Wesley Bakker

[Email address]

Abstract

During this lab you will learn how to program your device

IOT WORKSHOP Lab 2

Programming your device

# Goal

Your device comes pre-programmed for sending data to the Microsoft Azure IoT Hub. There is some room for improvement however. To be honest, I’ve been lying in Lab 1 ☹. The code that you find on GitHub, is not actually the code that is running on the device. On the device, we are running a previous version. The wiring is exactly the same, but the code is, ahum a bit different. You know how it works. Sometimes you just need to deliver and then, well…. **Sorry!!!** The code isn’t that good and it contains some bugs. However, to make up with you we will upload the latest and greatest version to your device during this lab.

# The development environment.

First of all, we need to set up our environment before we can actual program our device. So please download and install all the following software using the default values if you have not already done so.

1. Arduino IDE (<https://www.arduino.cc/en/Main/Software>)
2. Visual Studio IDE (<https://www.visualstudio.com/>) - Community edition suffices, but make sure to select the Visual C++ features when installing.
3. Visual Micro Visual Studio Add-In (<https://visualstudiogallery.msdn.microsoft.com/069a905d-387d-4415-bc37-665a5ac9caba/>)

To build solutions for any ESP8266 device, such as the WeMos D1 Mini R2, we need to add the “board” to our Arduino environment. This board however is not part of the standard boards library. So we first need to tell Arduino where it can find the software for our board.

Please open Arduino and click File->Preferences

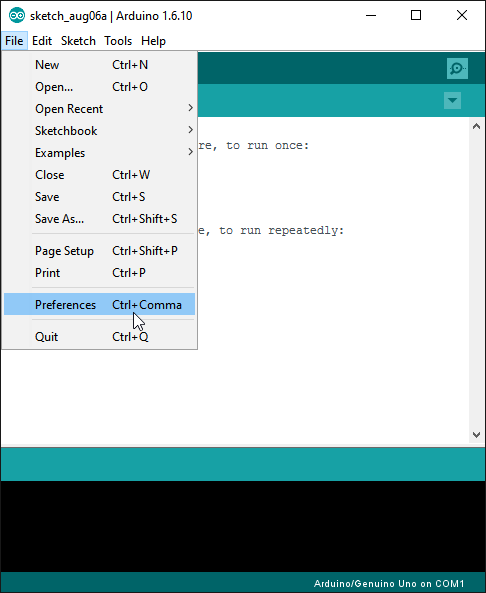


Figure Open the preferences screen

Add the following url in Additional Boards Manager URLs:

<http://arduino.esp8266.com/stable/package_esp8266com_index.json>

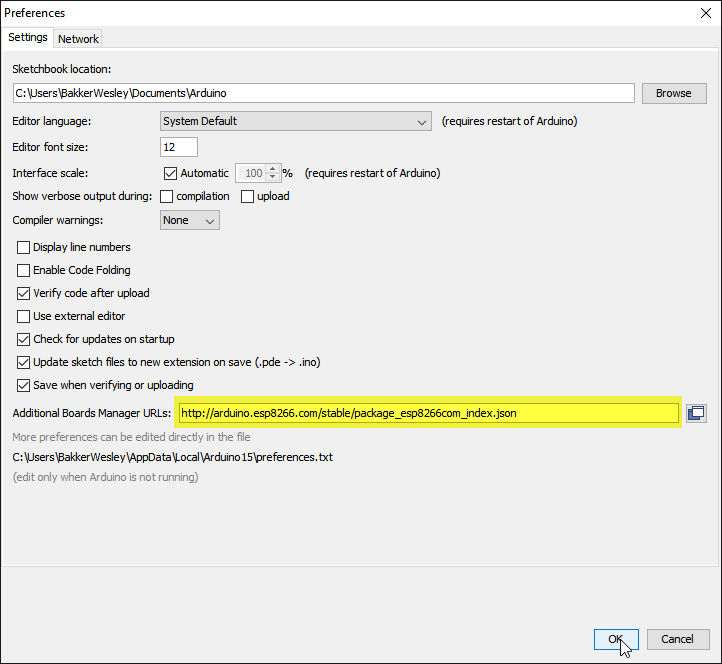


Figure Add an additional board manager url

After we told the Arduino IDE where to find the software for our board we can add it through the Boards Manager.

Please open Arduino and click Tools->Board->Boards Manager

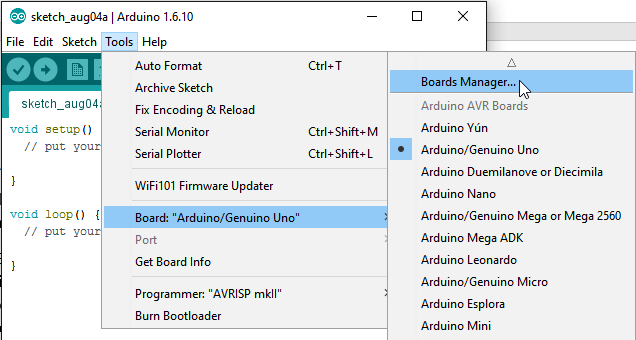


Figure 3 Opening board manager in the Arduino IDE

Search for “ESP8266”, install the ESP8266 board, close the board manager, and close the Arduino IDE.

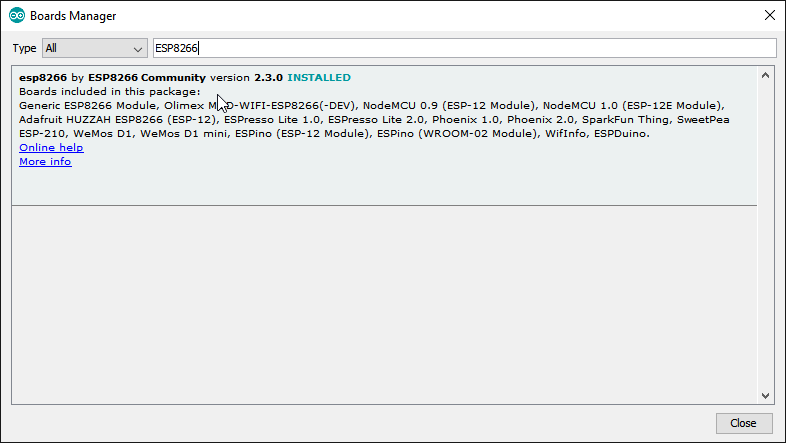


Figure 4 Search for and install the ESP8266 board

The environment is now ready to program almost any ESP8266 base device! Good job!

