Wesley Bakker

wesley.bakker@motion10.com

Abstract

During this lab you will connect all the wires

IoT Workshop Lab 1

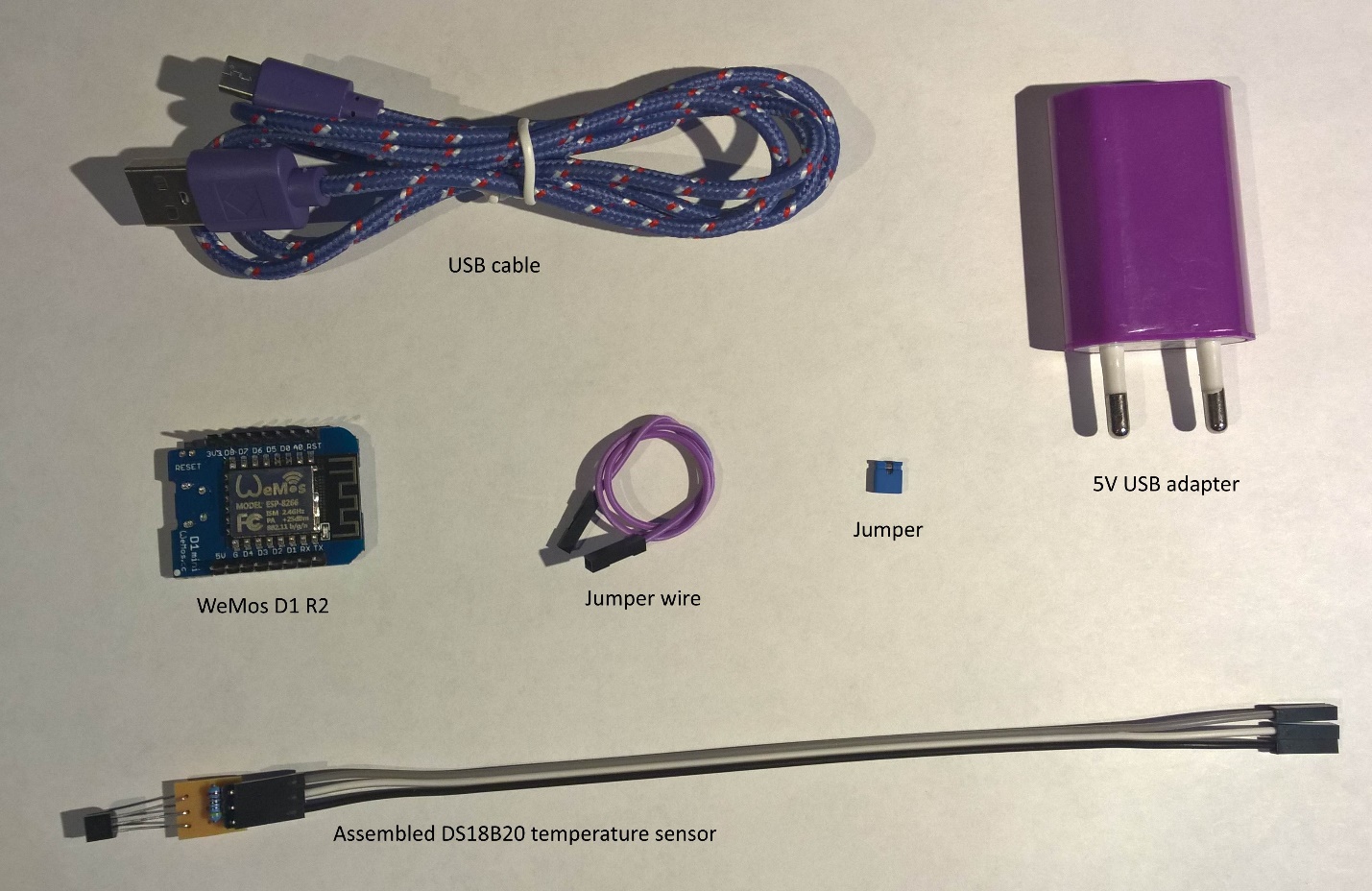
Wiring your device

# Goal

Your device comes pre-programmed for sending data to the Microsoft Azure IoT Hub. During this lab you will enable your device by connecting all the wires to the correct GPIO pins.

After completing this lab, you can configure your device and see messages being send by your device as they arrive at the cloud.

# What’s in the bag?



* Jumper allows for connecting two adjacent GPIO pins
* Jumper wire allows for connecting GPIO pins
* USB cable is both used for powering and programming your WeMos D1 R2
* 5V USB adapter is used for powering the device once it’s programmed
* WeMos D1 Mini R2 programmable WiFi IoT device
* Assembled temperature sensor is used for measuring the temperature with the WeMos D1 R2

Verify if everything is in the bag. If anything is missing, please ask one of the instructors to help you out.

# GITHUB

The code that is pre-programmed on the device is available on github.com. Please open a browser and navigate to the following url: <https://github.com/webbes/DallasWifiMonitor> where you’ll find the complete project.

