

Internet of Things Workshop

Lab 2

Building a Gateway

Change Record

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Author | Version | Change Reference |
| 10/5/2015 | Chmitch | 1.0 | Initial draft |
| 1/13/2016 | Chmitch | 1.1 | Updates based on a beta pass and comments by Steve Busby |
| 3/2/2016 | Stevebus | 2.0 | Changed gateway to be a UWP app (for Raspberry PI) vs. PC |

Table of Contents

[Change Record 2](#_Toc444689539)

[Introduction 4](#_Toc444689540)

[Environment setup 4](#_Toc444689541)

[1. Configuring the IOT Hub 5](#_Toc444689542)

[2. Setting up the gateway device 9](#_Toc444689543)

[3. Retrieving the serial port ‘name’ 10](#_Toc444689544)

[4. Coding the gateway 12](#_Toc444689545)

Introduction

This lab is focused on configuring an Azure IOT Hub for capturing your Internet of Things events, and writing a basic gateway in .Net to capture events and send them to the Azure IOT Hub. This lab is the second in a series that walks through building an end-to-end Internet of Things prototype for doing temperature monitoring.

In this series of labs you will:

1. Assemble an Arduino Uno device for temperature monitoring using a prototype kit, and code and deploy a sketch using the Arduino IDE.
2. Write a gateway application (Universal Windows App) on a Raspberry PI to receive the serial data from the Arduino and send data to an Azure IoT Hub.
3. Configure Azure Stream Analytics jobs for gathering and aggregating streaming data for reporting purposes.
4. Build a Power BI dashboard for visualizing real-time and historical event data from the sensor.
5. Integrate the gateway app with the Azure IoT Suite Remote Monitoring pre-configured solution.