# Download and extract the source

After everything has been installed, we can download the latest and greatest version of the software for our device from GitHub. Please navigate to the GitHub page where the solution is at: <https://github.com/webbes/DallasWifiMonitor> and download the solution as a zip.

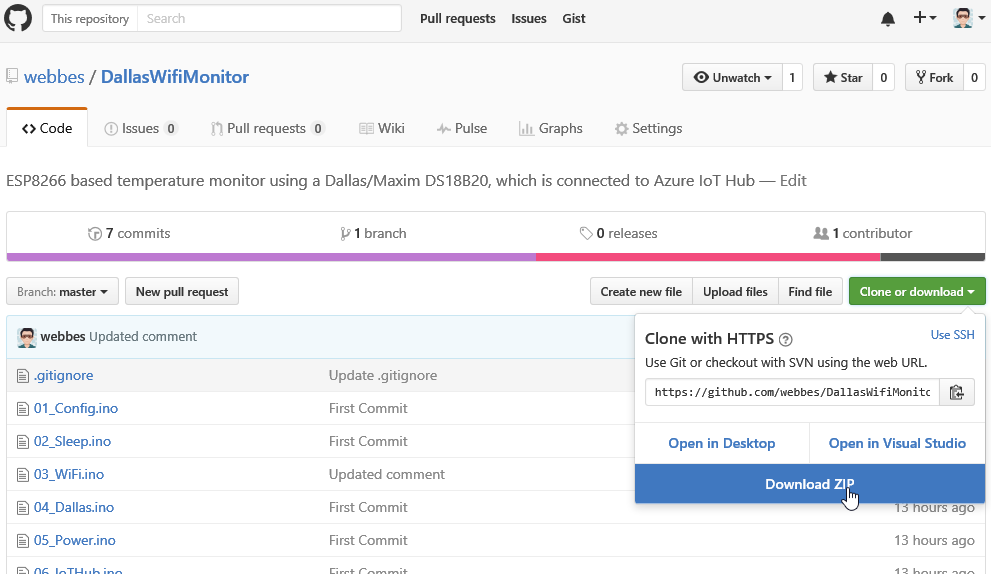


Figure 5 Download the source as a zip

You can unzip the solution to the Arduino folder which should be located in your “documents” folder. The Arduino folder should already contain a folder called “libraries”.

**The name and location of the folder is important. So extract it as “DallasWifiMonitor” and drop the “-master” extension.**

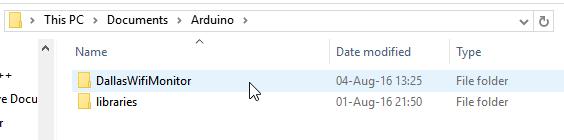


Figure 6 Our extracted solution in the "documents/arduino" folder

# Building the software

Before software can be uploaded to the device, we first need to compile/build the software.

Open Visual Studio and use the menu to open an Arduino Project.

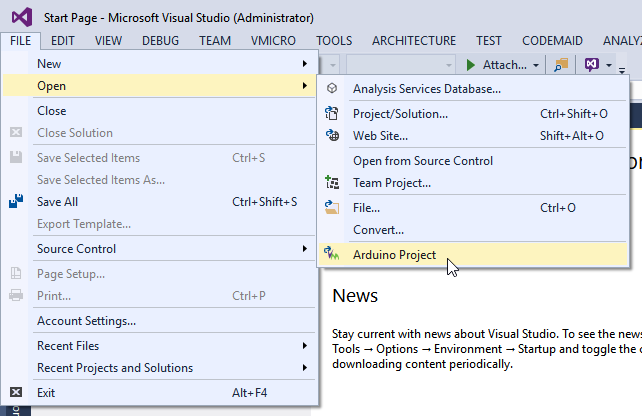


Figure 7 Opening an Arduino project inside Visual Studio

Navigate to our solution folder and select the DalasWifiMonitor.sln solution.

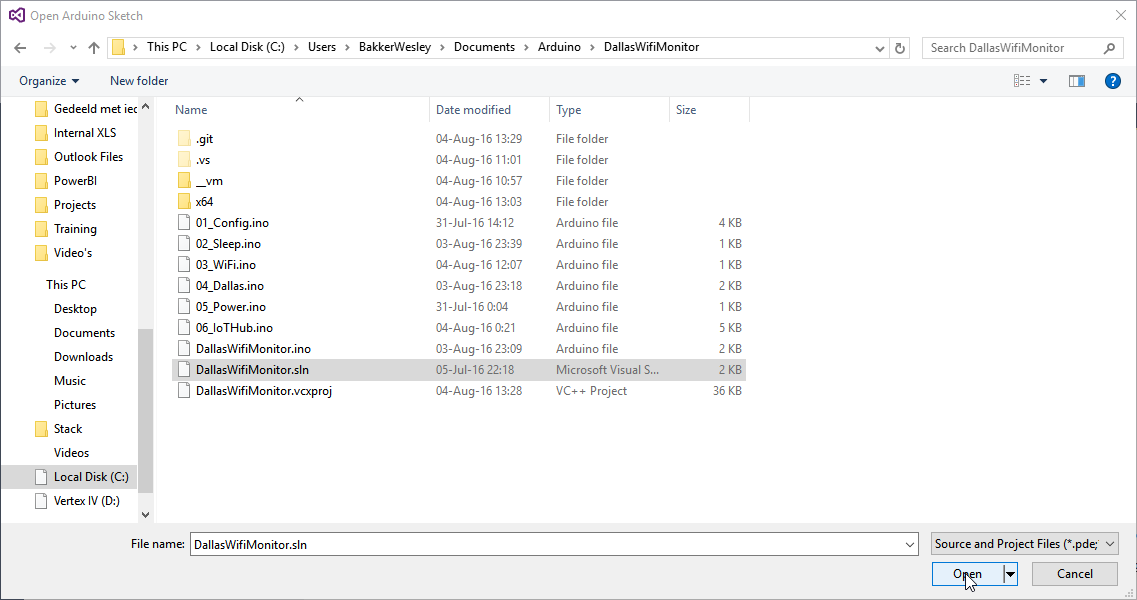
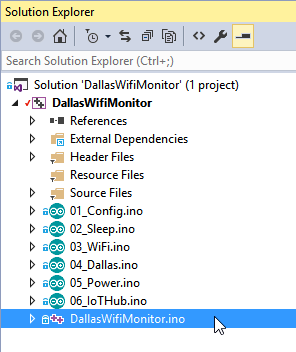