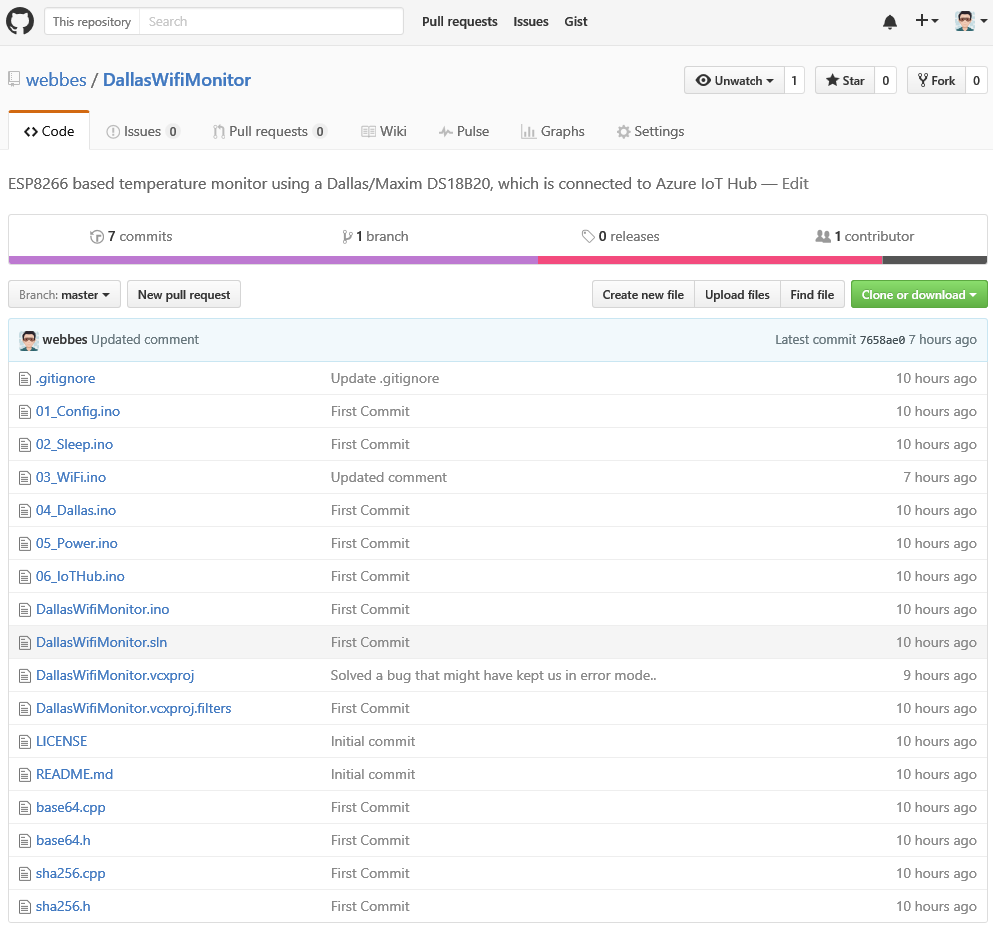


Figure project view

One of the good things about GitHub is that you can view these files without needing to download them and without any software requirements. Where and how we need to connect our wires is defined in the code and datasheets of the devices. So let’s get started.

# 02\_Sleep.ino

Click the 02\_sleep.ino on GitHub which will open the file.



Figure 02\_Sleep.ino

Notice that we do call the ESP.deepSleep() function. This will put the device in a low power consumption state between temperature measurements. Let’s look at the API documentation at: <https://github.com/esp8266/Arduino/blob/master/doc/libraries.md#esp-specific-apis>

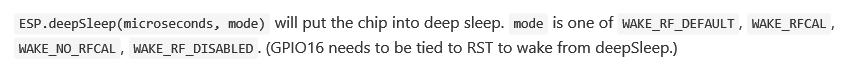


Figure API Documentation ESP8266

You’ll notice that, in order for the device to wake from deepSleep we need to connect GPIO16 to RST.

Head over to the WeMos D1 mini R2 documentation at: <http://www.wemos.cc/Products/d1_mini.html> and search for the pin mappings.



Figure Pin mappings WeMos D1 Mini R2

So we need to connect the ESP8266 GPIO16 pin to its RST pin. We now need to find the corresponding WeMos D1 Mini R2 pins. Which pins do we need to connect?

If you think you do know the answer, please continue to the next page.



Figure Pins we need to connect for waking up from deepSleep

So we need to connect the RST pin and the D0 pin. Head over to the WeMos D1 Mini R2 board and see if you can find the required pins.

Are they adjacent to each other? If so, we could use the **jumper** to connect them, if not we need to use the **Jumper wire**.

Make your decision, and connect the two pins. If you’re done, continue to the next page.

