Micropython IoT Hackathon Documentation

Release 1.3.4

Jeff Fischer

Aug 09, 2017

Contents

Abstract:

Due in large part to the availability of cheap, low-power, internet-connected microcontrollers, the Internet of Things is taking off. Python developers can get in on the excitement with MicroPython, an implementation of Python 3 that runs on very small devices with no operating system. MicroPython provides the standard Python REPL (read-eval-print-loop) interface, so you can interactively develop and debug applications on these small devices.

This document provides instructions for a hackathon/class centered around MicroPython. In the class, students will learn some basic electronics, wire up some sensors to a low-power wireless controller board (based on the ESP8266 microcontroller), load the MicroPython firmware, and interactively write simple applications to read from the sensors. It also discusses how to connect to other systems via the MQTT messaging protocol and exchange ideas on larger projects that can be built at home for low cost with beginner-level knowledge.

This was original created for use at a San Francisco Python Project Night meetup. It is designed to be reusable for other classes as well as for individual study.

Contents:

CHAPTER 1

1. Introduction

Background

MicroPython[1] is an implementation of Python 3 that can on on small devices without a traditional operating system. It requires just 256k of code space and 16k of RAM. One of the microcontrollers supported is the ESP8266[2], a low cost microcontroller with a 32-bit CPU, a built-in WiFi radio, and a number of input/output ports useful for interfacing with the physical world. Since the ESP8266 can run off a small lithium ion battery, it makes a great remote sensing and control platform. Complete development boards using the ESP8266 run anywhere from $3 to $16, cheap enough that you can deploy multiple systems.

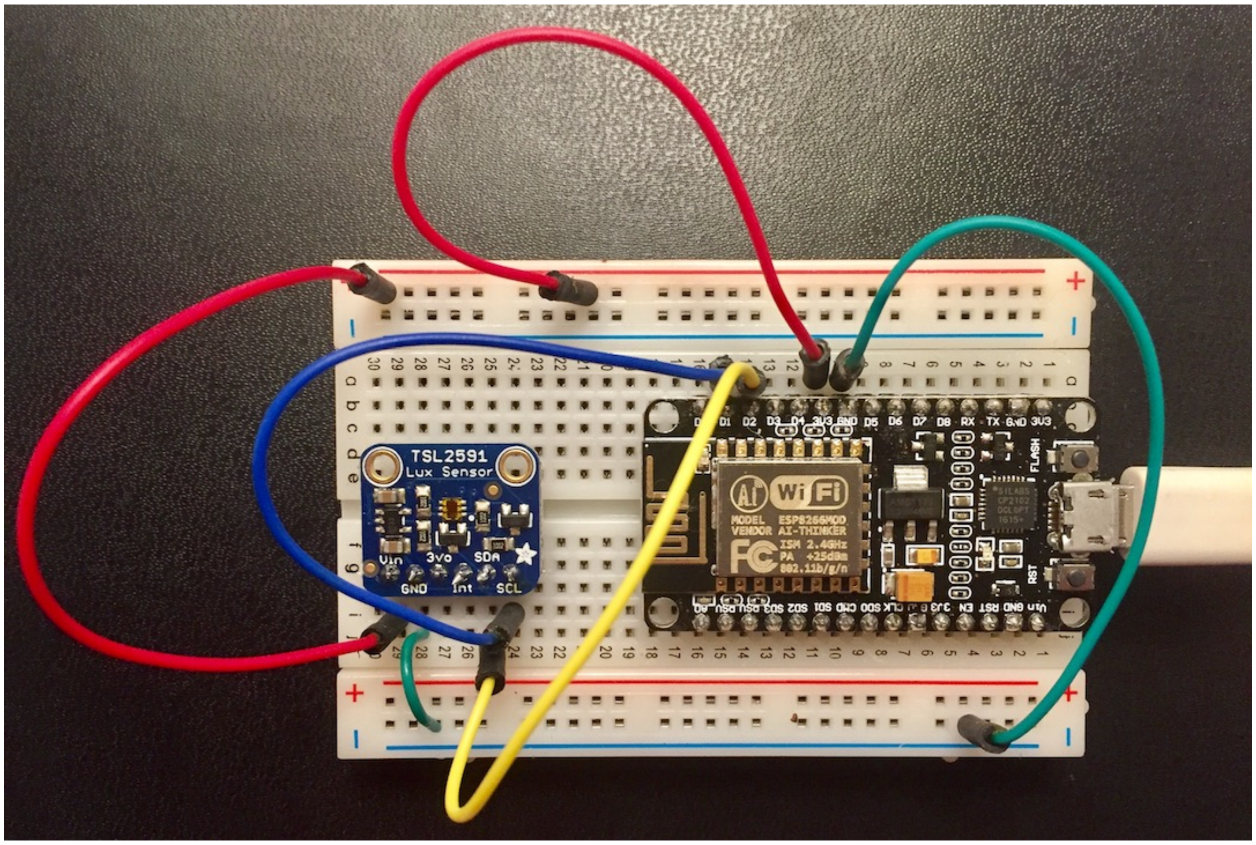
Hardware Overview

We will build a wireless system that combines the ESP8266 with a light sensor (the TSL2591 ambient light sensor from AMS). Each of these will be used via a *breakout board* – small printed circuit boards that combine multiple surface mount chips and provided easy connection points. The sensor and processor connect via a digital serial protocol called I2C[3]. This will involve connecting a total of four connections between the boards. We will facilitate these interconnections using a *prototyping breadboard*. The breakout boards can be plugged into the breadboard and then the wires stuck into the breadboard to connect the pins on the chips.

The breakout boards for the ESP2866 have a Micro-USB connector attached to a serial port on the processor. We will connect this to a laptop/PC, allowing for firmware updates, file transfer, and interactive communication.

Once the basic system is up and running, additional sensors can be added.

[1] http://www.micropython.org
[2] https://en.wikipedia.org/wiki/ESP8266
[3] http://cache.nxp.com/documents/user_manual/UM10204.pdf