Figure 8 Select the solution and click Open

In Solution Explorer select the DallasWifiMonitor.ino file.



Now carefully look at the first lines of this file:

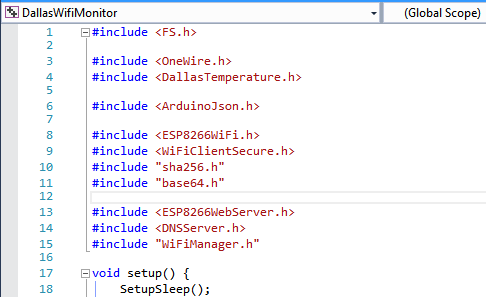


Figure 9 DallasWifiMonitor.ino (first few lines)

Our solution makes use of several libraries. Libraries can be seen as code we need, that has already been written by other people. Libraries save us a lot of time and you will use them very often. As our code makes use of these libraries we have to make sure we have these libraries available or our code will not compile.

Some of these libraries can be found and very easily installed through the official library managers in the Arduino IDE or the Visual Micro Plug in. Both these managers allow access to the exact same libraries. It is just a matter of preference which library manager you choose to use.

If you cannot find your specific library through one of these library managers, you have a very big chance to find your library somewhere on GitHub. It is smart to search for a library before you buy a sensor. If no library is available for your sensor, it might cost you a lot of time before you can use it.

In this lab we will install some of these libraries through the Arduino IDE, some through the Visual Micro Add-in and we will download another one from GitHub.

## Installing a library through the Arduino IDE

Open the Arduino IDE and click Sketch->Include Library->Manage Libraries

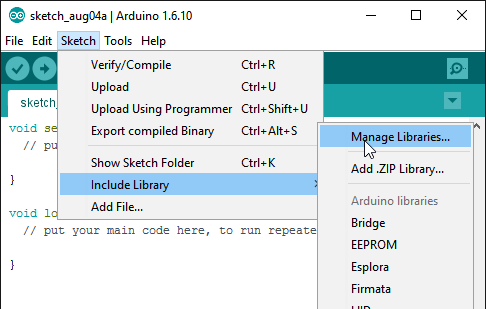


Figure 10 Open the library manager in the Arduino IDE

Search for “ArduinoJson” and install the library.

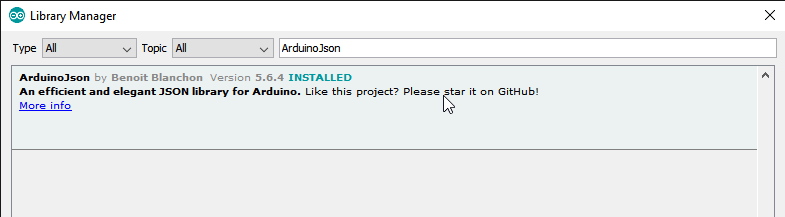


Figure 11 Install the ArduinoJson library

Search for “OneWire” and install the library.

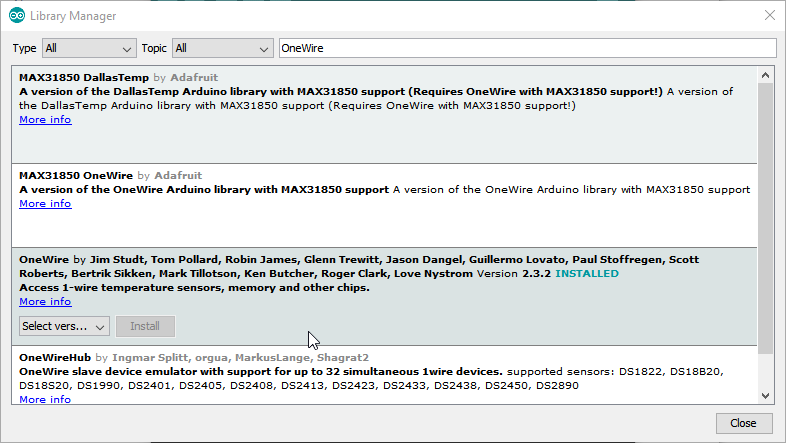


Figure 12 Install the OneWire library

## Install a library through the Visual Micro Add-in

In Visual Studio click VMICRO->Visual Micro Explorer

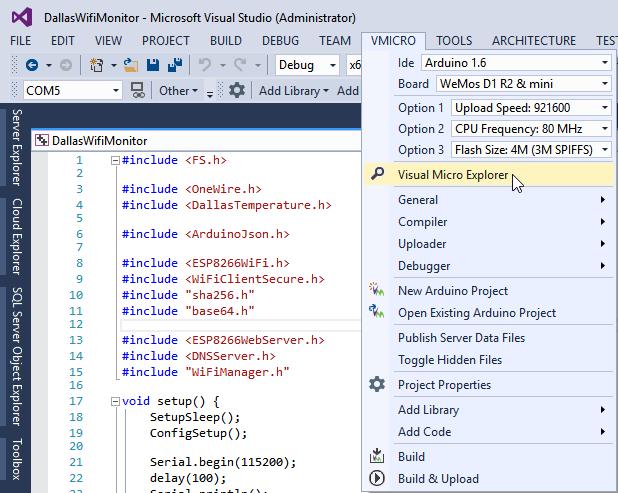


Figure 13 Open the Visual Micro Explorer

Open the Manage Libraries tab and search for WifiManager. Expand WifiManager by tzapu and select version 0.12 as this code has be written and tested with version 0.12. (If you are adventurous you can always try to build and test with the latest version.)