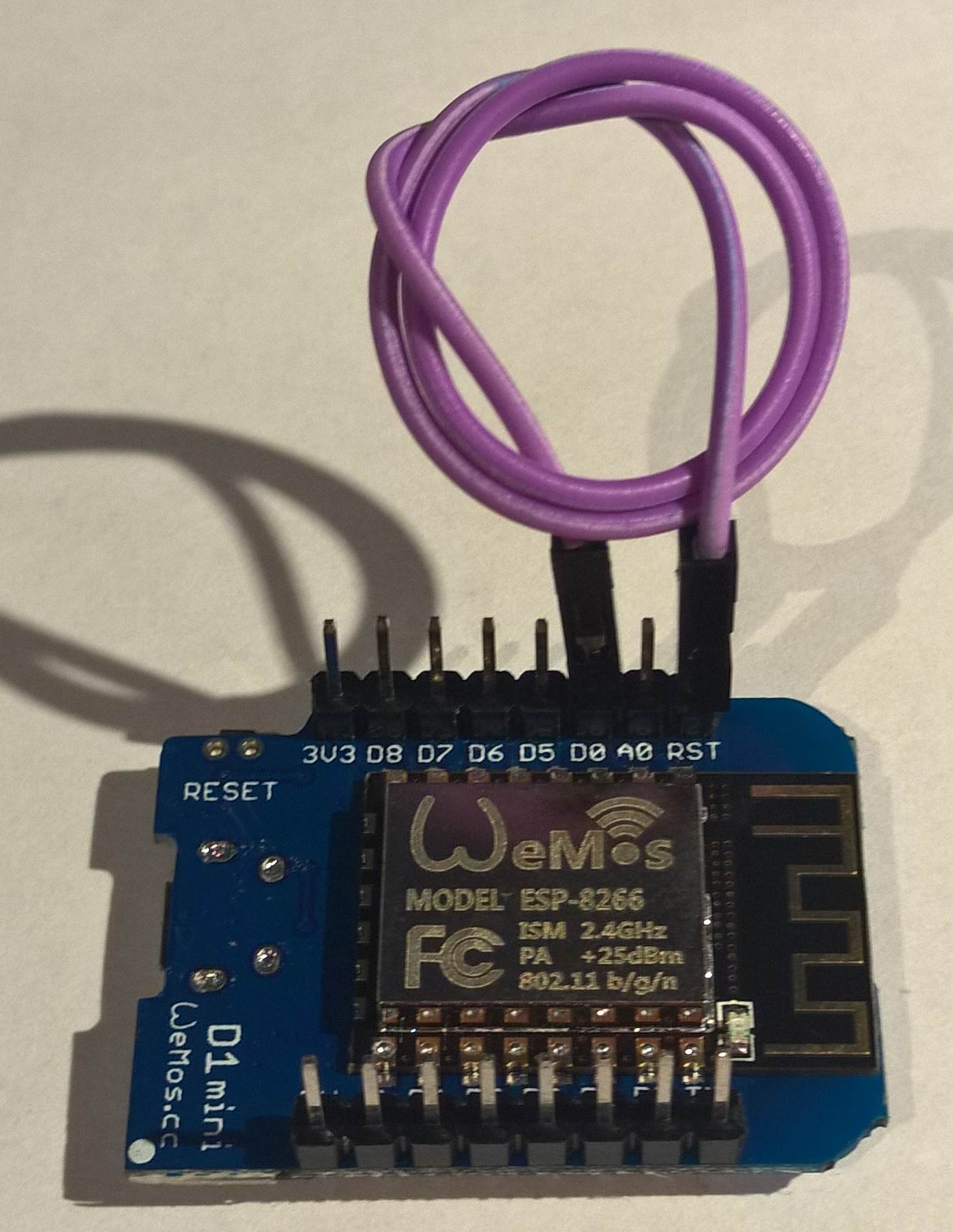


Figure Correct wiring for waking up from deep sleep

# 04\_Dallas.ino

Let’s get back to GitHub and now open the 04\_Dallas.ino.

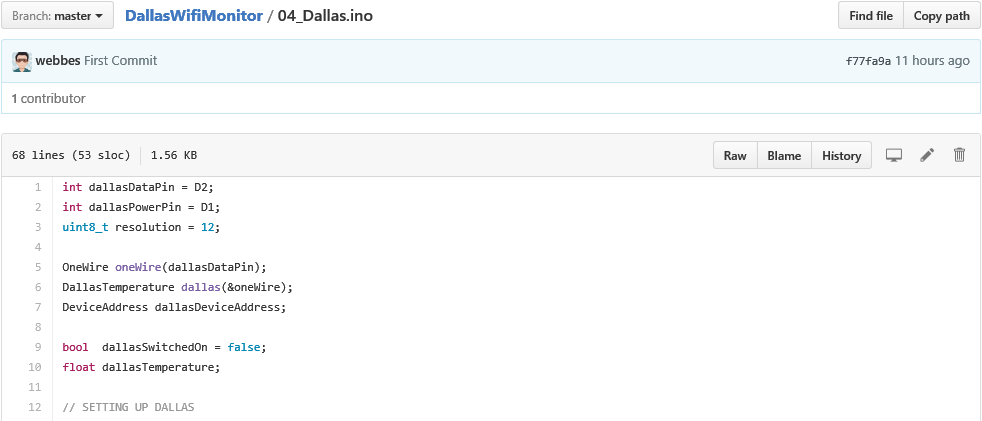


Figure 04\_Dallas.ino (first few lines)

When we look at the first two lines of code it states that we use the WeMos D1 Mini R2 digital pin D2 for data communications and digital pin D1 to apply power (turn it on and off).

We now need to head over to the datasheet of the DS18B20 and find the corresponding pins. Head over to the datasheet at: <https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf>

Can you find the correct pins? If so, continue to the next page.

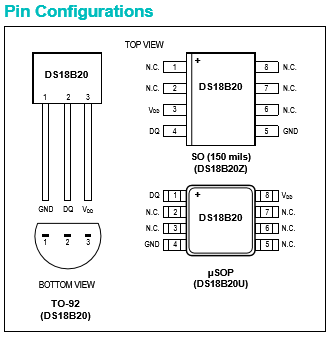


Figure Pin configuration on the DS18B20 as defined in the datasheet

As seen from the bottom or the flat front of the sensor the left pin is GND, the middle pin is DQ(data) and the right pin is Vdd (power).

Now take a closer look at our assembled temperature sensor.