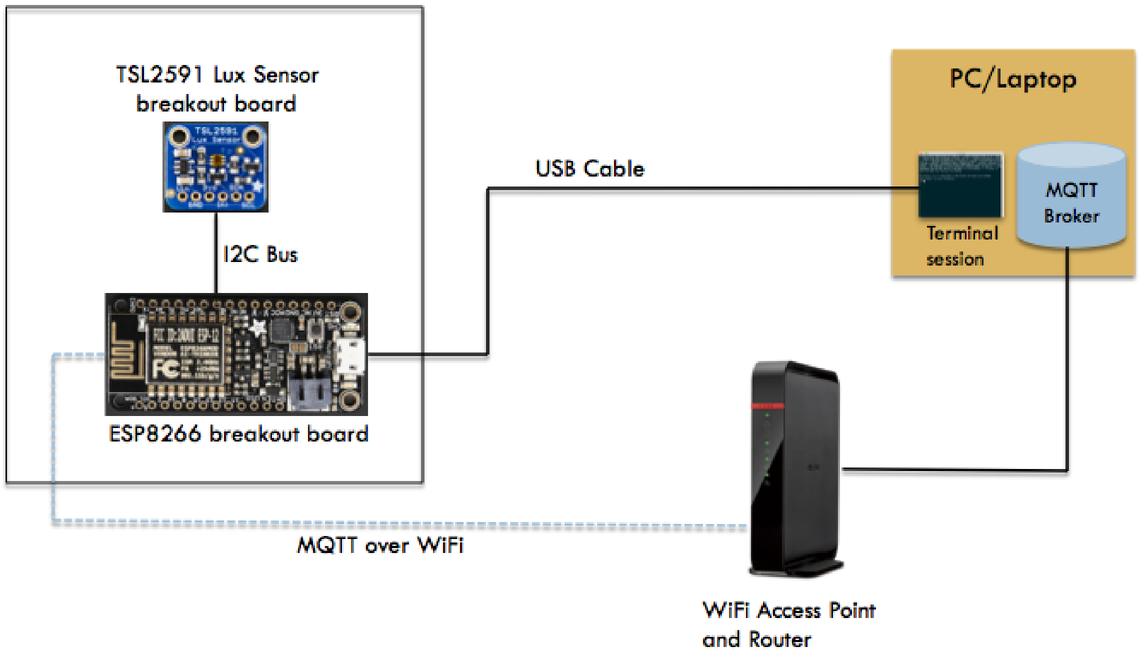


Software Overview

We will first update the firmware to use the latest version of MicroPython. Then, we can copy over a library module to read the light sensor. We can call it interactively via the Python REPL to sample the sensor. Next, we can copy over the ThingFlow framework and write a short program to periodically sample the sensor.

To use the system remotely, we will send messages over the WiFi radio. This involves setting up the credentials and sending messages via the MQTT protocol (MicroPython provides a library for this). We can set up an MQTT *broker* on the laptop/PC to receive these messages.

Finally, we will configure the ESP8266 to start our software upon reboot. Here is a block diagram of the completed system:



GitHub Repository

Example code and the source for this documentation are available in a GitHub repository at: https://github.com/jfischer/micropython-iot-hackathon.

Outline of this Document

We cover the construction and programming of the system starting with the parts you need and moving toward end-to-end applications. The following topics are covered:

- *2. Required (and Recommended) Parts and Tools* - the parts you will need to find/purchase and the tools required to build and debug the system.
- *3. Hardware Assembly* - connect a light sensor up to the ESP8266.
- *4. Software and Firmware Install* - install firmware and interact with the board over a USB connection.
- *5. ThingFlow Application* - use the ThingFlow framework to build a simple light sensor sampling application.
- *6. Messaging with MQTT* - use the MQTT protocol to send our sensor events over the wireless network.
- *7. Projects* - we sketch out some follow-on projects involving additional sensors and larger applications. This is the jumping point for you to be creative and try out your own ideas.
- *8. Teachers' Notes* - some notes for instructors who are using the material as a template for a class.
- *9. Quick Software Setup* - for fast software and firmware setup.

OK, now let's get started with the *parts and tools* that you need.

CHAPTER 2

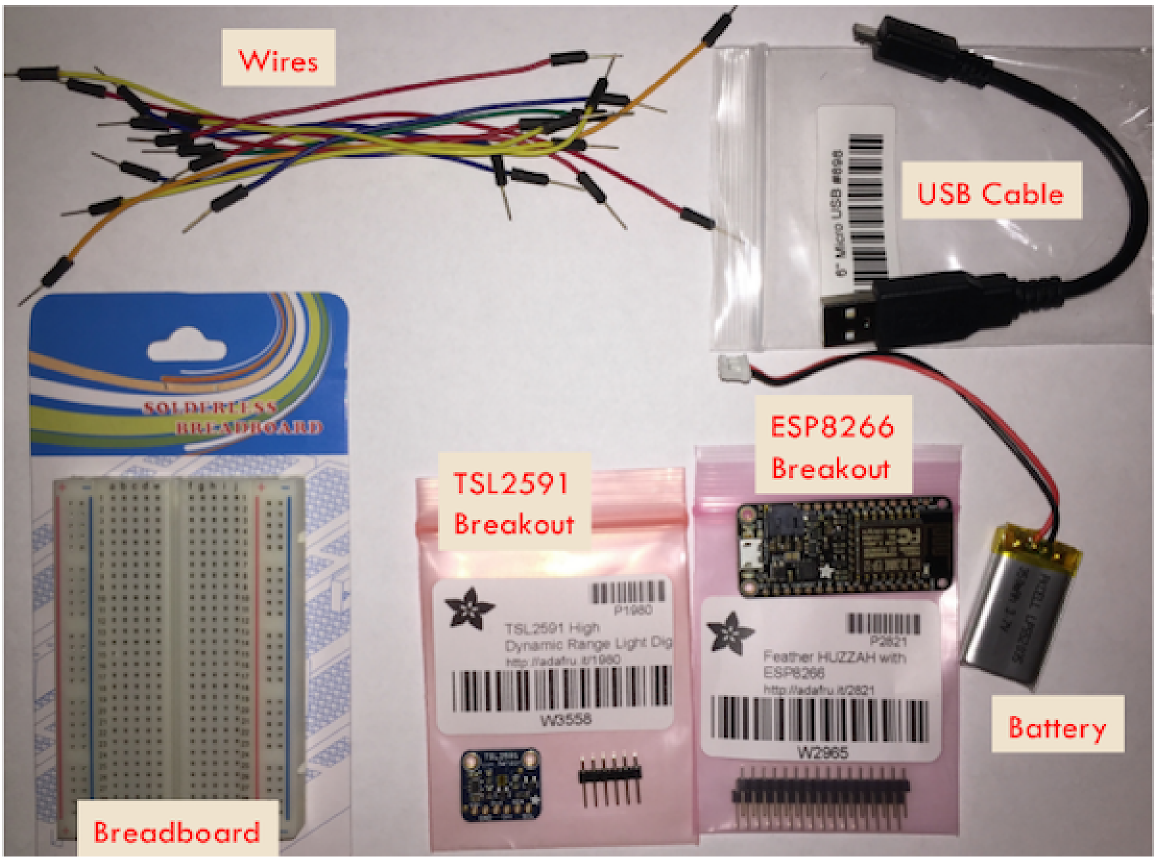
2. Required (and Recommended) Parts and Tools

Parts

The table below lists the parts required for assembling our system:

Part	Adafruit Part	Alternative Part(s)	Price Range
ESP8266 Breakout Board	Adafruit Feather HUZZAH with ESP8266 WiFi (part 2821)	Various NodeMCU boards	$4.00 - $15.95
TSL2591 Breakout Board	Adafruit TSL2591 Digital Light Sensor (part 1980)	N/A	$6.95
Lithium Ion 3.7v 350mAh battery	Lithium Ion Polymer Battery 3.7v 350mAh (part 2750)	Any 3.7v with JST-PH connector (for Adafruit ESP8266 board only)	$6.95
Half-size breadboard	Half-size breadboard (part 64)	Various	$2.00 - $5.00
Micro-USB to USB cable	USB cable - 6” A/MicroB (part 898)	Any with data connection	$2.95
Breadboarding wires	Breadboarding wire bundle (part 153)	22-26GA solid core wire	$6.00
Total			**$21.90 - $43.80**

And, here is a picture of the parts:



Details

Here are some details about each part. If you are looking to save money, see the options discussed for each part.