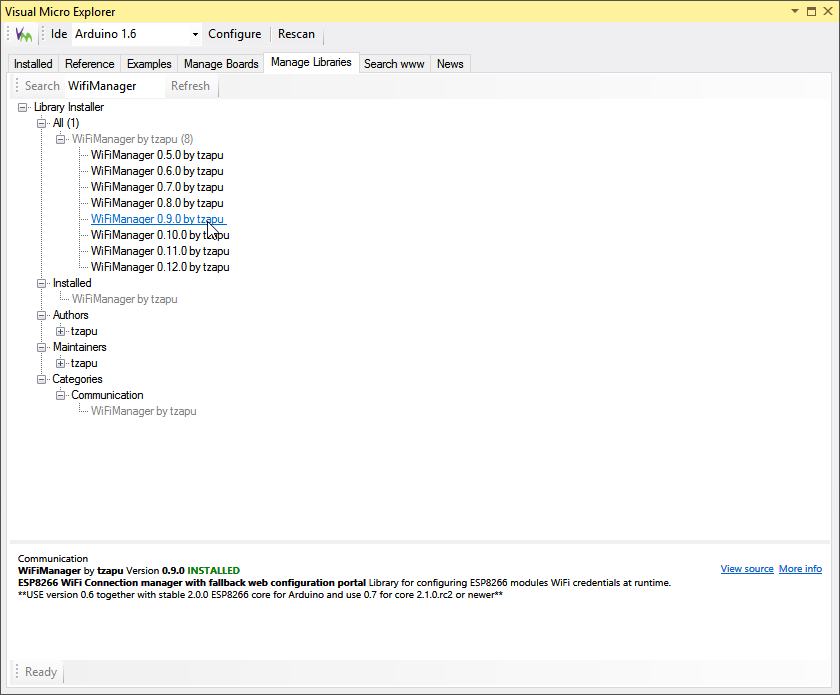


Figure 14 Installing a library with Visual Micro Explorer

## Install a library from GitHub

Some libraries are not in the standard repository or the libraries in the repository do not support that one feature you need for your specific sensor. Good chance that you can find a required library somewhere on GitHub. I had that issue with the temperature sensor. The version in the library manager’s repository contained a few bugs that really bothered me, so I went to GitHub, forked the code, made some changes and now use that library in my solution. Let download the library from GitHub.

Navigate to: <https://github.com/webbes/Arduino-Temperature-Control-Library> and download the source as a zip.

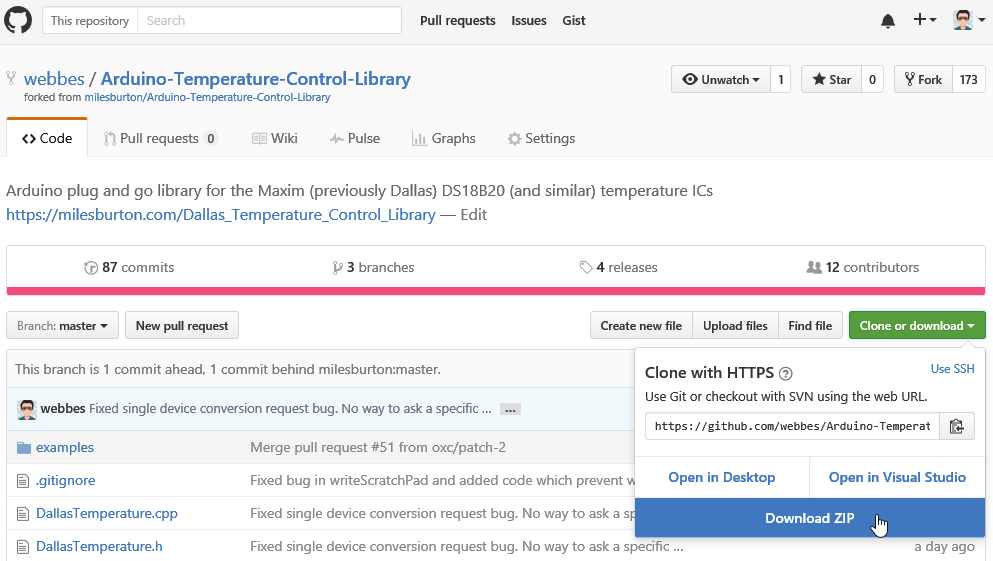


Figure 15 Download ZIP

But where do we have to extract that folder? Hmm…. Remember that Arduino folder in our Documents library? It contained a folder called “libraries”. Let’s have a look over there.

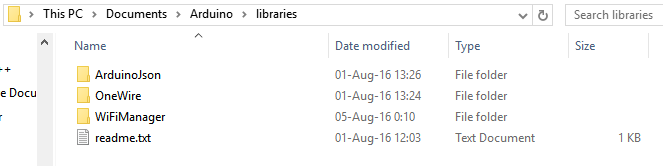


Figure 16 There are my libraries!

Extract the zip file and copy the folder it contains to the libraries directory.