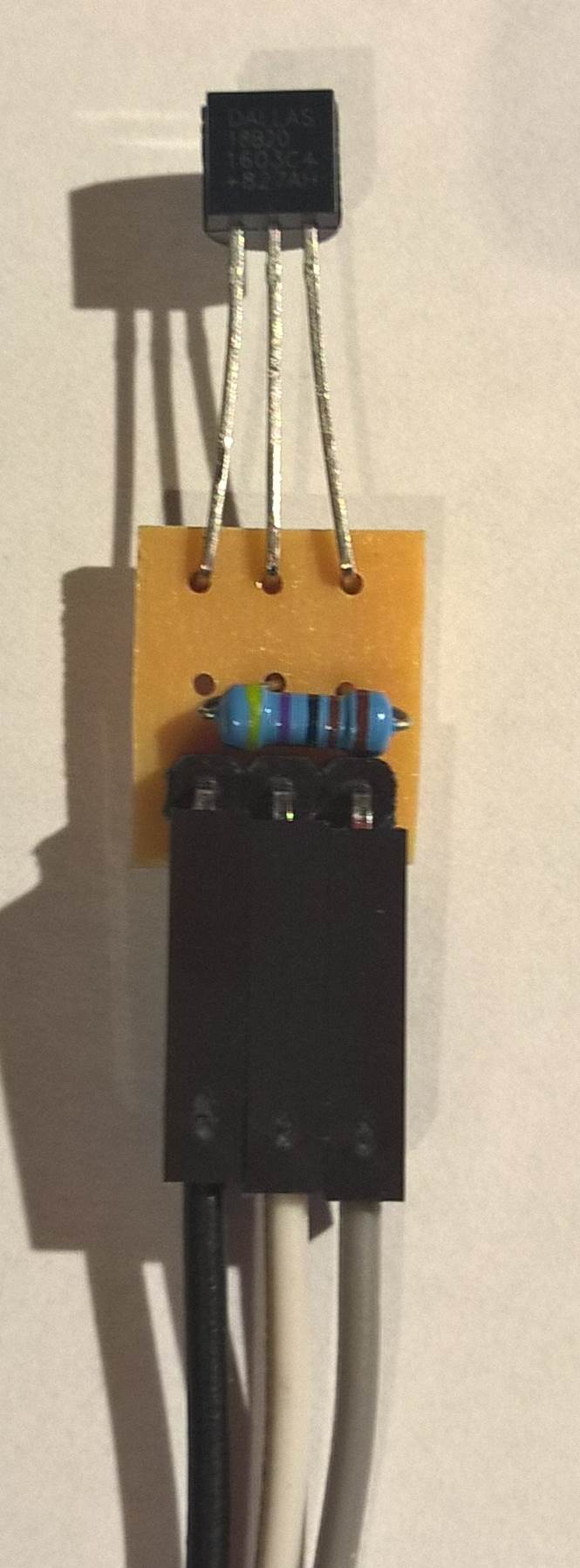


Figure Assembled temperature sensor

Can you figure out which wire color should be connected to which WeMos D1 Mini R2 pin?

In this picture the black wire should connect to G (which stands for GND Figure 4), the white wire should be connected to D2 and the grey wire should be connected to D1. As the colors of your jumper wires differs from this example, please take a moment to write down which wire connects to which pin for your configuration.

|  |  |
| --- | --- |
| WeMos D1 Mini R2 Pin | Jumper wire color |
| G (GND) |  |
| D1 (Vcc - Power) |  |
| D2 (DQ – Data) |  |

Once done, you can connect the wires to the correct pins. Please take your time as connecting the wires wrongly might damage the device permanently!