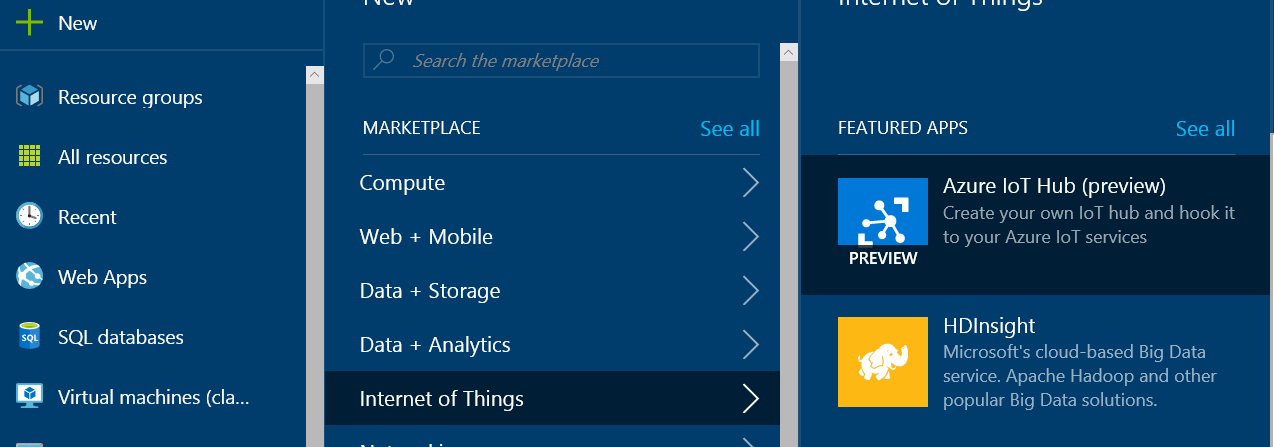
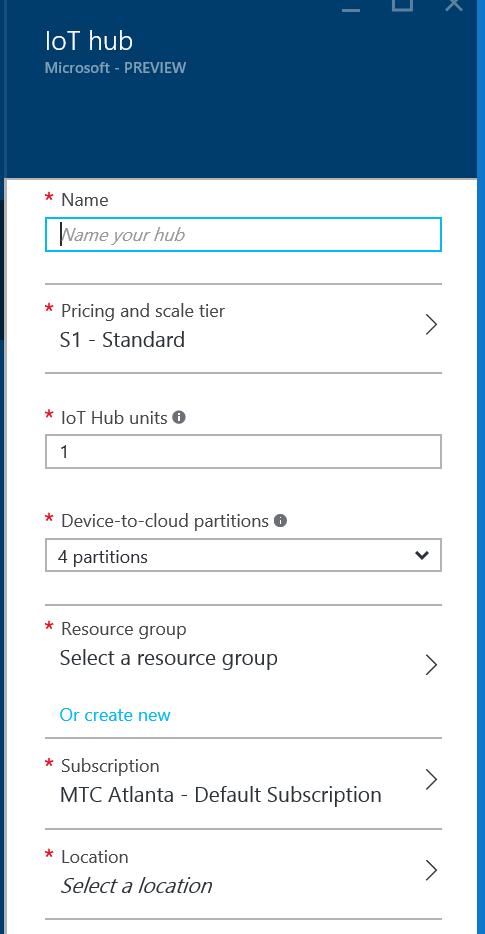
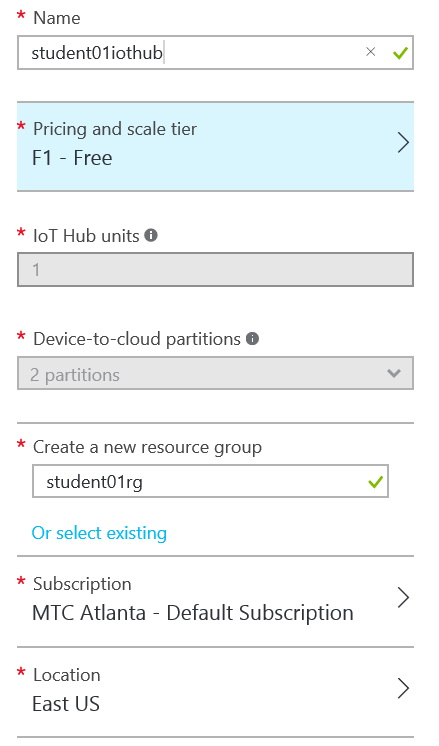
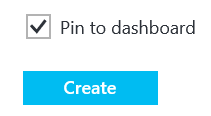
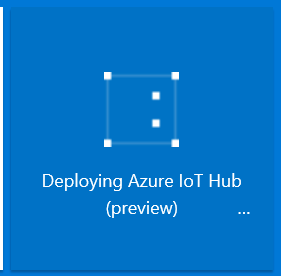
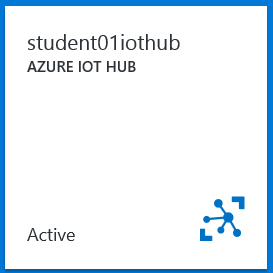
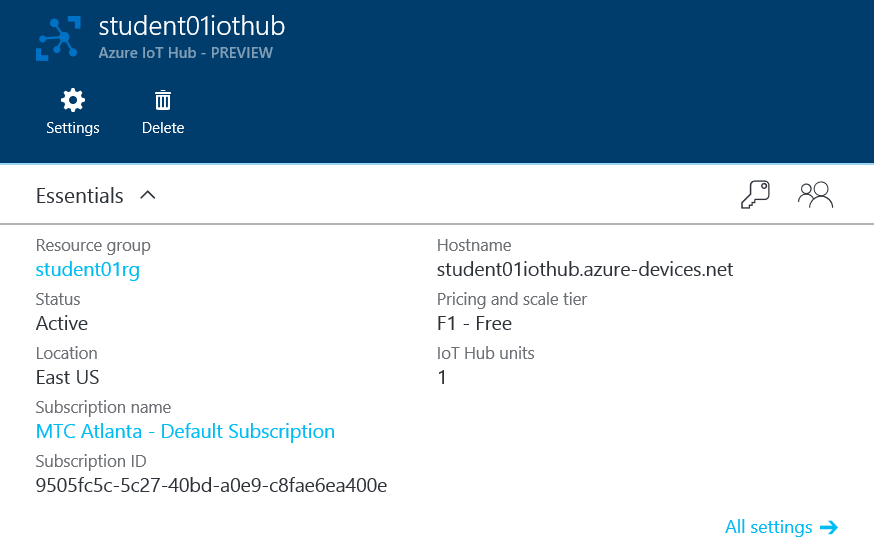
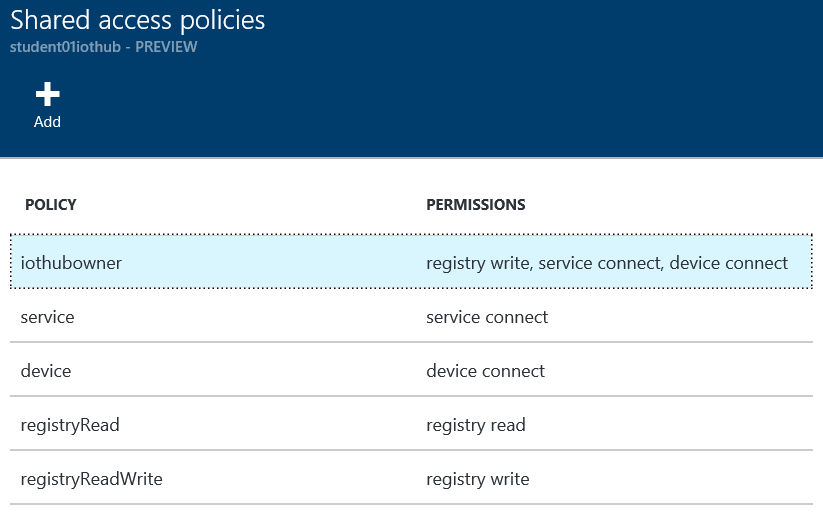
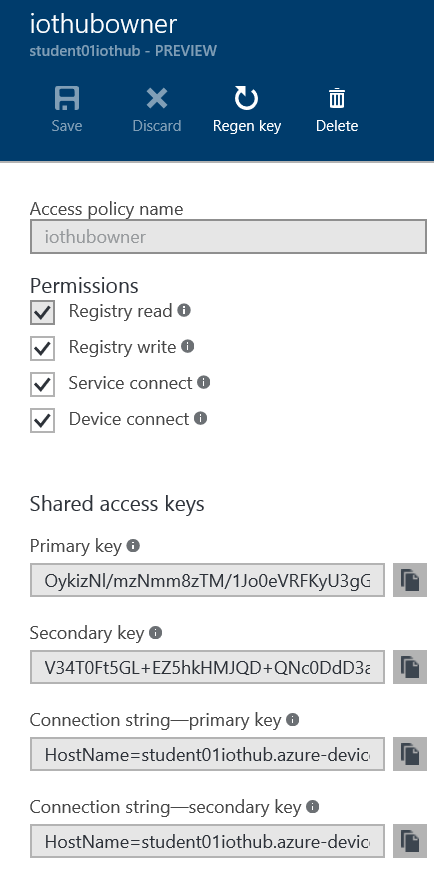
At the end of this lab you will have a functioning IoT gateway capable of receiving sensor data from the Arduino device in lab 1, and posting to Azure IoT Hub