ESP8266 Breakout Board The board from Adafruit is expensive, but high-quality and well-documented. It also has a built-in battery connection and can charge the battery via the USB. You can also use boards built for the NodeMCU project. These can be found on Amazon for around $9 or directly from Chinese suppliers for as cheap as $4 (e.g. gearbest.com or alibaba.com). As the NodeMCU boards do not have a built-in battery connection, connecting a battery is a little more complex (and beyond the scope of this Hackathon).

TSL2591 Breakout Board This board is specific to Adafruit. It includes some extras, like a voltage regulator. Adafruit and SparkFun also have a breakout board for the TSL25**6**1. This light sensor also uses the I2C bus, but it is not as sensitive as the TSL2591.

Lithium Ion Battery You only need this if you have the Adafruit Feather HUZZAH board and are going to run the system disconnected from the USB (most remote sensing scenarios). You can also use batteries with a higher mAh rating.

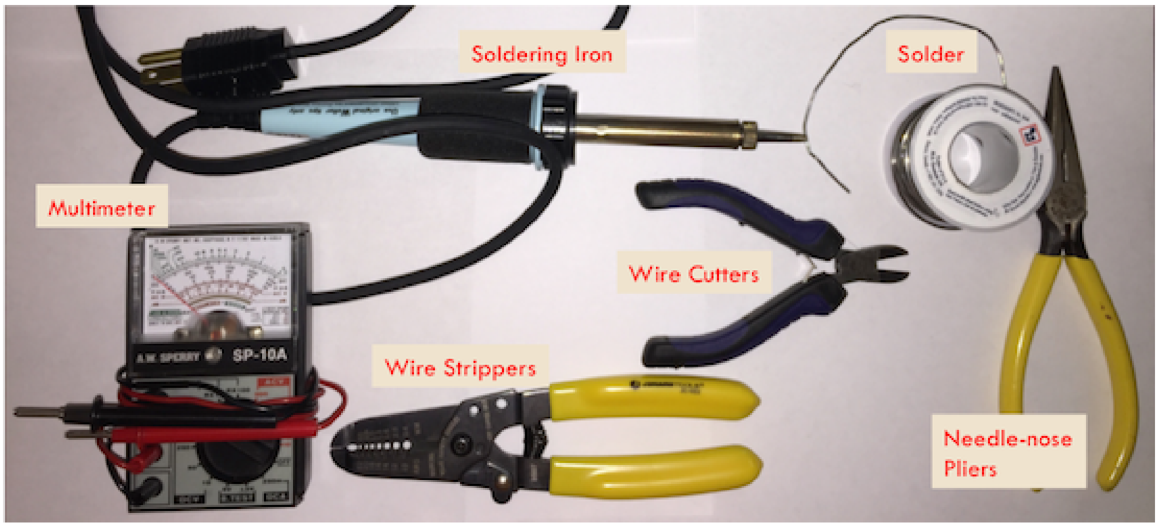
Half-size breadboard This will fit our two breakout boards. If you have more sensors you want to connect, you might consider a full-size breadboard. These are available from many suppliers. If you are buying for a class, you might find quantity discounts (e.g. we found 10 boards for $1.60 each).

Micro-USB to USB cable This connects the ESP8266 breakout board to the Laptop/PC. You can use just about any such cable (you probably have a few laying around your house), but it must have both data and power connections.

Breadboarding wires These are pre-cut male jumper wires that we will use to make the connections on the breadboard. You can also just get a spool of insulated solid wire from 22 to 26 gauge and use a wire stripper and cutters to get wires of the right size.

Recommended Tools

Here is a picture of the tools we recommend to build and test the system:



Details

Multimeter A multimeter (or multitester) is an instrument that measures voltage, current, and resistance. As you build your system, you can use the resistance measurements to check whether connections are made (should be either resistance of zero or infinity). You may also need/want to check voltages. The multimeter pictured is an old analog one. There are also new digital meters available for around $25.

Soldering iron and solder Adafruit provides the headers (a strip of plastic containing pins) separate from their breakout boards. We need the soldering iron and solder to connect the headers to the boards. This allows us to insert the breakout boards into the breadboard. If you are participating in a group hackathon, this step may have already been done for you.

Wire strippers and wire cutters If you are not using the pre-cut jumper wires, you will need to cut and strip your wires. You may want to have these tools handy anyway. Most wire strippers also can cut wires, but I find it helpful to have a separate tool.

Needle-nose pliers Although not strictly necessary, these can be helpful.

Now, we are ready to begin the *hardware assembly*.

CHAPTER 3

3. Hardware Assembly

To assemble our system, we will connect our light sensor to the ESP8266 board. This involves connecting ground and power, and then the two signals needed for the I2C bus. The exact details depend on which ESP8266 board you are using (different boards have different pinouts).

Cautions

First, a few important cautions to consider when working with electronic hardware.

Avoid Static Electricity

Static electricity (e.g. that shock you sometimes get when touching a doorknob) can damage electronic components. When working with the breakout boards, you should take precautions to avoid static electricity. Some simple precautions:

- If possible, avoid carpeted floors, rubber-soled shoes, and wool clothing.
- Most computer boards and chips come in anti-static packages. Keep them in the packaging until you need them.
- Touch some grounded metal (e.g. your computer's chassis, if it is metal) before touching any electronic components.
- Ideally, use a grounded anti-static wrist strap and mat when working with your components.

Plenty of resources about anti-static procedures are available online (e.g.[1],[2]).

Avoid Short-circuits

In digital electronics, electrical current flows from a positive voltage supply (usually indicated by a red terminal), through circuitry, and to the electrical ground (usually indicated by a black terminal). The positive power voltage is