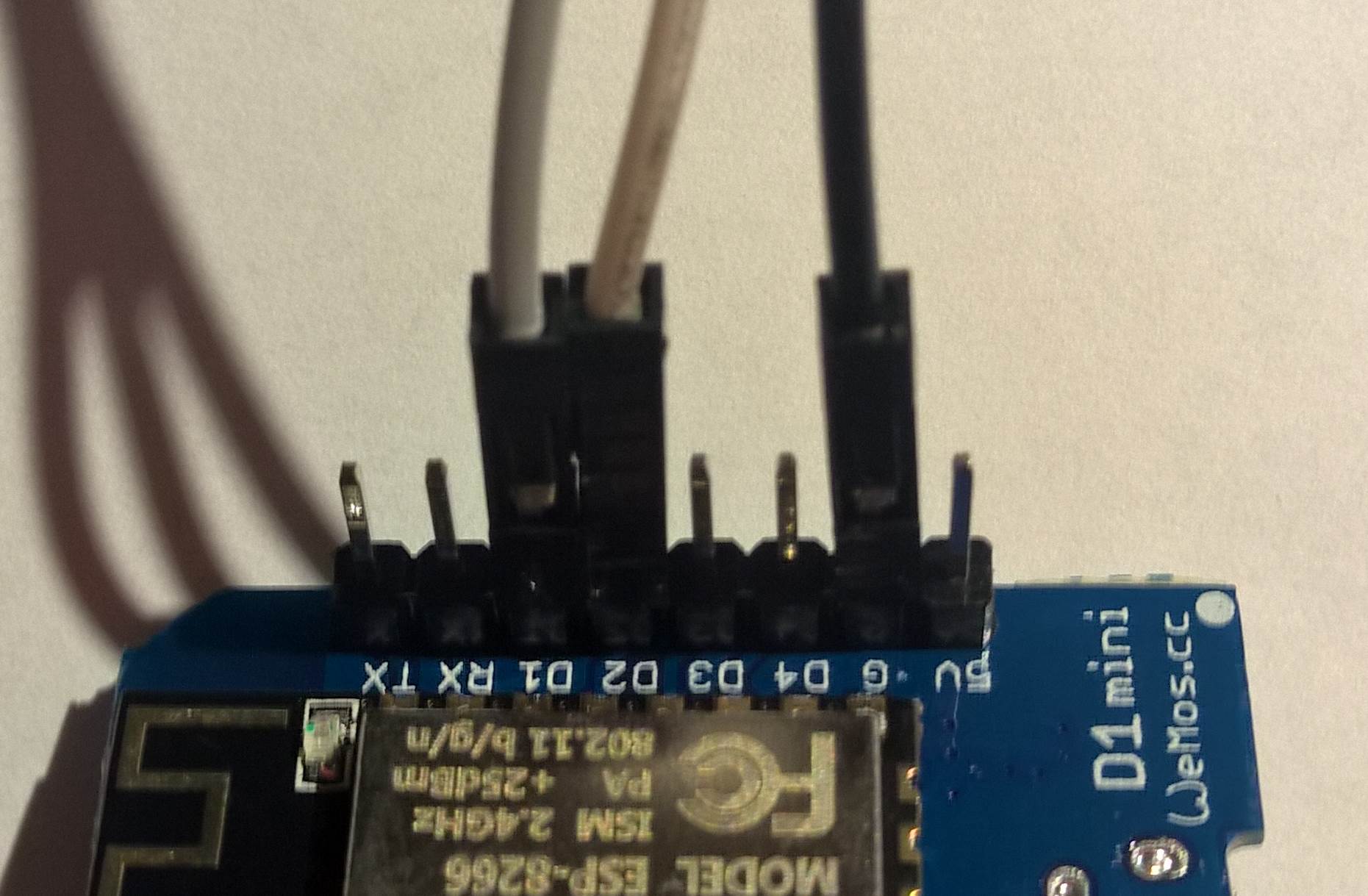


Figure Sample connection of the assembled temperature sensor

# 01\_Config.ino

The first time when we start our device, we would like to be able to configure the network settings. To do so we need to start the device in config mode. Please head back to GitHub and open the 01\_Config.ino file.

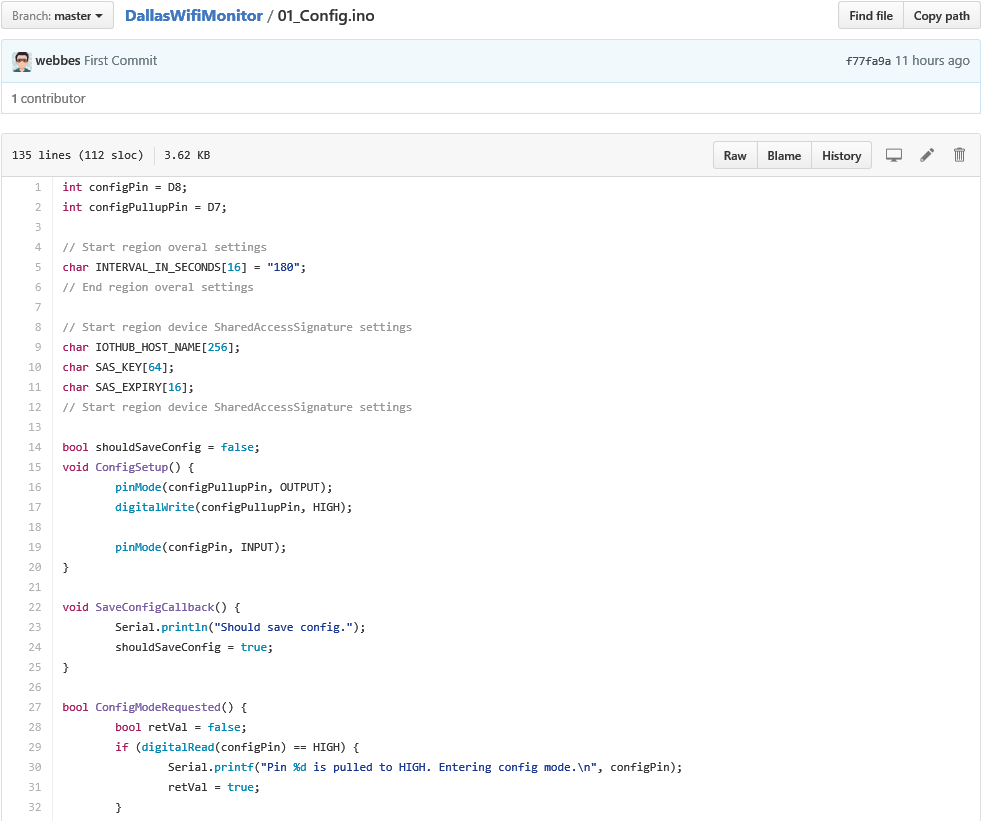


Figure 01\_Config.ino (first few lines)

When you read the code carefully you’ll notice that we configure a configPullupPin and a configPin. The configPullupPin pin is always set to high on setup and in the ConfigModeRequested() method we return true if the configPin reads high as well. In effect this means that if we somehow connect pin D8 and D7, the device will start in config mode.

The reason we selected pin D7 and D8 is because these pins are located next to each other. Connecting them is as easy as putting a jumper on these two pins.

Please place the jumper on the two pins and we are done with the wiring!