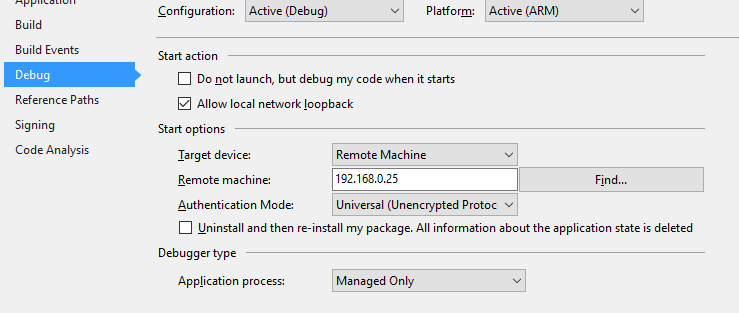
Environment setup

This lab module uses visual studio for development, and assumes Lab 1 has already been completed. **If Visual Studio is not already installed** on your workstation please follow these steps for installation:

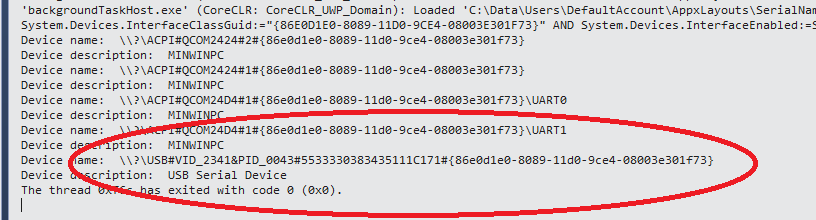
1. Using a web browser navigate to [www.visualstudio.com](http://www.visualstudio.com)
2. Download and install Visual Studio Community Edition with the default options.
3. Configuring the IOT Hub

In this section you will provision an IOT hub for capturing sensor events, and setup its shared access signature for securing the hub:

1. Using a web browser navigate to portal.azure.com. Authenticate with the studentXX account given to you.
2. Create an Azure IoT Hub from the portal navigation using + New -> Internet of Things -> Azure IoT Hub.  
   
3. You’ll be presented with the create screen for IoT hub creation. On this screen enter a unique name for your IoT hub, its pricing tier, and location.   
   
4. For the name enter student<xx>iothub, choose the “Free” pricing tier, create a new resource group called student<xx>rg if one does not already exist, and pick a region (ie. East US).  
   
