

Windows 10 IoT Core Labs – IoT Hub Starter

This lab is an introduction to using the Azure IoT Hub. It takes a simple pressure / temperature sensor and connects it to a Raspberry Pi running the Windows 10 IoT Core image and sends data to the IoT Hub.

APPROXIMATE TIME (EXCLUDING PREPARATION WORK): 60 Minutes

PREREQUISITES:

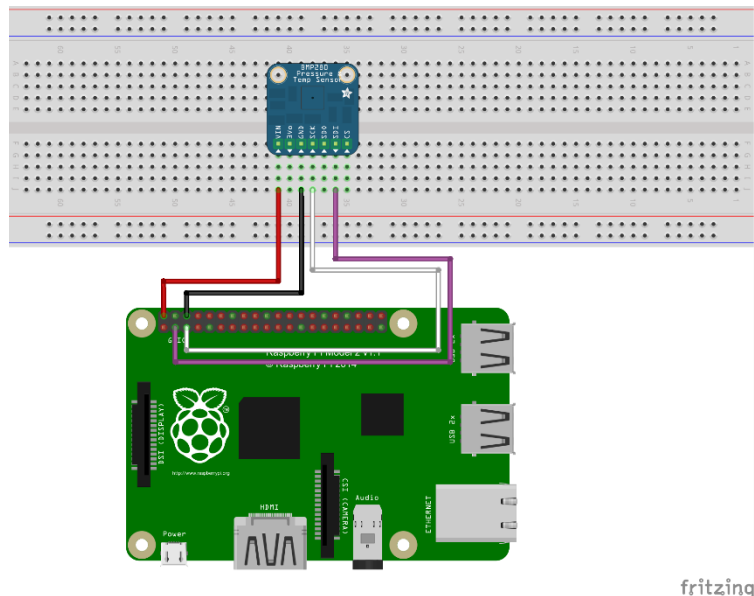
- Setup / Install Windows 10 (v 10.0.10586) or higher
- Download Windows 10 IoT Core Dashboard (<http://go.microsoft.com/fwlink/?LinkID=708576>)
- Setup / Install Visual Studio 2015 Update 2 (You can download Visual Studio Community Edition For Free: <http://go.microsoft.com/fwlink/?LinkID=534599>)
- Install IoT Core Project Templates (<https://visualstudiogallery.msdn.microsoft.com/55b357e1-a533-43ad-82a5a88ac4b01dec>)
- Azure Device Explorer (<https://github.com/Azure/azure-iot-sdks/releases>)
- Azure Subscription (A 30 day trial subscription is available)

PARTS LIST:

- Raspberry Pi 2
- BMP280 Temperature / Pressure sensor (<https://www.adafruit.com/products/2651>)
- Breadboard / Wires

HARDWARE SETUP:

1. Wire the hardware per the following diagram:



DEVICE SETUP:

A. Preparing The Device

The device should already have an image of Windows 10 IoT core on it, but if it does not, you can always install it onto the SD Card.

1. Launch Windows 10 IoT Core Dashboard
2. Click "Set up a new device" in the left hand menu
3. Select the appropriate Device Type and OS Image
4. Insert the Micro SD card into your computer, click "Download and Install" and follow the directions

B. Boot The Device

It may take ~5 minutes for the device to boot up for the first time.

C. Connecting To The Device

When online, the device should appear under the "My Devices" tab on the IoT Core Dashboard.

1. Note the IP address of the machine and either connect to http://<IP_ADDRESS>:8080 or click the globe under "Open in device Portal" for that specific device
2. The default user name is **Administrator**
3. The default password is **p@ssw0rd**