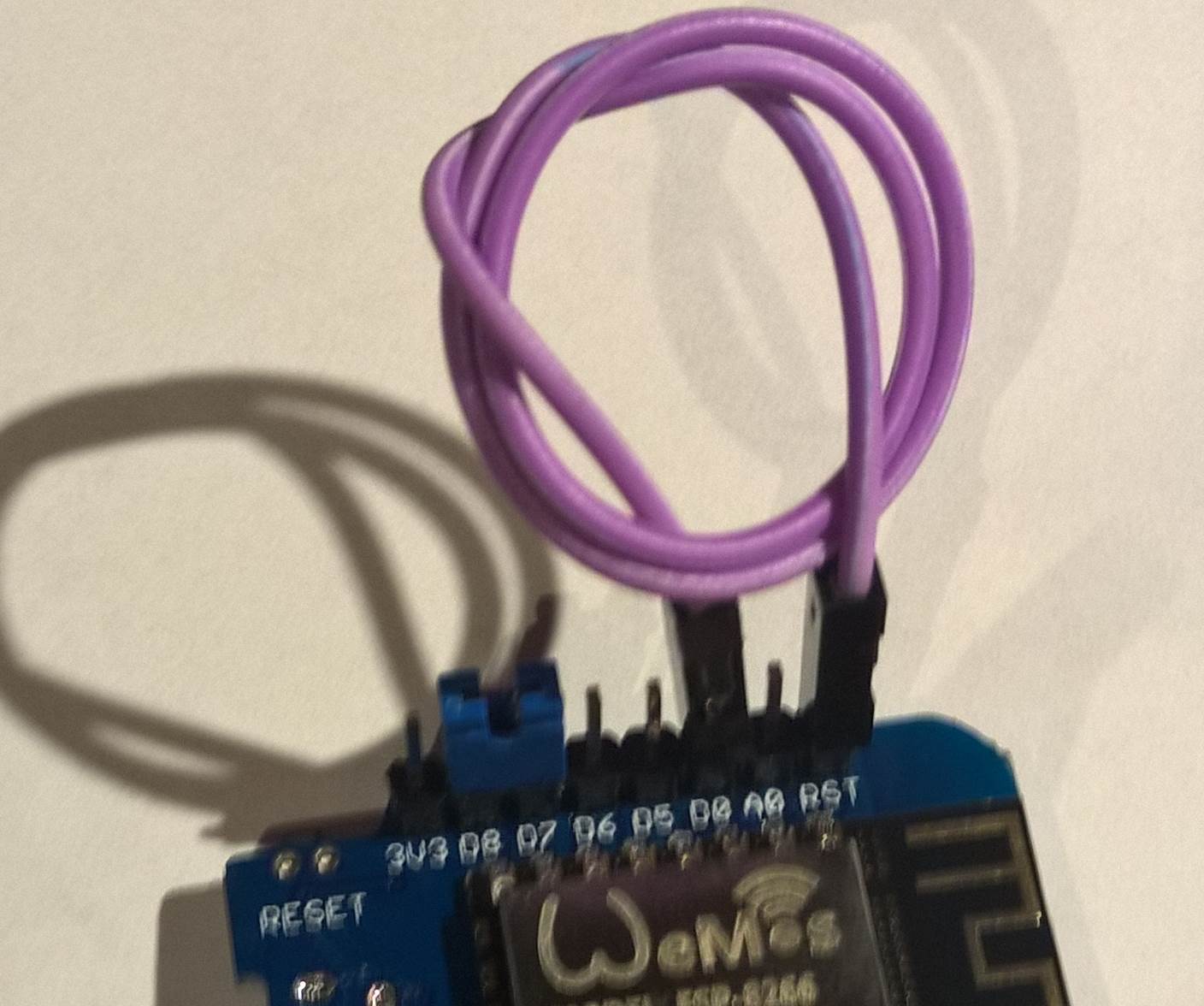


Figure Jumper places on pins D7 and D8

# Configuring the network settings

After validating your wiring with one of the instructors it is now time to turn on your device. Simply connect the USB cable to the WeMos D1 Mini R2 and supply power with the USB adapter.

The device is now started, but… how to configure? I don’t see a keyboard, mouse or screen attached!

Hmmm…. It’s a WiFi device, so let’s see if something appears if we open our network settings.

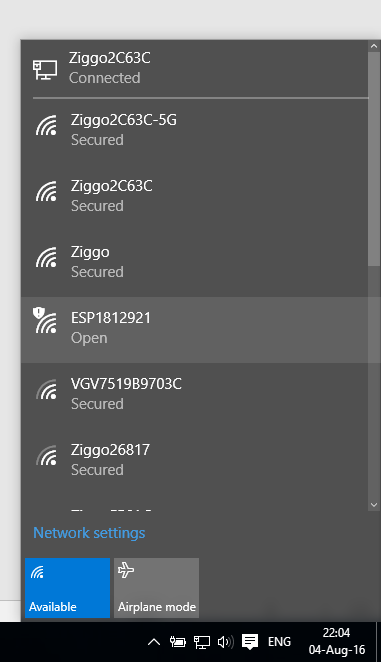


Figure Our device shows up in available networks

Every ESP8266 has a unique identifier. Our program configures the device as an access point with the name ESP<UniqueID> when the device is setup in config mode. The ID of your device can be found on a piece of paper that was also included in the bag. (You must have wondered what to do with it, didn’t you?)

Simply connect to your device and see what happens.