5. Click the “Create” button to provision the IoT Hub.  
   
6. You’ll be taken back to the azure dashboard page where you’ll see an indicator that your IoT Hub is being deployed.  
   
7. Once the hub creation is complete you’ll see the following:  
   
8. You’ll be taken to the IoT Hub home page.   
   
9. Click on the key to acces the security policies   
   
10. You’ll be presented a listing of different shared access security policies.  
    
11. Click on “iothubowner” to access the security keys for this access signature.  
    
12. Copy the primary connection string with the key by clicking the  icon next to the primary key, and save the copied string. You’ll need this connection string later.
13. Retrieving the serial port ‘name’
14. In order to talk to the Arduino over the serial port, we need to get the name of the serial port that gets created when we connect the Arduino (this is unique per gateway)
15. Connect the USB cable to your Arduino Uno device and connect the other end to one of the free USB ports on your RPI device. The Arduino device should light up and you should see the TX pin blink every couple of seconds.
16. In the Lab 2 folder, navigate to the ‘Lab2\Completed\SerialName’ folder and find the “serial name.sln’ project. Open this solution in Visual Studio
17. Right click on the ‘SerialName’ project (not solution) and choose properties
18. On the Debug tab of the project properties, choose “Remote Machine” for the Target Device, for Remote Machine enter the IP address for your RPI from the IoT Dashboard, and for Authentication Mode ensure that ‘Universal (Unencrypted Protocol)’ is selected

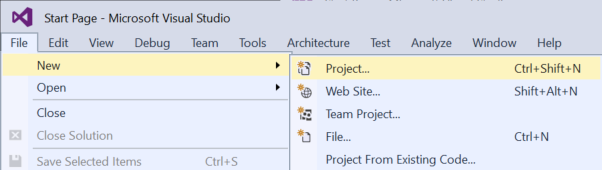


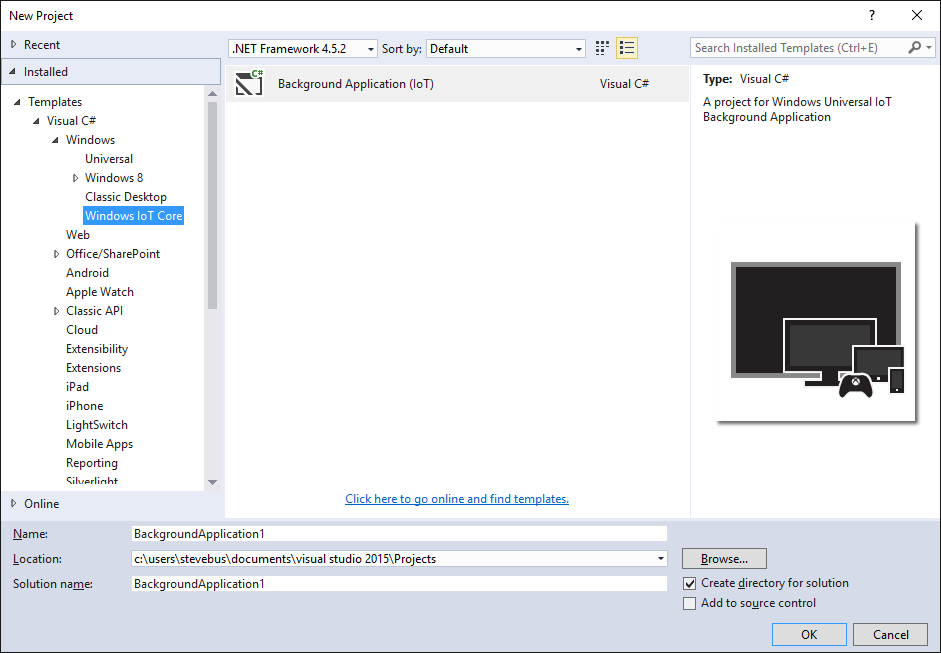
1. Hit the save button to save the properties. Open StartupTask.cs file.
2. Under the Build menu, hit Build solution to make sure the solution will build successfully
3. Once we have a successful build, under the Build menu, hit “deploy solution”
4. Hit F5 to run.
5. Once the solution is deployed and running, make sure the “Output” Window is open in Visual Studio. You should see debug information start to stream by. Look for the text “List of Serial Ports”. Beneath that will be a list of serial ports found on the RPI. Look for one that starts with [\\USB\.\](file:///\\USB\.\) and that will be the one we want (the name will say “USB Serial Device”). See screenshot below for example. Copy it in its entirety and paste into Notepad. We’ll need this later. See screenshot below.