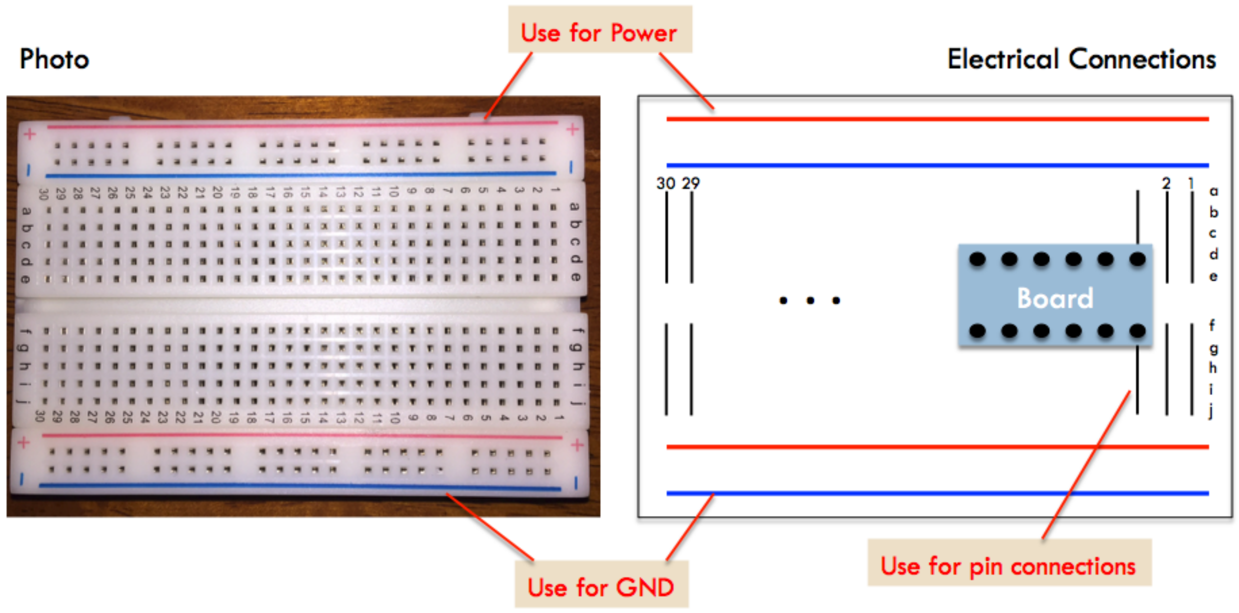
[1] http://www.howtogeek.com/169994/how-to-protect-your-pcs-hardware-from-static-electricity-when-working-on-it/
[2] http://www.wikihow.com/Avoid-(Static)-Electric-Shock

typically 5 volts or 3.3 volts. You should **never** allow the power lines to be connected to the ground line, without the proper electrical components in between. Ohm's law[3] states that current flowing is equal to the voltage divided by the electrical resistance. If this resistance is zero (e.g. a direct connection), then an infinite current can flow. This is known as a *short circuit*. Causing one will not only destroy your electrical components, but may also damage the USB port on your computer.

OK, with that we are ready to start hooking up the hardware.

Breadboards

For easy, solderless connections that can be changed, we will use a breadboard. The picture below shows an example breadboard and the electrical connections that exist between the holes:



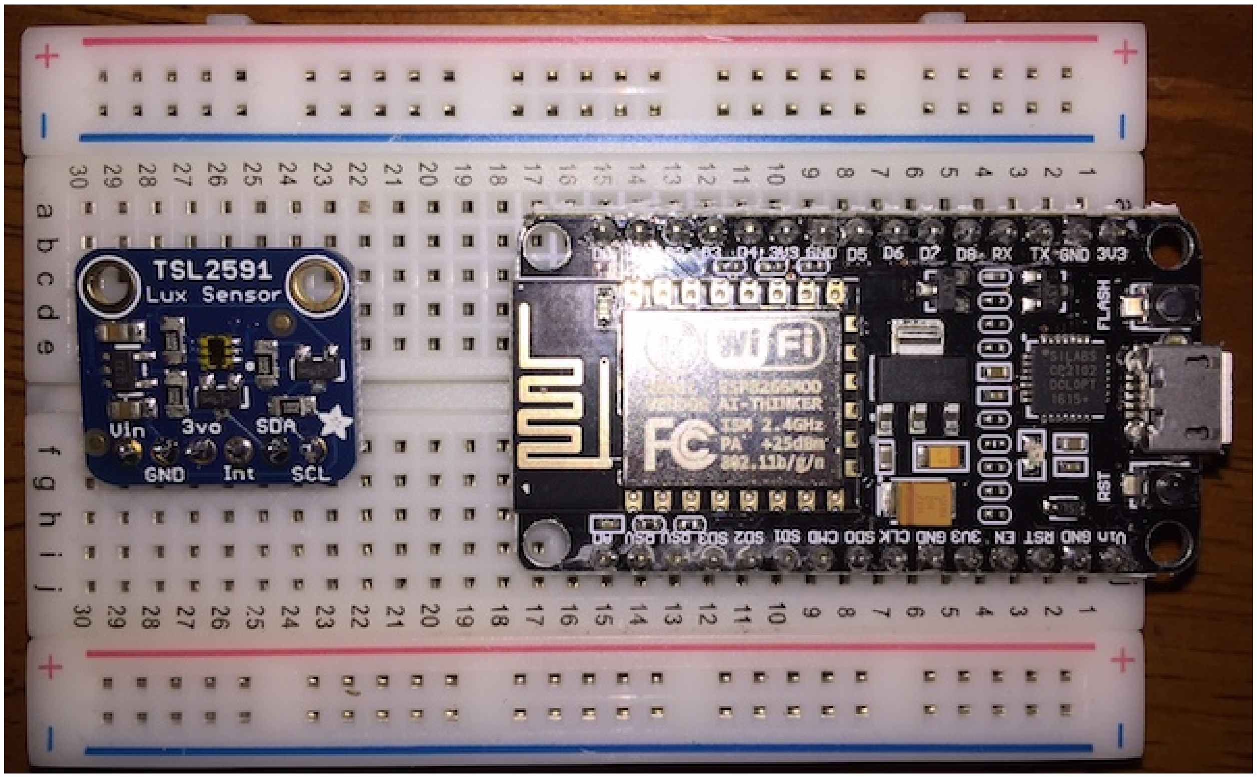
If you were to peel the backing off the breadboard, you would see metal connections matching the lines shown in the right side of the picture. These connect individual holes in a way that can be used to facilitate connections between pins on the boards by inserting jumper wires. There are four horizontal connections running the length of the breadboard that are intended for power connections. Between the top and bottom power connections, there are two rows vertical connections, with 30 columns (at least on our half-size board).

Seating and Soldering the Breakout Boards

The ESP8266 and TSL2591 Breakout boards are inserted with their two rows of pins parallel to the power connections and perpendicular to the smaller columns. This allows the holes directly above and below the pins to be used for connections.

If you hold the board with the blue Ground row along the bottom, I recommend seating the ESP8266 board on the right side. The miniUSB connector should be along the right edge of the board, for easy connectivity. The TSL2591 board is then seated to the left of the ESP8266 board. The picture below shows the two boards once they have been seated:

[3] https://en.wikipedia.org/wiki/Ohm's_law



Soldering

If one or both of your boards do not have their headers pre-soldered, I suggest seating the headers into the breadboard, placing the breakout board on top of the headers, and then soldering. Detailed instructions on soldering the headers may be found on the Adafruit website[4],[5].

Board connections

The Adafruit Feather HUZZAH and NodeMCU boards have different pinouts. Please follow the instructions below for your board. Note that you should do all this wiring with your board unpowered (no battery and no connection to your laptop via USB).