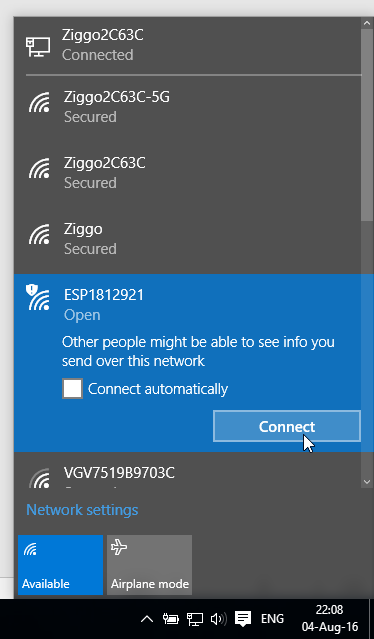


Figure Connect to your device

A browser should open up and navigate to <http://192.168.4.1> of not (Edge browser ☹) please enter the address manually(or click this [link](http://192.168.4.1)) and the configuration screen should show up.

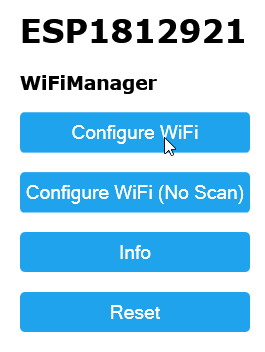


Figure The configuration interface in your browser.

Now click “Configure WiFi”.

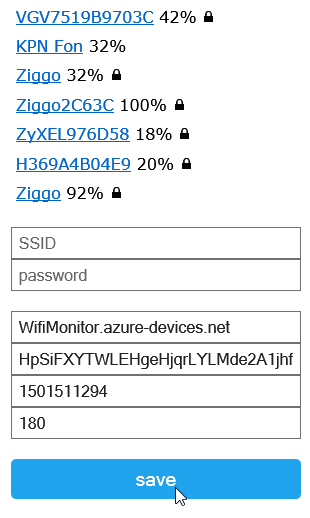


Figure WiFi configuration interface

Ask one of the instructors for the SSID and password in your location. You can simply click the required SSID and enter the password.

!!!DO NOT CHANGE ANY OF THE OTHER SETTINGS!!!

After you clicked “save” you will see the following screen:

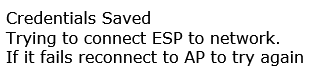


Figure Credentials saved

Just wait for about 30 seconds and if all went well, you should have lost your connection to the device.



Figure Lost connection

The reason you lost the connection is that the device automatically reboots after successful configuration. This actually means everything went well. If you did not lose the connection, please try to configure the network again!

Disconnect the USB adapter to power down the device and remove the jumper between D7 and D8. If we now power up the device again, we should not get into config mode.

So power up the device and verify if the device its network does NOT show up.(If the device cannot connect to the network, it will automatically move to config mode again.)

# Verify if your device is sending information to the cloud

The device is now up and running, but how do we know it is actually working? As the device is programmed to send temperature values to the cloud, we must connect to the cloud and see if our messages indeed arrive there.

Luckily for us, there is the Device Explorer application. So please head over to github again and download and install the latest release of device explorer:

<https://github.com/Azure/azure-iot-sdks/releases>

When you open Device Explorer, it will ask you for a connection string. Copy and paste the following connection string and click “Update”.

HostName=WifiMonitor.azure-devices.net;SharedAccessKeyName=student;SharedAccessKey=VdA+/NTX7yVDxQt8/jMkwkMsb2qgv1LLX6ldLec3fuo=

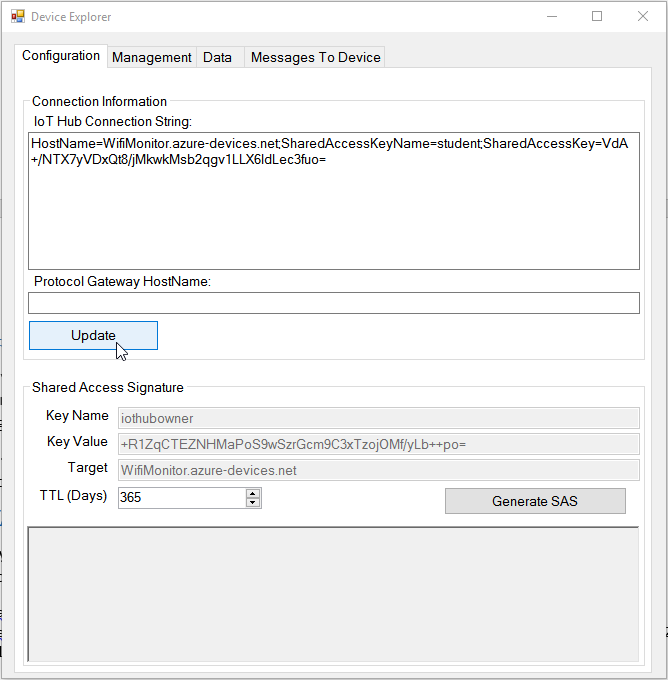


Figure Paste the connection string and click update in device explorer

Click the tab “Management” tab and notice your device is listed.