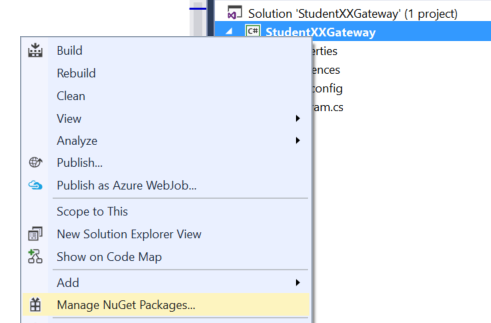


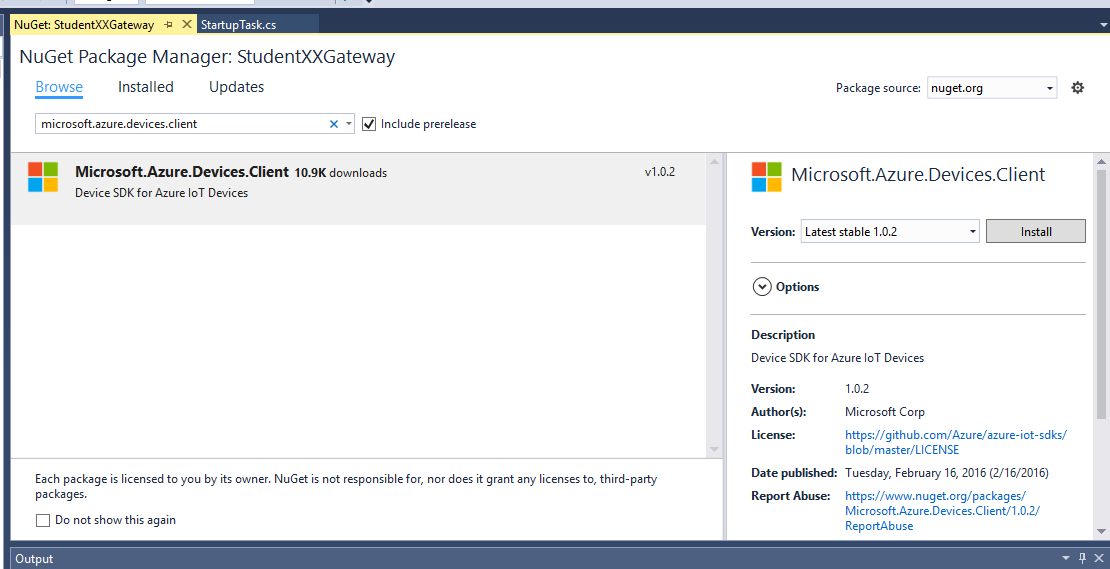
1. Coding the gateway

In this section you will write a Universal Windows App that opens a serial connection to read from the Arduino device, wraps the values in a JSON message, and finally posts the data to the newly created IOT Hub:

1. Launch Visual Studio 2015 and create a new project via the File -> New -> Project menu item.  
   
2. In the “New Project” screen select “Background Application (IoT)” as the project type, and choose a name for the project. Call your project “StudentXXGateway” (replace XX with your student ID provided)



1. Our UWP app will be communicating with an IOT hub so let’s add a NuGet package to enable this. Right click on the project an select manage NuGet packages.  
   
2. In the results select “Microsoft.Azure.Devices.Client” and click the “Install” button.



1. Click “I Accept” on the licensing dialog
2. Repeat steps 4 & 5 with Newtonsoft.Json.
3. Close the package manager window.

In the project we first need to add some using statements to simplify our class references for com port interaction and Azure Service Bus interaction. Open ‘startuptask.cs’ and at the end of the block of using statements add the following items:  
  
using Newtonsoft.Json;

using Microsoft.Azure.Devices.Client;

using Microsoft.Azure.Devices;

using Windows.Devices.SerialCommunication;

using System.Threading.Tasks;

using Windows.Storage.Streams;

using System.Diagnostics;

1. We need to create a class level variable for interacting with the Azure IOT Hub. Right after the Class Program opening bracket “{“ add the following code.

private BackgroundTaskDeferral deferral;

static DeviceClient deviceClient;

private static string iotHubUri = "stevebusrm.azure-devices.net"; // <iothubname>.azure-devices.net

private string deviceId = "student<xx>device"; // replace XX with your studentID

private static string deviceKey = "<device primary key here>";