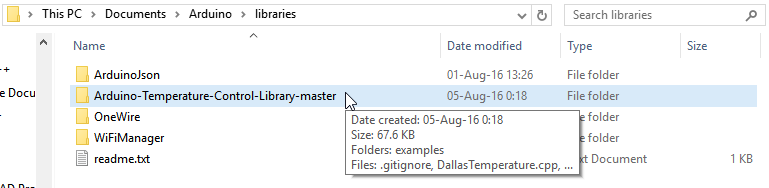


Figure 17 "Installing a library from GitHub"

With all libraries installed we are almost ready to build the code. We first need to tell Visual Micro to scan the “libraries” folder again, to make it aware of our newly added libraries.

In Visual Studio click VMICRO->Visual Micro Explorer and then click “Rescan”.

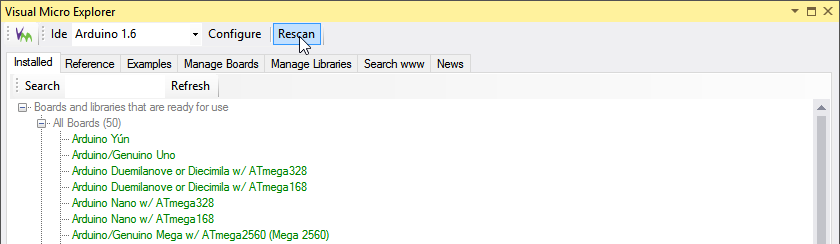


Figure Rescan for new libraries

Close the Visual Micro Explorer.

When I build my code I prefer to get Verbose information on how everything is going. So let’s configure Verbose output as well.

In Visual Studio click VMICRO->Compiler->Verbose

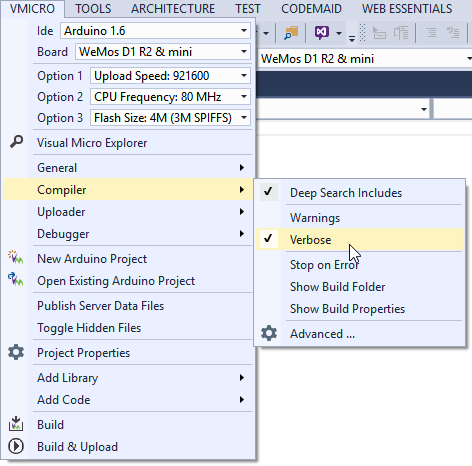


Figure Configure verbose debug output

Ok, that was it. We can now finally build our code!!!

In Visual Studio click VMICRO, verify if your board is correct and hit, Build

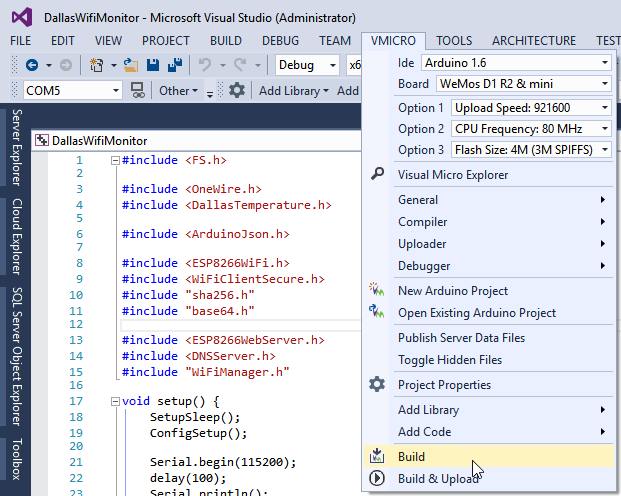


Figure 20 Build your solution

If all went well the Output window should display the size of your program and the memory used. If you see any errors, please contact an instructor.

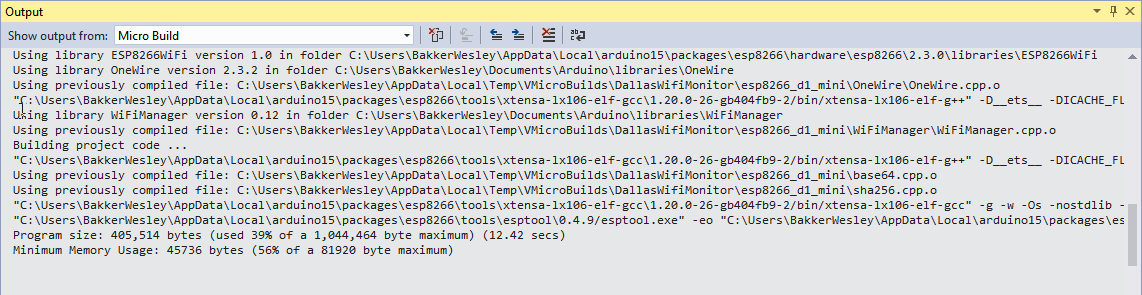


Figure 21 Successfully build the code

# Uploading the code to our device

And now comes the tricky part. Getting the code to run on our device. Let’s take it step by step again. First we need to connect the device to our development machine. That’s easy enough. Let’s plugin the USB cable to one of our USB port.

If your operating system complains about drivers, you can download them over here: <http://www.wemos.cc/downloads/>

If all goes well however you will notice that your machine now has a new COM port in the Device Manager.

