropdown select your device.

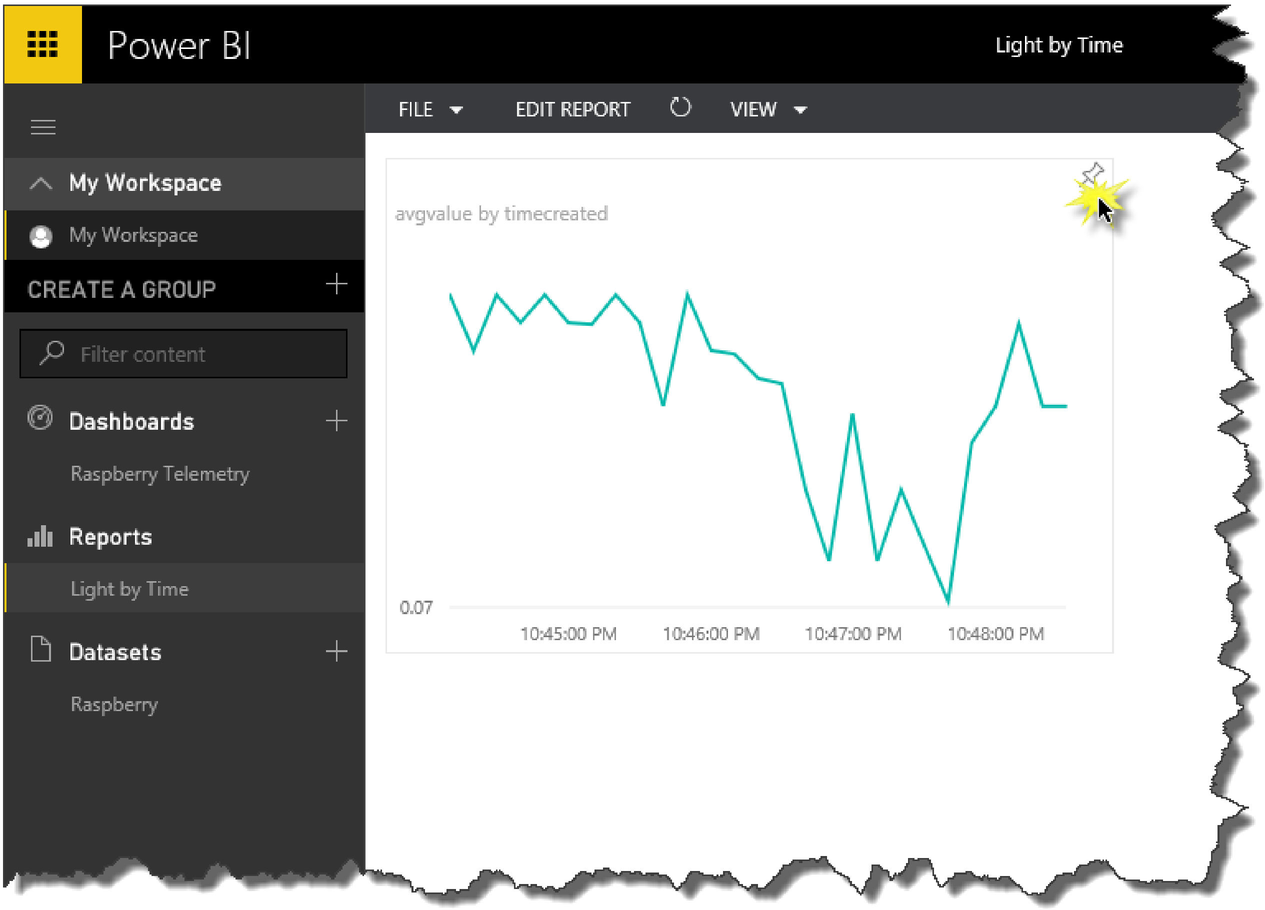




⌦ **STEP 5:** Click the **SAVE** button and set LIGHT BY TIME as the name for the report.

⌦ **STEP 6:** Now create a new Dashboard, and pin this report to it. Click the plus sign (+) next to the **Dashboards** section to create a new dashboard and name it Raspberry Telemetry.

⌦ **STEP 7:** Go back to your report and click the Pin Live Page icon to add the report to the newly created dashboard.



⌦ **STEP 8:** Experiment with other chart types and remember you have access to both Light and Temperature data in the Power BI reporting tool.

Congratulations, you have finished!

EVALUATION

Congratulations, you have successfully completed the IoT Den Experience. You have deployed a Universal Windows App to a Raspberry Pi, streamed data to Microsoft Azure, ingested telemetry using Azure IoT Hub and visualised data with the Power Bi.

Please complete the following steps before you leave.

⌦**STEP 1**: Close Visual Studio.

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

APPENDIX

TROUBLESHOOTING

FORCING A TIME RESYNC

1. From “Windows 10 IoT Core Dashboard”, right mouse click your device and Connect using PowerShell
2. Authenticate
3. At the command prompt, type “w32tm /resync”and press the Enter key to execute.
4. Type Date and press the Enter key to verify the date and time are correct.

LAST BOOT DATE AND TIME

From PowerShell

wmic os get lastbootuptime

USEFUL NETWORK COMMANDS

From PowerShell

- netsh wlan show profile
- netsh wlan add profile *Wi-Fi-ProfileName.xml*
- netsh wlan export profile key=clear
- netsh wlan delete profile *ProfileName*
- netsh wlan connect name= *ProfileName*
- netsh wlan show interfaces
- netsh wlan delete profile *ProfileName*
- netsh wlan add profile Wi-Fi- *ProfileName*.xml
- netsh wlan connect name= *ProfileName*
- netsh interface ipv4 set dns "Wi-Fi" static 192.168.1.1
- netsh interface ipv4 set address "Wi-Fi" static 192.168.1.107 255.255.255.0 192.168.1.1

LAB SUPPLEMENT DATA

Azure Stream Analytics

- Azure authentication Email address and password
- Stream Analytics Job Name
- IoT Hub Consumer Group
- Shared access policy key
- Shared access policy key
- IOT HUB POLICY KEY
- Consumer group

Power BI

- Power Bi Email
- Power Bi Password
- Dataset Name
- IoT Hub Connection String