// classes related to serial communication

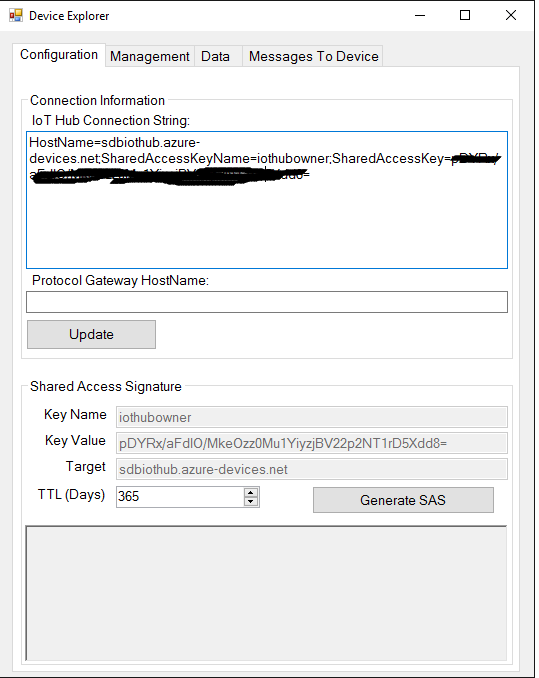
static string serialDeviceName = "<serial port name here>";

private static SerialDevice serialPort;

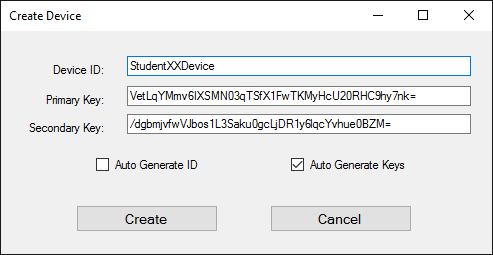
private static DataReader dataReaderObject = null;

private static DataWriter dataWriteObject = null;

1. Now we need to create a device in IoTHub and get its key. This can be done programmatically via the RegistryManager class, but for this POC, we will do it manually through the Device Explorer application (to introduce you to that tool).
   1. On the Cortana search bar, type “Device Explorer” and open the app.
   2. If Device Explorer is not on your dev machine, you can download and install it from <https://github.com/Azure/azure-iot-sdks/releases/download/2016-03-11/SetupDeviceExplorer.msi>



* 1. In the Connection Information box, enter your connection string from your IOTHub and click the “Update” button.
  2. Switch to the “management” tab and click “Create” to create a new device
  3. Enter “StudentXXDevice” (replacing XX with your student ID) into the DeviceID field and click Create



* 1. In the “Device Created” confirmation dialog, copy the DeviceID and Keys. You’ll need this in the next step. Click Done.

1. Replace the placeholder in step 8 with the Primary Key from step 9.d
2. Replace the <serial port name here> in step 8 with the serial port name you copied in section 3. Note that, in C#, you have to escape strings that contain backslashes. You can do so by adding an @ sign in front. So…

"\\?\FTDIBUS#VID\_0403+PID\_6001+AL01GQQHA#0000#{86e0d1e0-8089-11d0-9ce4-08003e301f73}";

becomes

@"\\?\FTDIBUS#VID\_0403+PID\_6001+AL01GQQHA#0000#{86e0d1e0-8089-11d0-9ce4-08003e301f73}";

1. The first thing we need to do is create a function to allow us to open the serial port and set the connection parameters. Add the following function to the StartupTask class  
     
    private async Task SetupSerialConnection()

{

// get a handle to the serial device specified earlier

serialPort = await SerialDevice.FromIdAsync(serialDeviceName);

if (serialPort == null)

Debug.WriteLine("Oops - cannot connect to serial port");

Debug.WriteLine("connected to serial port");

// Configure serial settings

serialPort.WriteTimeout = TimeSpan.FromMilliseconds(1000);

serialPort.ReadTimeout = TimeSpan.FromMilliseconds(1000);

serialPort.BaudRate = 9600;

serialPort.Parity = SerialParity.None;

serialPort.StopBits = SerialStopBitCount.One;

serialPort.DataBits = 8;

serialPort.Handshake = SerialHandshake.None;

// Display configured settings

string status;

status = "Serial port configured successfully: ";

status += serialPort.BaudRate + "-";

status += serialPort.DataBits + "-";

status += serialPort.Parity.ToString() + "-";

status += serialPort.StopBits;

Debug.WriteLine(status);

}

1. Next we will add a function to read the data from the serial port, and then send the data to IoTHub. Add the following function to the StartupTask Class (feel free to copy/paste ☺ )

private async Task ReadAsync()

{

// for the lab, we have a fixed message length, so we can just read that many bytes at a time

// otherwise we have to read a byte at a time and look for newlines

uint ReadBufferLength = 13;

// Set InputStreamOptions to complete the asynchronous read operation when one or more bytes is available

dataReaderObject.InputStreamOptions = InputStreamOptions.Partial;

// load the reader with data from the buffer

UInt32 bytesRead = await dataReaderObject.LoadAsync(ReadBufferLength);

// did we get a message?

if (bytesRead > 0)

{

// get data out of the buffer as a string

string x = dataReaderObject.ReadString(bytesRead);