D. Deploying To The Device From Visual Studio

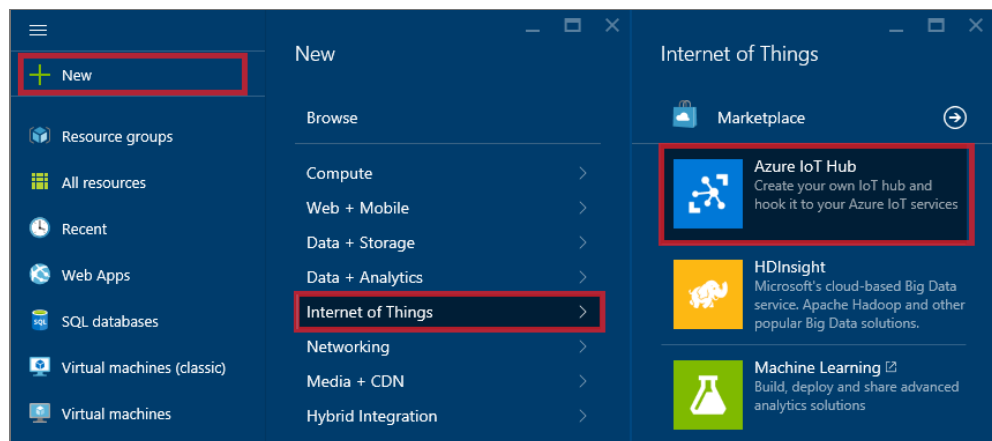
1. In Visual Studio, on the build bar, make sure that the appropriate build / architecture is selected:



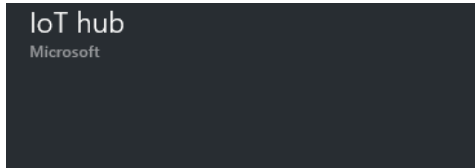
2. Right click the project name and select properties
3. In the settings dialog, click the Debug option
4. Under start options (If the device shows up under IoT Core Dashboard, you should be able to hit "Find" to populate the data
 - i. Target Device: **Remote Machine**
 - ii. Remote Machine: **<IP ADDRESS>**
 - iii. Authentication Mode: **Universal**
5. Build / Deploy the project to the device

AZURE IOT HUB SETUP:

1. Login to Azure Portal (<https://portal.azure.com/>)
2. New -> Internet of Things -> Azure IoT Hub



3. Specify the configuration for the hub



* Name

* Pricing and scale tier

S1 - Standard

* IoT Hub units ⓘ

* Device-to-cloud partitions ⓘ

4 partitions

* Resource group

+ New

New resource group name

* Subscription

Visual Studio Enterprise

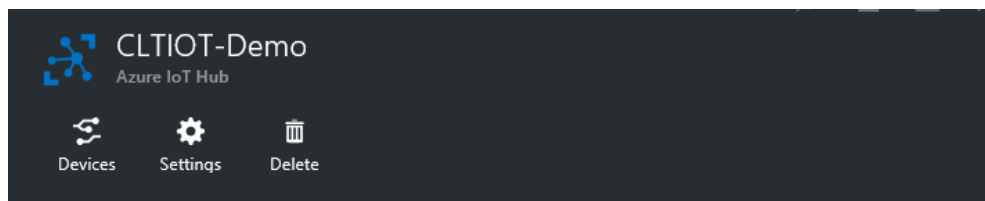
* Location

East US

- Name: **Name of Hub**
- Pricing: **Select F1 – Free**
- Resource Group: **Resource Group Name**
- Location: **Pick Location**

4. Click Create

5. When it is created, take note of the configuration properties and select key icon



Essentials ^

Resource group	CLTIOT-RG	Hostname	CLTIOT-Demo.azure-devices.net
Status	Active	Pricing and scale tier	F1 - Free
Location	East US	IoT Hub units	1

6. Select the Shared access policy of iothubowner and make note of the connection strings and keys. Make sure that you protect the keys as this will allow for devices to connect to your hub

The screenshot shows the 'Shared access policies' configuration page for an IoT Hub named 'iothubowner'. On the left, a table lists policies and their permissions:

POLICY	PERMISSIONS
iothubowner	registry write, service connect, device connect
service	service connect
device	device connect
registryRead	registry read
registryReadWrite	registry write

On the right, the configuration for the 'iothubowner' policy is shown:

- Access policy name:** iothubowner
- Permissions:** Registry read, Registry write, Service connect, Device connect (all checked).
- Shared access keys:** Primary key and Secondary key (both masked).
- Connection string—primary key:** HostName=CLTIOT-Demo.azure-devices.
- Connection string—secondary key:** HostName=CLTIOT-Demo.azure-devices.