The wiring tables below make power connections through the horizontal power buses on the breadboards. This makes it easy to add more circuitry to the board later. If you do not plan to do so, you can wire the power pins of the ESP8266 and sensor boards directly to each other, saving a few connections. Of course, be sure to wire 3V to 3V and GND to GND.

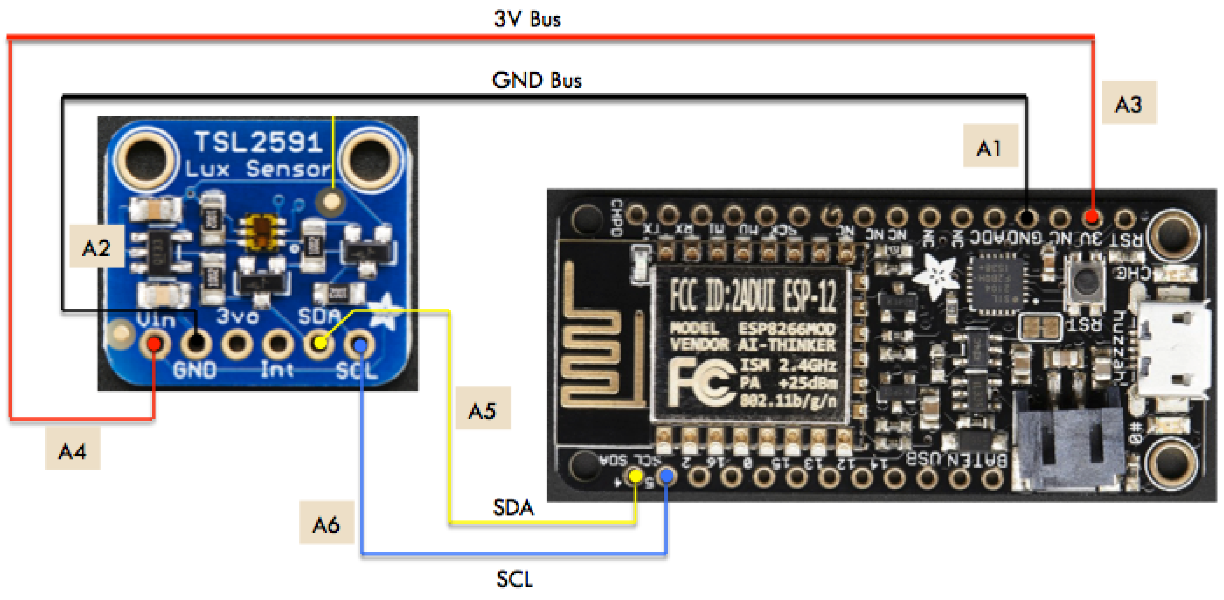
Adafruit Feather HUZZAH ESP8266

Here are the connections to make if you have the Adafruit ESP8266 board:

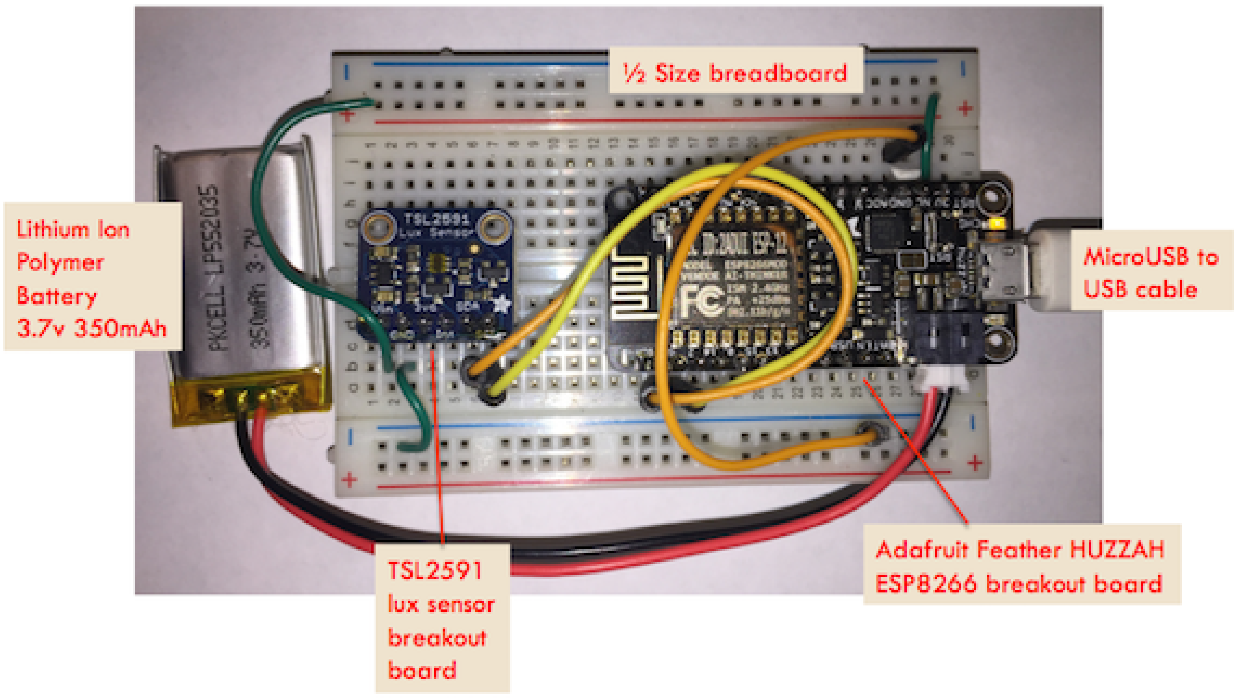
[4] https://learn.adafruit.com/adafruit-feather-huzzah-esp8266/assembly
[5] https://learn.adafruit.com/adafruit-tsl2591/wiring-and-test?view=all#assembly

Connection Number	Signal Name	Location of Source	Location of Destination
A1	GND	ESP8266 board pin #4 from right on top row	Bottom row of breadboard
A2	GND	Bottom row of breadboard	TSL2591 board pin #2 from left
A3	3V	ESP8266 board pin #2 from right on top row	Top row of breadboard
A4	Vin	Top row of breadboard	TSL2591 board pin #1 from left
A5	SDA (GPIO #4)	ESP8266 board pin #1 from left on bottom row	TSL2591 board pin #5 from left
A6	SCL (GPIO #5)	ESP8266 board pin #2 from left on bottom row	TSL2591 board pin #6 from left

Here is a logical wiring diagram:



And, finally, here is a photograph of the completed system using the Adafruit Feather Huzzah board:


½ Size breadboard
Lithium Ion Polymer Battery 3.7v 350mAh
MicroUSB to USB cable
TSL2591 lux sensor breakout board
Adafruit Feather HUZZAH ESP8266 breakout board