// chop off the CRLF

x = x.Substring(0, x.IndexOf("\r\n"));

// split the humidity and temp into an array

string[] readings = x.Split(',');

string tempStr = string.Format("Humidity={0}, Temperature={1}", readings[0], readings[1]);

System.Diagnostics.Debug.WriteLine(tempStr);

// create a new telemetryDataPoint object to hold the data

var telemetryDataPoint = new

{

DeviceId = deviceId,

Temperature = readings[1],

Humidity = readings[0],

ExternalTemperature = 0

};

// serialize the telemetryDataPoint object into JSON

string messageString = JsonConvert.SerializeObject(telemetryDataPoint);

// send the message to IotHub

SendDeviceToCloudMessagesAsync(messageString);

}

}

1. The next step is to implement the SendDeviceToCloudMessagesAsync function. This is the function that actually sends the message we received to IoTHub. Add the following function to your StartupTask class

static async void SendDeviceToCloudMessagesAsync(string messageToSend)

{

// conver message to a byte array and wrap it with IoTHub message metadata

var message = new Message(Encoding.ASCII.GetBytes(messageToSend));

// send the message to IoTHub

await deviceClient.SendEventAsync(message);

}

1. Now we need a function to write data to the serial port. This will allow us to send ‘commands’ to the Arduino device to simulate command and control of the device. Add this function to the StartupTask class

static async Task WriteSerialAsync(string Message)

{

// get a handle to the output buffer of the serial port so we can write to it

dataWriteObject = new DataWriter(serialPort.OutputStream);

// write the data and append a newline (arduino uses this to parse)

// this line only writes to internal buffer

dataWriteObject.WriteString(Message + '\n');

// write to serial port

UInt32 bytesWritten = await dataWriteObject.StoreAsync();

if (bytesWritten > 0)

{

string tempStr = Message + " ";

tempStr += " written successfully!";

}

// detach our handle to the serial buffer and destroy

dataWriteObject.DetachStream();

dataWriteObject.Dispose();

}

1. Next we need to write the code to receive commands sent to the device from IoTHub. This code calls an async function to receive a command from the hub, parses the command and then calls the WriteSerialAsync function to send the command to the Arduino device. Add the following function to the StartupTask class

static async Task ReceiveCommands()

{

Message receivedMessage;

string messageData;

while (true)

{

// checck for a messages

receivedMessage = await deviceClient.ReceiveAsync();

// did we receive a message?

if (receivedMessage != null)

{

// unwrap the messages object and get the message string

messageData = Encoding.ASCII.GetString(receivedMessage.GetBytes());

string tempStr = String.Format("\*\*\*\*\*\*Received Command: {0}", messageData);

Debug.WriteLine(tempStr);

// tell IoTHub that we got the message (updates the status in the portal)

await deviceClient.CompleteAsync(receivedMessage);

// see what the command was and act on it (send to arduino)

if(messageData.Contains("ON"))

{

Debug.WriteLine("Turning LED on");

WriteSerialAsync("ON").Wait();

}

else if(messageData.Contains("OFF"))

{

Debug.WriteLine("Turning LED Off");

WriteSerialAsync("OFF").Wait();

}

else

{

Debug.WriteLine("Unrecognized command - ignoring...");

}

}

// wait before polling again (only have to do this if it's HTTP connection)

System.Threading.Tasks.Task.Delay(10000).Wait();

}

}

1. Now that we have the support methods in place, we need to add the code to call these methods. Replace the code in the “Run” method with this code

// get a deferral token. this keeps the app from 'exiting' until we want it to

deferral = taskInstance.GetDeferral();

// create the connection to IoTHub, based on the URI, device ID, and key from above

deviceClient = DeviceClient.Create(iotHubUri,

AuthenticationMethodFactory.

CreateAuthenticationWithRegistrySymmetricKey(deviceId, deviceKey),

TransportType.Http1);

// connect to the serial port

SetupSerialConnection().Wait();

Debug.WriteLine("Setting up Data Reader");

// get a pointer to the buffer of data that has been sent to the serial port so we can read it

dataReaderObject = new DataReader(serialPort.InputStream);

Debug.WriteLine("Wiring up Command Receiver...");

// start the thread to listen for "commands" from IoThub (i.e. turn our LED on/off)

ReceiveCommands(); //.Start();

// loop forever and receive serial data and sent to IoTHub

Debug.WriteLine("Starting receive loop");

while (true)

{

ReadAsync().Wait();

}

1. The final step, we have to give the app the ‘permission’ to use the serial port. UWP apps have a standard way to specify the capabilities required by the device. These are included in the Package.appxmanifest file as part of the project. Open that file by right-clicking on it and choosing “Open with” and picking “XML (Text) Editor” from the list. Replace the entire <Capabilties> XML segment with the below.