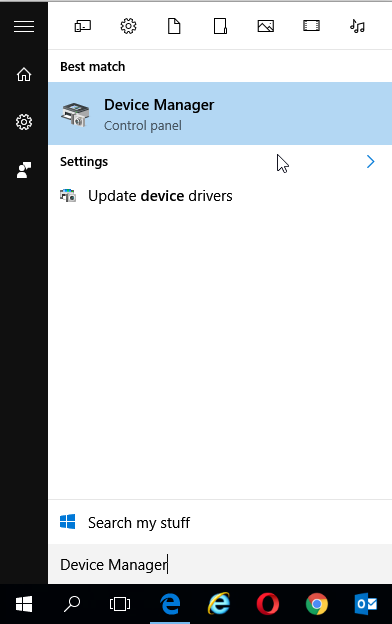


Figure 22 Open Device Manager

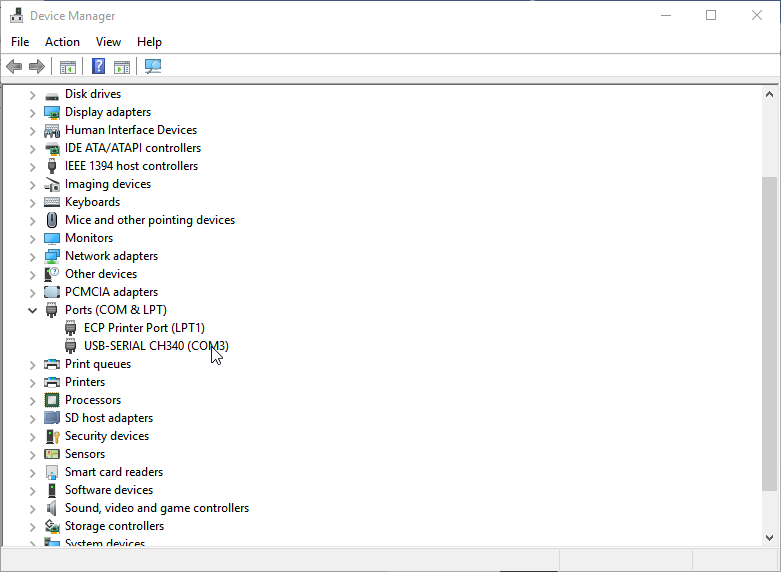


Figure 23 We have a new device called USB-SERIAL CH340 (COM3)

Please take a note of the COM port that is assigned to the device. As we will need this to upload our code.

In Visual Studio we can now try to Build and Upload our code using Visual Micro.

First select the noted COM port **in the VMicro toolbar** in my case that’s COM3 and make sure your device is selected as board again.

Then build and upload your code.

Click VMICRO->Build & Upload

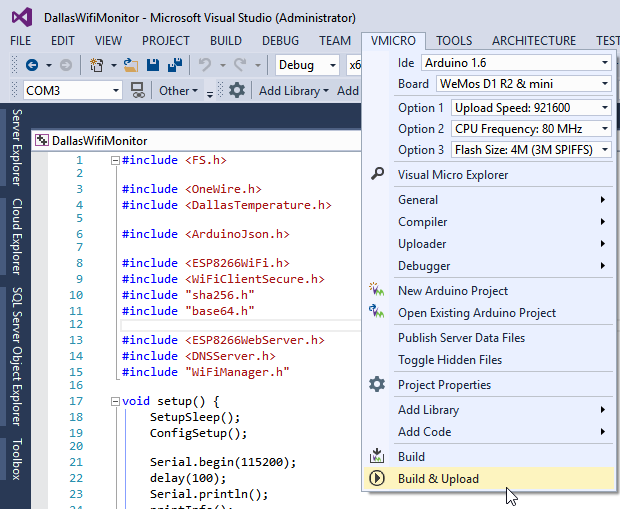


Figure 24 Build & Upload

Unfortunately, this almost always fails on some machines ☹!

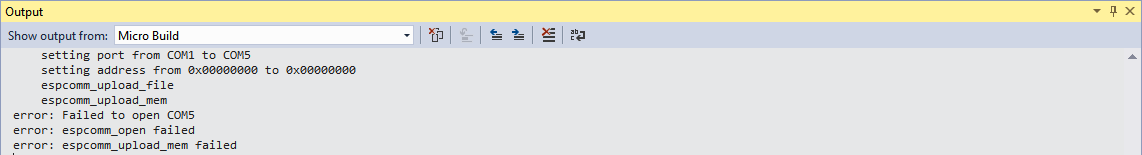


Figure If your output window contains error messages... it failed

If you don’t see the error messages, but success (lucky bastard), you can continue to the last part of our lab: “Debug your code”.

Don’t worry if you do see the error messages. That is the risk when you are living on the edge and want to code your own devices. We will not give up however and try to apply a little trick to make it succeed.

First need to scroll up in our output window and find and copy the command that is being executed to upload the program to our device.

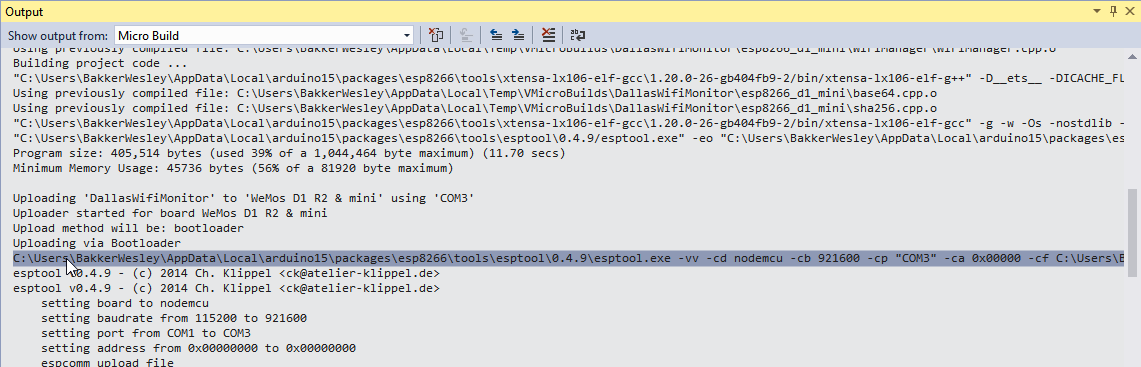


Figure 26 Find and copy the command that tries to upload the compiled program to our device

With the command on our clipboard we will open a command prompt.