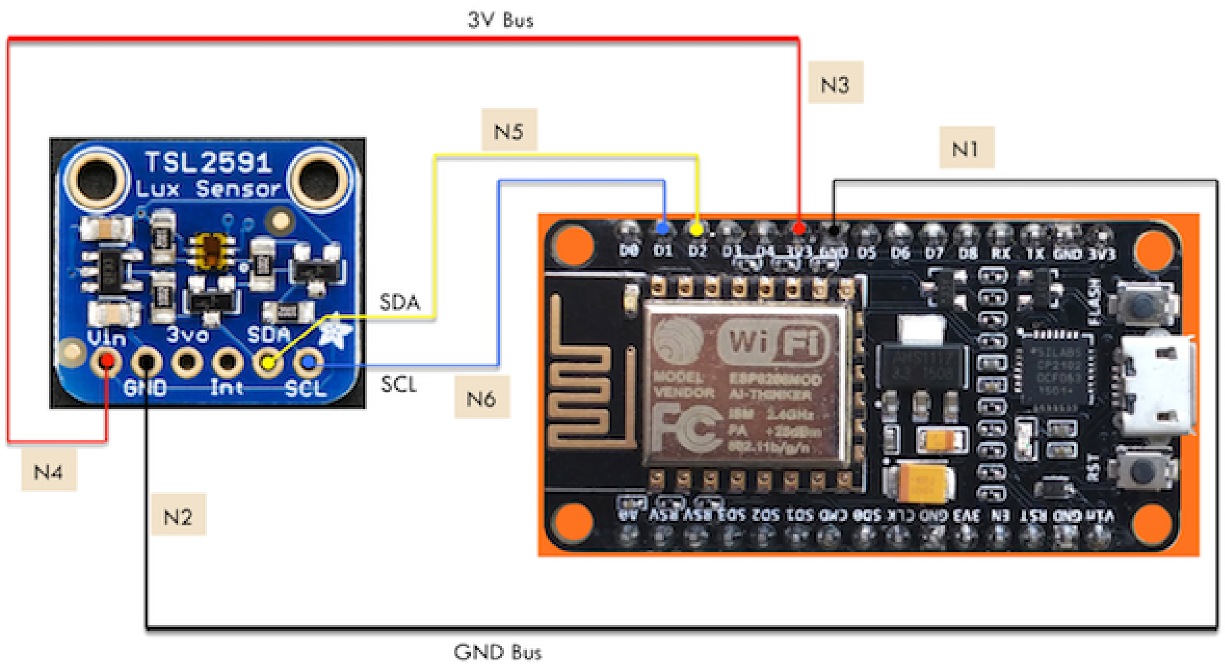
Now, please skip down below to the *Verifying Connections* section.

NodeMCU

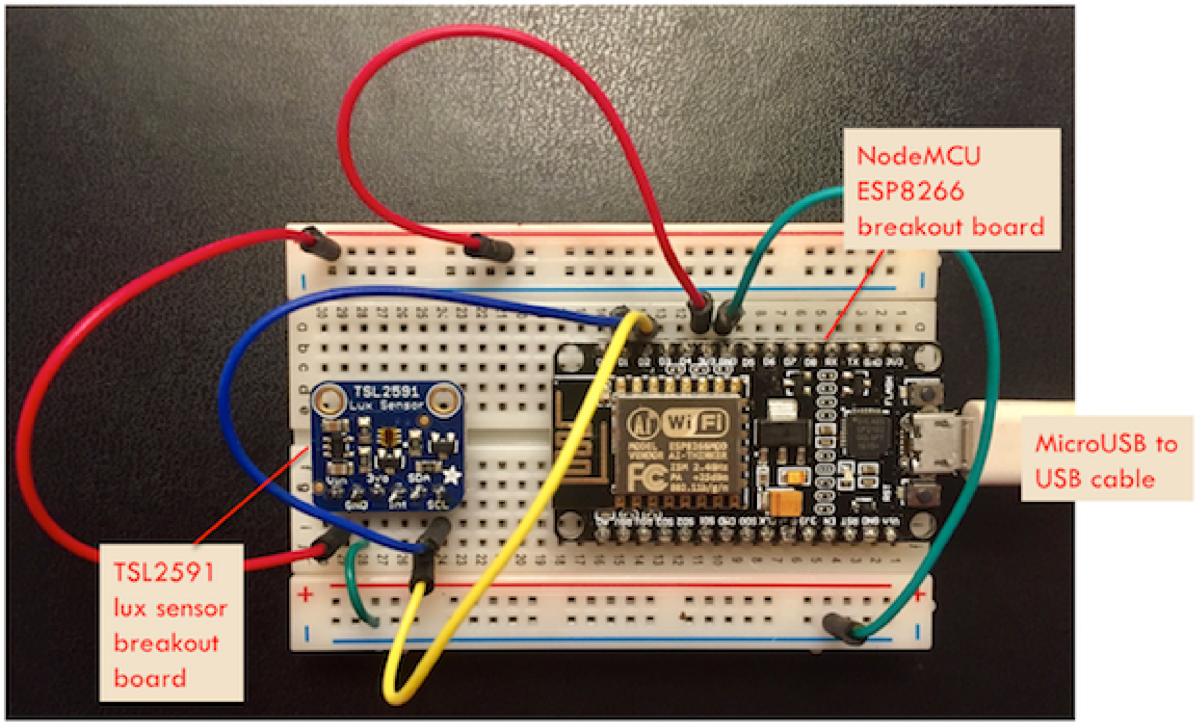
Here are the connections to make if you have the NodeMCU ESP8266 board:

Connection Number	Signal Name	Location of Source	Location of Destination
N1	GND	ESP8266 board pin #7 from left on top row	Bottom row of breadboard
N2	GND	Bottom row of breadboard	TSL2591 board pin #2 from left
N3	3V3	ESP8266 board pin #6 from left on top row	Top row of breadboard
N4	Vin	Top row of breadboard	TSL2591 board pin #1 from left
N5	SDA (D2)	ESP8266 board pin #3 from left on top row	TSL2591 board pin #5 from left
N6	SCL (D1)	ESP8266 board pin #2 from left on top row	TSL2591 board pin #6 from left

Here is a logical wiring diagram:



And, finally, here is a photograph of the completed system using the NodeMCU board:


NodeMCU ESP8266 breakout board
MicroUSB to USB cable
TSL2591 lux sensor breakout board

Verifying Connections

Now that the system has been wired up, we can verify the connections. You might start by visually inspecting the connections to make sure they match the wiring diagram. Pay special attention to the 3V and ground connections to avoid short circuits. If you are unsure, set your multitester to resistance mode and check the resistance between GND and 3V. It should be a high value.

Now, you can use the microUSB to USB cable and connect your system to your laptop. You should see an LED light up (at least briefly) on the board. You can test the power connection by setting your multitester to voltage mode, placing the black lead on a GND pin, and the red lead on a 3V pin. You should see a voltage around 3 volts.

Now, we are ready to install the *firmware and software*!

CHAPTER 4

4. Software and Firmware Install

We are now ready to start installing software on our board, over the USB connection. For this, you will need a Mac, Linux PC, or Windows PC. We have only tested the firmware procedure on Mac and Linux, so those systems are recommended. However, others have successfully installed MicroPython from Windows, so it should work.