7. At this point, devices can now be registered to connect to the hub

REGISTERING A DEVICE

1. Launch Azure Device Explorer
2. On the Configuration tab, paste the Connection string from step 6 above and then click Update

The screenshot shows the 'Device Explorer' application window with the 'Configuration' tab selected. The 'Connection Information' section contains the following fields:

- IoT Hub Connection String:** A large text area for pasting the connection string.
- Protocol Gateway HostName:** A text field.
- Update:** A button to save the configuration.

3. Click the Management Tab to manage all devices connected

- Click Create to register a new device. The keys will be automatically filled in, just give it a unique device name

- Right click on a device to get the connection string for the device and use this value in the appropriate place in the code (see <REPLACE> in the code)

	Id	PrimaryKey	SecondaryKey	ConnectionStr	ConnectionStat	LastActivi
▶	Skoon-BLE	c0zCyY8QMv	3reXGGua+k	HostName=C	Disconnected	4/16/2016
*						

Copy data for all device
Copy data for selected device
Copy connection string for selected device

- Devices can be added / removed / managed via the command line tool as well (<https://github.com/Azure/azureiot-sdks/blob/master/tools/DeviceExplorer/readme.md>)

THE CODE

The complete code is available at the following location: <https://github.com/SkoonStudios/loTLabs/tree/master/Temperature-IoTHub>

The application simply creates an instance of the class to read the sensor, and then sets up a periodic timer to read the data off of the sensor and then send a JSON string to the Azure IoT Hub each time the timer is fired.

A. Create A New Project

- Launch Visual Studio
- Select "New Project"
- Under "Templates", select "Visual C#" and then "Blank App (Universal Windows)"
- Give the app a name and location. The code sample below assumes you have called your application "Temperature-IoTHub"
- Open the Nuget Package Manager Console
- Add the following packages
 - Install-package Newtonsoft.Json
 - Install-package Microsoft.Azure.Devices.Client
 - Install-package PCLcrypto

B. Adding The Code

1. Create class for reading the sensor

The sensor data class is generally derived from the datasheet provided by the chipset manufacturer. For the purposes of this lab, copy / paste from the source project.

- i. Right click project name -> Add -> Class
- ii. Name the file BMP280.cs and click Add
- iii. Remove the scaffolded code and start with the following blank class file
- iv. Add the code from the sensor class in the GitHub project

2. Adding the main code

- i. Open main.xaml.cs
- ii. Replace the existing code with the following scaffold

```
using Microsoft.Azure.Devices.Client;
using Newtonsoft.Json;
using System;
using System.Diagnostics;
using System.Text;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Navigation;

namespace Temperature_IoTHub
{
    public sealed partial class MainPage : Page
    {
        public MainPage()
        {
            this.InitializeComponent();
        }

        protected override async void OnNavigatedTo(NavigationEventArgs navArgs)
        {
        }

        protected override void OnNavigatedFrom(NavigationEventArgs e)
        {
        }
    }
}
```

- iii. Add code to MainPage for the class variables. You will need to replace the code in the connection string with the value when the device was created in Device Explorer

```
//Create a constant for pressure at sea level.  
//This is based on your local sea level pressure (Unit: Hectopascal)  
private const float CLT_SEA_LEVEL_PRESSURE = 1027.3f;  
private TimeSpan TIMER_TICK = TimeSpan.FromMilliseconds(1000);  
private static string CXN_STRING = "<REPLACE>";  
  
//A class which wraps the barometric sensor  
private BMP280 _ptSensor = null;  
  
// Callback timer for reading sensor  
private DispatcherTimer _timer = null;  
  
// Azure Device  
private DeviceClient _devClient = null;
```

- iv. Add code to the OnNavigatedTo event to initialize the variables

```
protected override async void OnNavigatedTo(NavigationEventArgs navArgs)  
{  
    try  
    {  
        //Create a new object for our barometric sensor class  
        _ptSensor = new BMP280();  
  
        //Initialize the sensor  
        await _ptSensor.Initialize();  
  
        // Create Azure Device  
        _devClient = DeviceClient.CreateFromConnectionString(CXN_STRING,  
TransportType.Http1);  
  
        // Create Timer  
        _timer = new DispatcherTimer();  
        _timer.Interval = TIMER_TICK;  
        _timer.Tick += OnTimerTick;  
        _timer.Start();  
    }  
    catch (Exception ex)  
    {  
        Debug.WriteLine(ex.Message);  
    }  
}
```