<Capabilities>

<Capability Name="internetClient" />

<DeviceCapability Name="serialcommunication">

<Device Id="any">

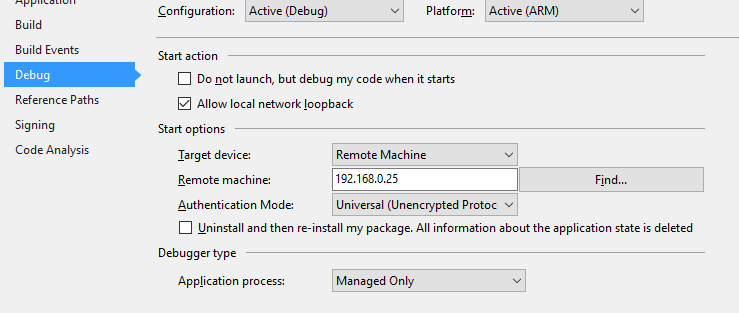
<Function Type="name:serialPort" />

</Device>

</DeviceCapability>

</Capabilities>

1. ) Right click on the ‘Student<XX>Gateway project (not solution) and choose properties
2. On the Debug tab of the project properties, choose “Remote Machine” for the Target Device, for Remote Machine enter the IP address for your RPI you got in step 3, and for Authentication Mode ensure that ‘Universal (Unencrypted Protocol)’ is selected



1. Your gateway code is complete. Hit F5 to run the code. After a few minutes to deploy it, you should see feedback (from the debug.writeline methods in the code) that the app has successfully connected to the Arduino, has set up the thread to listen for commands, and is now reading data from the serial port and posting to IoTHub. You should see output that looks like this (the Humidity/temperature lines are good. They may be mixed among other debug output not produced by the app)

'backgroundTaskHost.exe' (CoreCLR: CoreCLR\_UWP\_Domain): Loaded 'C:\Data\Users\DefaultAccount\AppxLayouts\StudentXXGateway-uwpVS.Debug\_ARM.stevebus\System.Reflection.Primitives.dll'. Module was built without symbols.

'backgroundTaskHost.exe' (CoreCLR: CoreCLR\_UWP\_Domain): Loaded 'Anonymously Hosted DynamicMethods Assembly'.

Humidity=41.00, Temperature=69.80

The thread 0x554 has exited with code 0 (0x0).

The thread 0xef4 has exited with code 0 (0x0).

Humidity=41.00, Temperature=69.80

Humidity=41.00, Temperature=69.80

Humidity=41.00, Temperature=69.80

Humidity=41.00, Temperature=69.80

Humidity=41.00, Temperature=69.80

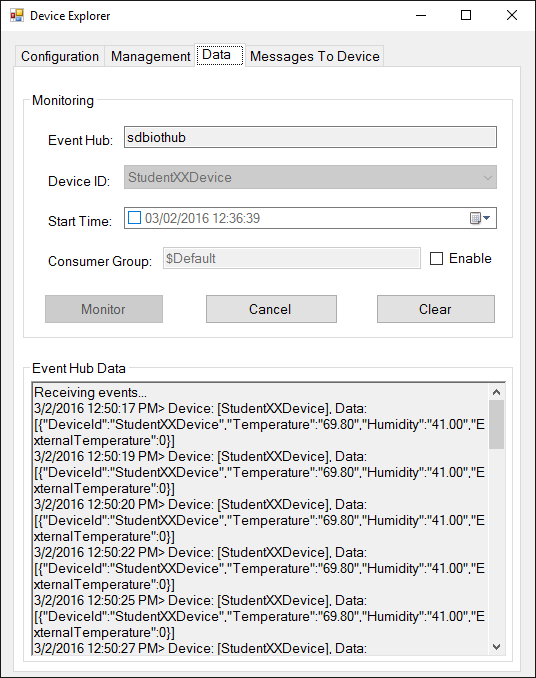
The thread 0x178 has exited with code 0 (0x0).

Humidity=41.00, Temperature=69.80

Humidity=41.00, Temperature=69.80

Humidity=41.00, Temperature=69.80

1. To confirm the app is talking to IoTHub, you can pull back up the Device Explorer app. Go to the “Data” tab, select your device (if it’s not autoselect) and hit “Monitor”.. this connects to the hub as a “reader” and spies on the messages as they are going into the hub. You should see output similar to this, showing the JSON messages flowing into the hub.



1. Congratulations – you now have a RPI based gateway pulling data from your Arduino device and sending to IoTHub.