In the instructions that follow, we will use the term “host computer” to mean your PC/Mac/Linux box and “ESP8266” to mean the ESP8266-based system you have assembled on the breadboard.

Software Prerequisites

Python

On your host computer, you will need an installation of Python 3, version 3.4 or later. As described below, you will be adding a few packages to your Python install. Rather than add these packages to your system-wide Python install, we recommend installing them into a *virtual environment*. See the Python documentation page for venv for details.

Erasing and flashing the firmware is done through a utility called `esptool.py`[1]. You can install it via `pip` with the command:

```
pip install --upgrade esptool
```

If `pip` on your system defaults to Python 2, then use `pip3` instead.

We will also use `mpfshell`, a shell-like utility for moving files between the ESP8266 and the host computer.[2] It is available on GitHub: https://github.com/wendlers/mpfshell. Install it with:

```
pip install git+https://github.com/wendlers/mpfshell
```

We will use the ThingFlow framework for writing some of the software running on the ESP8266. Please download or clone the ThingFlow repository on GitHub: https://github.com/mpi-sws-rse/thingflow-python. Note that we will

[1] https://pypi.python.org/pypi/esptool/1.3

[2] MicroPython has a web-based interface for transferring files over the WiFi network. However, we have seen some problems with using it and recommend using `mpfshell`

not be using the packaged version of ThingFlow available on pypi.python.org. That version is only for the standard CPython implementation. We will be using the code in the `micropython` subdirectory of the ThingFlow source tree.

MicroPython Firmware

You can download the latest version of the MicroPython firmware for ESP8266 from here: http://micropython.org/download#esp8266. You will want the latest stable firmware version[3]. Since the name of the firmware file will change over time, we will refer to it as `esp8266-YYYYMMDD-vX.Y.Z.bin` in any commands. Just substitute with name of your image file.

Mac-specific

If you are using a Mac, you will need to download a USB to UART bridge driver from Silicon Labs. This will allow your Mac to send serial communications for the Silicon Labs USB adapter chip on your ESP8266 board. The driver is available from their website: https://www.silabs.com/products/mcu/Pages/USBtoUARTBridgeVCPDrivers.aspx

For interacting with the Python REPL running on the ESP8266, you will use the `screen` program. It comes pre-installed on MacOS.

Linux-specific

You should not need any drivers for the serial connection. You will want to add your user to the `dialout` group so it can access the tty device for the serial connection. Replacing `MYUSER` with your user id, the command is:

```
sudo adduser MYUSER dialout
```

We will be using the GNU `screen` program to interact with the Python REPL running on the ESP8266. If it is not already installed, you will need to do so. If you are running a Debian-based distribution (Debian, Ubuntu, Raspbian), then you can install it via:

```
sudo apt-get install screen
```

Windows-specific

You will need the `PuTTY` program to interact with the Python REPL running on the ESP8266. It is free and available at http://www.putty.org.

As stated above, we have not attempted this process with a Windows host. We suggest that you follow the instructions here: http://www.instructables.com/id/The-Super-Easy-Micropython-ESP8266-Guide-No-Guessw/?ALLSTEPS.

The rest of the instructions below are for Mac and Linux. You should be able to follow along using the Instructables guide as a reference. You will use `PuTTY` instead of `screen`. Of course, the command options will be slightly different.

Testing Your Connection

With the prerequisite software installed, you can now connect your ESP8266 and verify that it can talk to your host system. Using the microUSB to USB cable, plug it into the USB port. Now, look for a new serial tty device in

[3] At the time this is being written, the latest stable version is 1.8.7, and the firmware image filename is `esp8266-20170108-v1.8.7.bin`.

your /dev directory. On Mac, it should be called /dev/tty.SLAB_USBtoUART. On Linux, it should be called /dev/ttyUSB0. The device will only be present when your ESP8266 board is plugged in.

Now, run screen as follows (substituting TTYDEVICE with ttyUSB0 or tty.SLAB_USBtoUART):

```
screen /dev/TTYDEVICE 115200
```

The 115200 is the baud rate. Now, press the reset key. You should see some characters, either a prompt or some garbage. If you get garbage, you might try a baud rate of 9600. Do not be worried at this point if you only see garbage, we are going to rewrite the firmware anyway. As long as you saw the tty device, you should be OK.

Now, exit screen (pressing Control-A k and then "y"), and run the reset command in your terminal session. You are ready to install the firmware.

Installing the Firmware and Testing the REPL

We will use esptool.py to erase the old firmware and install the latest version of MicroPython[4]. Plug your ESP8266 back into the USB port and run the following from the command line:

```
esptool.py --port /dev/TTYDEVICE erase_flash
```

Now, install the new firmware with the following (substituting in the device name and MicroPython image file name):

```
esptool.py --port /dev/TTDEVICE --baud 460800 write_flash \
--flash_size=detect --verify -fm dio 0 esp8266-YYYYMMDD-vX.Y.Z.bin
```

If this step fails, you might try lowering the baud rate (e.g. from 460800 to 115200).

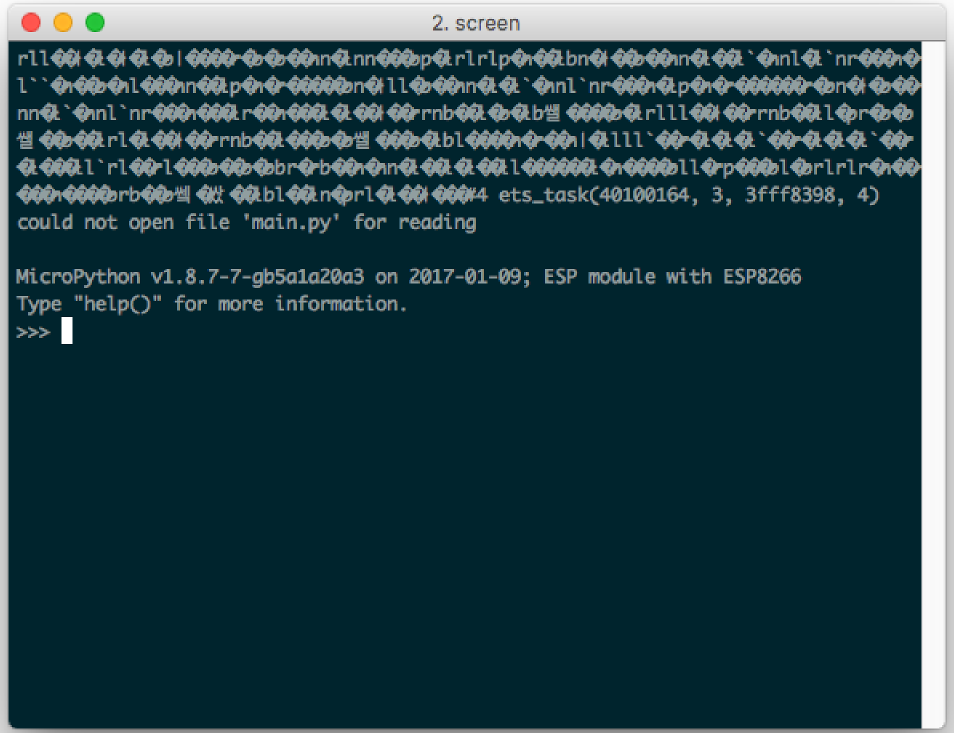
Next, we will use screen to connect to our board. From the command line:

```
screen /dev/TTYDEVICE 115200
```

Now, press the reset button on your ESP8266 board[5]. In your screen session, you should see some garbage characters followed by a version string and a prompt. For example:

[4] You can also find instructions for this in the MicroPython documentation. The relevant section is at https://docs.micropython.org/en/latest/esp8266/esp8266/tutorial/intro.html

[5] On both the NodeMCU and Adafruit boards, the reset button is labeled "RST". You can use this button to force a reboot of the system. This is useful to get it to a known state (recommended after plugging into the USB connection).