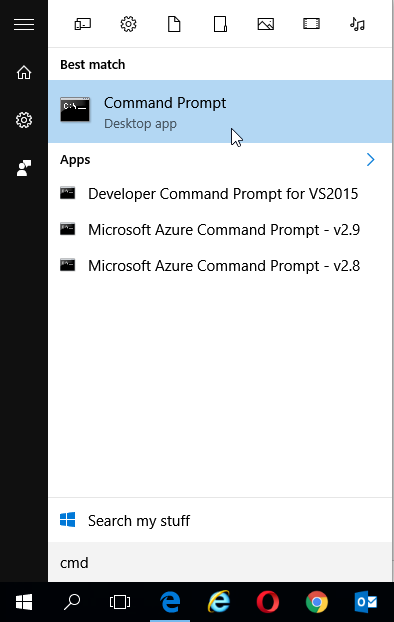


Figure 27 Open a command prompt

We can now paste the copied command in our command prompt. This will try to upload your code, without recompiling again. This saves us a lot of time! Unfortunately, it executes immediately and just like before, it won’t succeed.



Figure 28 Past the command in a command prompt

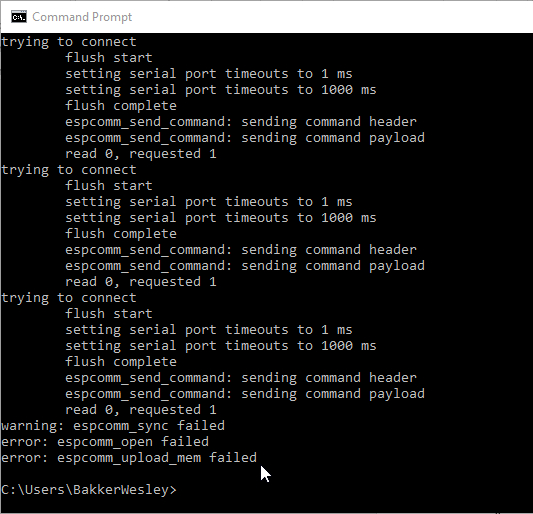


Figure 29 And it won't succeed

Hang on as we are almost there. Like I said, we need to apply a little trick to upload our code. The steps are as follows:

1. Hit the UP arrow on your keyboard, which will display the command once again, but don’t hit ENTER just jet.
2. First press the reset button on the device and keep it pressed
3. Then hit ENTER on the keyboard and right after that, release the reset button

You should see a few failed attempts and then…….VICTORY!!!!!

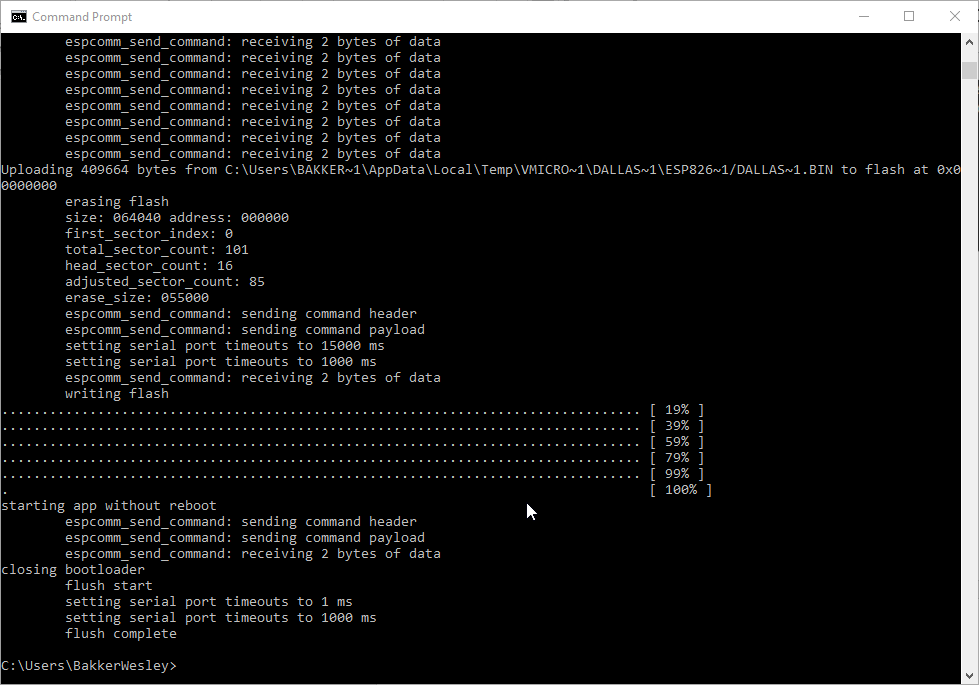


Figure 30 This is what victory looks like!

You might need a couple of tries with the trick, but you will succeed!

You’ve now programmed your own device! Wow, you must feel like a hero again, but… no cigar until the fat lady sings.

# Debug your code

We could, just like in the first lab, see if our messages arrive at the cloud. But when programming, you might want to see a bit more information. Let’s connect to the device and see what it’s actually doing.

Throughout the code you should notice a lot of Serial.print and Serial.println commands. This actually sends messages to the …… you guessed it, serial port. And yes, we can connect to the serial port. So let’s do that and see what happens.

In Visual Studio you can click on that strange looking icon next to your COM port selector.

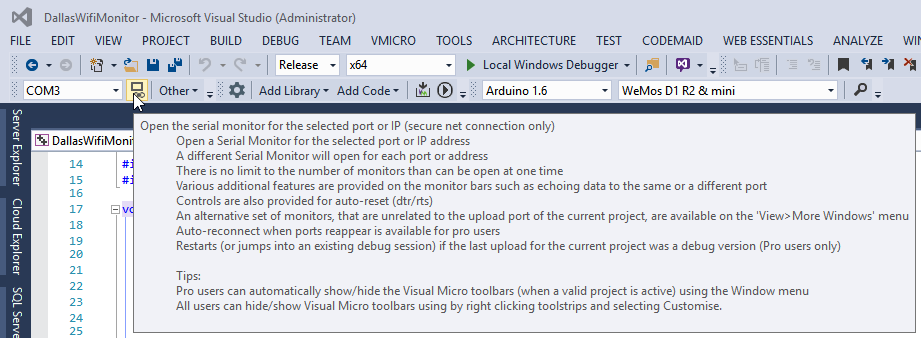


Figure 31 Open the Serial monitor

Then we need to ensure that the bit rate is equal to what we’ve programmed.