- v. Add code to stop the timer in OnNavigatedFrom

```
protected override void OnNavigatedFrom(NavigationEventArgs e)
{
    _timer.Stop();
}
```

- vi. Add code for the timer Tick event

```
private async void OnTimerTick(object sender, object e)
{
    float temperature = await _ptSensor.ReadTemperature();
    float pressure = await _ptSensor.ReadPressure();
    float altitude = await _ptSensor.ReadAltitude(CLT_SEA_LEVEL_PRESSURE);

    // Write the values to your debug console
    Debug.WriteLine($"Temperature: {temperature.ToString()} deg C");
    Debug.WriteLine($"Pressure: {pressure.ToString()} Pa");
    Debug.WriteLine($"Altitude: {altitude.ToString()} m");

    if (null != _devClient)
    {
        try
        {
            // Send the values to Azure IoT Hub
            var obj = new
            {
                time = DateTime.UtcNow.ToString("o"),
                temperature = temperature,
                pressure = pressure,
                altitude = altitude,
            };

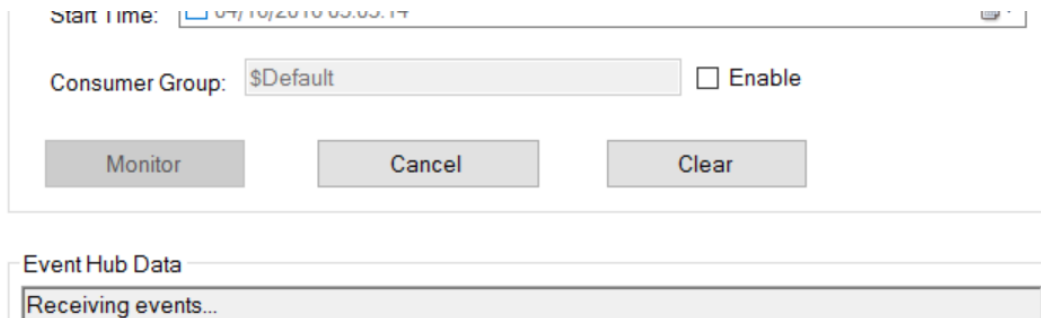
            string jsonText = JsonConvert.SerializeObject(obj);
            Message msg = new Message(Encoding.UTF8.GetBytes(jsonText));
            await _devClient.SendEventAsync(msg);

            Debug.WriteLine($"AZURE: {jsonText}");
        }
        catch (Exception ex)
        {
            Debug.WriteLine("Exception when sending message:" + ex.Message);
        }
    }
}
```


TESTING THE CODE

If the project is deployed and the debugger is connected, the diagnostic output window should write data each time the sensor is read. The Azure Device Explorer can also view the data coming from the device.

1. Click the Data tab and select the Device ID you want to monitor
2. Click the Monitor button to start monitoring the device data
3. Run the IoT application
4. When an event is captured, you should see it in the output pane



Start Time: 07/10/2016 03:03:17

Consumer Group: \$Default ☐ Enable

Monitor Cancel Clear

Event Hub Data

Receiving events...

SUMMARY / ACKNOWLEDGEMENTS

This project was modified from an original introductory hack night for the Charlotte IoT Meetup Group (<http://www.charlotteiot.com>). The sample is loosely adapted from the on the “Weather Station” (<https://www.hackster.io/windows-iot/weather-station>) sample provided as part of the Adafruit Starter Pack for Windows 10 IoT Core on Raspberry Pi 2. We encourage you to innovate and put your own creative touches on this project.

This document may be used freely if kept fully intact. If you do build, innovate, or have questions on this project, I do ask that you please let us know at info@skoonstudios.com.



<http://www.skoonstudios.com>

Last Revised: April 16, 2016

** Skoon Studios, LLC has no affiliation or specific endorsement with Adafruit Industries but in the past have bought components and recommended the detailed tutorials and instructions available on the site.