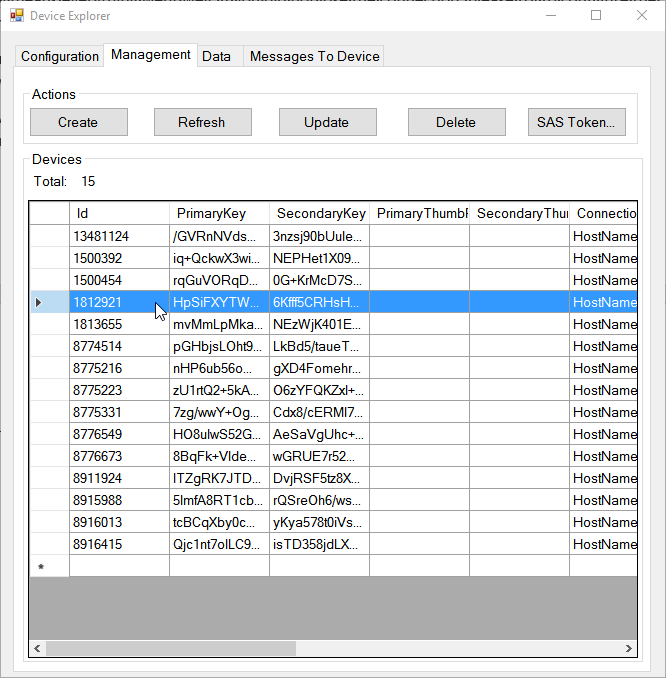


Figure Notice your device is listed

Click the “Data” tab, select your device from the dropdown and click “Monitor”.

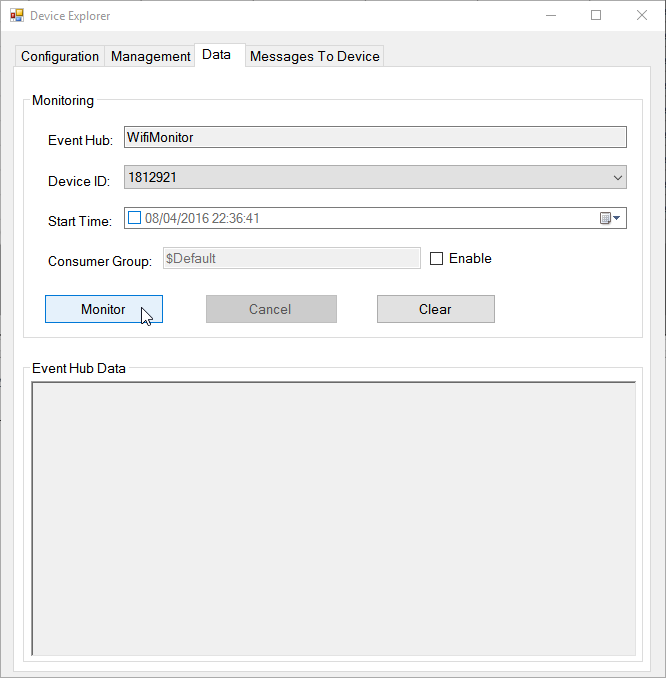


Figure Slect your device and click Monitor

You can now wait for a maximum of 3 minutes OR press the small reset button on the site of your device. After a few seconds, if everything went well (fingers crossed), you should see that the Azure IoT Hub just received a message that was sent by your device!

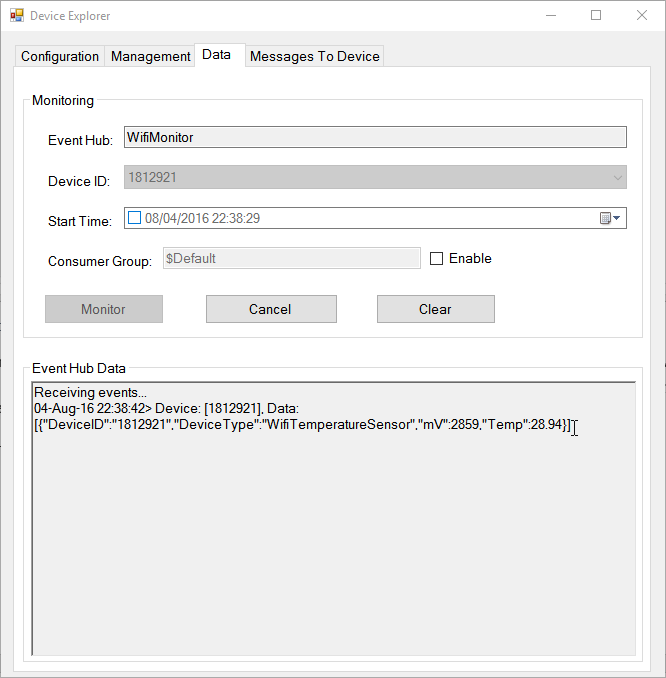


Figure Message received! (Yes, it's hot in here...)

# Wrap up

You’ve wired up your device, configured your device and the cloud now receives message from your device. You’re a hero!

[](https://www.google.nl/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=0ahUKEwiPlOrk0KjOAhVCaxQKHTYrAh0QjRwIBw&url=http://felskids.org/event/weekly-theme-super-heroes/&bvm=bv.128617741,d.d24&psig=AFQjCNEL5sT_1MI856kCNt8Jgf5x2Kr43Q&ust=1470429993468906)

In later labs you might need to reconfigure your device to send its messages to your own IoT Hub. Simply put the device in config mode again by placing the jumper on pins D7 and D8. I will miss the device in my hub though and sure do hope that you will take good care of him.

In the next lab, I’ll teach you how to program your own device. If you are not into programming or short on time. No worries. All other labs during this day can be completed without finishing this next lab. Enjoy!