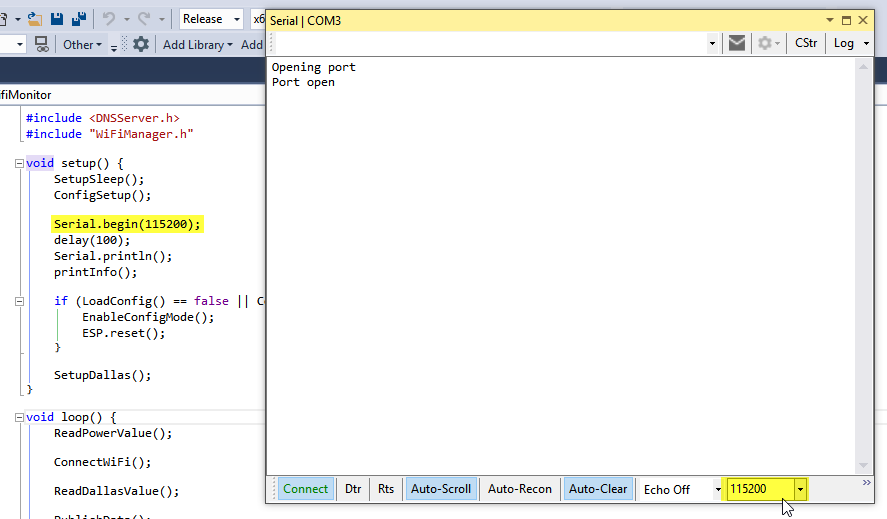


Figure 32 Verify the bit rate

And then after a while (or after pressing the reset button again ☺)

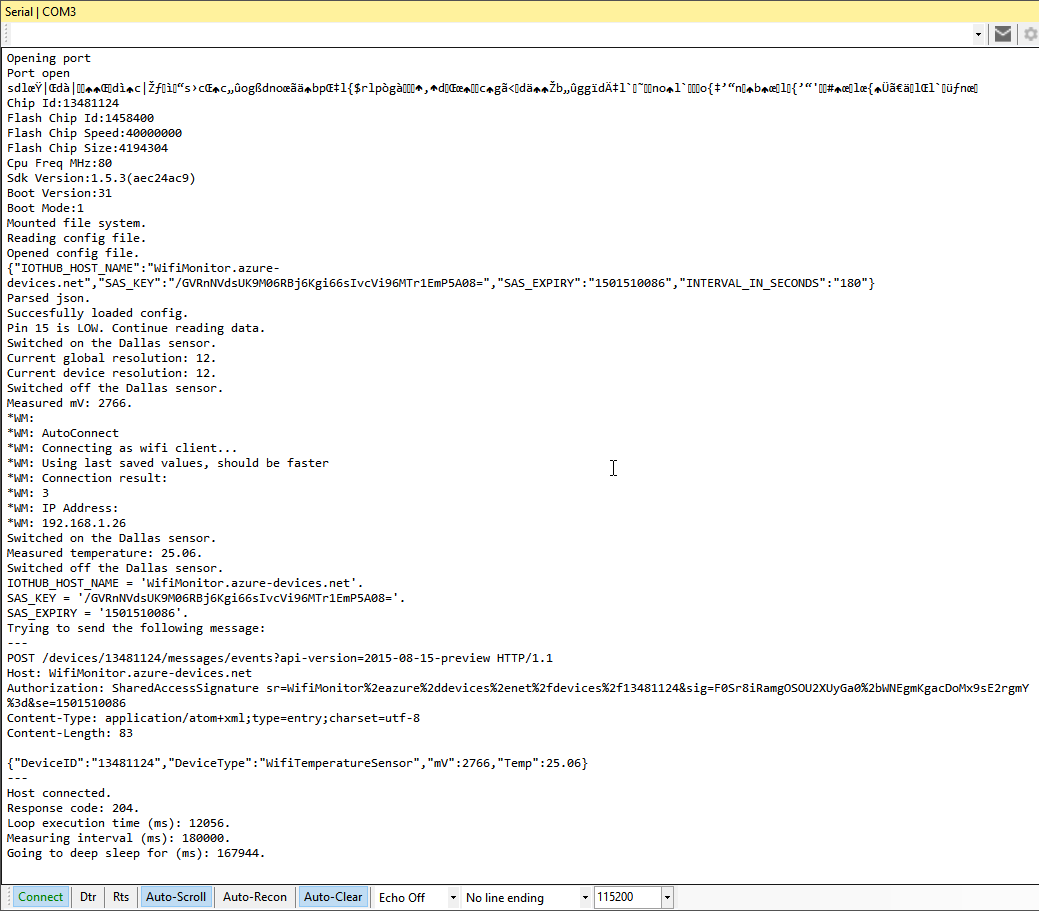


Figure 33 We can actually see what is going on

The response code here tells me that sending the message was successful. So it all works.

# Wrap up

You’ve setup your development environment and uploaded your first program to your own device! You must feel an even bigger hero than before don’t you?

[](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)

Figure 34 Enlarged heroes picture

Honestly, this is where it all starts. You will encounter some tough moments. Trust me. Just stick to it and continue. Can you imagine how long it took me to find out the trick to upload code to this particular device? Can you imagine how amazed (read frustrated) I was when the one time my temperature readings went well, and the second time they were way off (85 degrees Celsius). I just pulled through and a very nice working and configurable device, sending its messages to the Azure IoT Hub is the result.

We are however not done for today. The next session will guide you through on how to configure the Azure Cloud to receive and process your messages. Cause what use is sending messages if nobody is listening for them? The last session of today will guide you through on how to analyze the gathered data and create awesome looking reports with it.

Enjoy the rest of your day and please promise me to take care of your device.