2. screen

```
rll��������|�����������n�nn����p�rlrlp����bn�������n����`�nl�`nr�����
l``������nl����n���p������������n��ll�����n���`�nl`nr�����p�����������n�����
nn��`�nl`nr����������r������������rnb������b썰������rlll����rnb���l�pr���
썰������rl�������rnb�����������썰������bl���������|�lll`�������`�������`��
������l`rl����l������������br��b������n��������l�������������ll�p����l��rlrlr�n���
������������rb����썰 ��� ���bl����n�prl���������4 ets_task(40100164, 3, 3fff8398, 4)
could not open file 'main.py' for reading

MicroPython v1.8.7-7-gb5a1a20a3 on 2017-01-09; ESP module with ESP8266
Type "help()" for more information.
>>>
```

You should now be able to type Python code in at the prompt:

```
>>> print("Hello, world")
Hello, world
>>>
```

Congratulations, you have MicroPython running on your ESP8266! If you had problems, you might try looking at the troubleshooting hints provided in the MicroPython documentation[6].

Next, let's write an *application* for our board.

[6] https://docs.micropython.org/en/latest/esp8266/esp8266/tutorial/intro.html#troubleshooting-installation-problems

CHAPTER 5

5. ThingFlow Application

Now that our board is up and running, we can use ThingFlow to build a simple application.

ThingFlow

ThingFlow is a Python 3 framework for building IoT event processing and filtering applications. It is centered around *event streams*, which are infinite sequences of timestamped sensor value samples. The key components in ThingFlow are *sensors*, which capture data from the outside world, *things*, which transform event streams, and the *scheduler* which manages the recurring sampling of sensors. ThingFlow allows you to write your IoT code as data flows ("from this to that") instead of as procedural code ("if this then that").

There are two implementations of ThingFlow in Python. The goal is to provide a common API from very small sensor nodes (like the ESP8266) to large servers (for event concentration and data science). We will be using the MicroPython version, which may be found in the `micropython` subdirectory of the thingflow-python repository.

Due to the memory limitations of the ESP8266, the MicroPython version is very stripped down, and only supports a subset of the full ThingFlow API. That will be fine for our application, as the focus of our code will be to sample the light sensor and send the values over the network to a larger system for processing and / or storage.

Reading the Light Sensor

A *sensor* in ThingFlow is a Python object which satisfies two criteria:

1. It provides a `sensor_id` property which can be used in event messages to uniquely identify the sensor.
2. It provides a `sample()` method which returns the current value associated with the sensor.

The ThingFlow MicroPython implementation includes a sensor implementation for the TSL2591 lux sensor. We will first copy this over to the ESP8266 and verify that we can read the sensor.

Copying Files

Close any previous `screen` sessions with your ESP8266 system, and restart it.

In a terminal session, go to the `micropython` subdirectory of the ThingFlow repository. From this directory, run the `mpfshell` utility. In this session type the following (substituting `tty.SLAB_USBtoUART` or `ttyUSB0` for TTYDEVICE):

```
open TTYDEVICE
ls
```

The `open` command establishes a connection to your ESP8266. The `ls` will list the files on that system (MicroPython includes a very simple filesystem implementation). Initially, you should only see the file `boot.py`, which is installed as a part of the firmware flash.

Now, run the command `lls`. This lists the files on your host system, in the directory where you ran `mpfshell`. Now, run the following:

```
lcd sensors
lls
```

This changes the host directory to the `sensors` subdirectory and lists the files. One of the files you should see is `tsl2591.py`. This is the code which implements the interface to the light sensor.

Now, run:

```
put tsl2591.py
ls
```

This copies the file `tsl2591.py` to the ESP8266 and lists the files on it. We should now see our sensor code file, in addition to `boot.py`.

Calling the Light Sensor

Next, we want to import and call the sensor code from the MicroPython REPL. There are two ways to do this:

1. You can access the REPL directly from `mpfshell`. Just enter the command `repl`, and you will be in the REPL. You can later exit the REPL and get back to the main shell via the key combination CONTROL and "]".
2. Alternatively, you can exit `mpfshell` and then run `screen` again to get a REPL directly.

Either way, once we are in the REPL, we want to import the Tsl2591 class from `tsl2591.py`, instantiate an instance, and call its `sample()` method. Here is an example session:

```
>> from tsl2591 import Tsl2591
>>> tsl = Tsl2591('lux-1')
>>> tsl.sample()
296.3712
```

The `lux-1` passed to the the `Tsl2591` constructor is the sensor id. In this case, it can be an arbitrary string. The call to `sample()` may take almost a second to complete (it takes some time to properly sample the value from the lux sensor). The value returned in the light reading in units of *lux*. Try running the sample call again with your hand covering the light sensor – it should return a lower value. Now, we have verified that the light sensor is working for us!

Debugging problems

If you get an error when instantiating or reading from the light sensor, you might have a bad connection or a problem with your sensor. Here's what an error might look like in your REPL session:

```
>>> from tsl2591 import Tsl2591
>>> tsl = Tsl2591('lux-1')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "tsl2591.py", line 83, in __init__
  File "tsl2591.py", line 91, in set_timing
  File "tsl2591.py", line 150, in enable
  File "tsl2591.py", line 61, in write_byte_data
OSError: [Errno 19] ENODEV
```

If you get an error like this, double-check your wiring. If you believe all your connections are correct, you might check whether one of the physical connections is bad. Set your multitester to resistance mode. For each connection, place one lead on the pin of the ESP2866 board and the other on the associated pin of the TSL2591 board. The resistance should measure zero if there is indeed a connection.

A Light Sampling Application

Now, we will copy over the main module of ThingFlow and use the scheduler to periodically call our sample method and print the result. First, start `mpfshell` in the `micropython` directory. Copy `thingflow.py` over to the ESP8266 as follows:

```
open TTYDEVICE
put thingflow.py
ls
```

You should see that `thingflow.py` is now on the ESP8266.

Next, go back to the MicroPython REPL. We will import the ThingFlow core and our sensor. Then we will instantiate the sensor object and a scheduler. Finally, we will call the scheduler with the sensor, asking it to sample the sensor once every two seconds and print the resulting event. Here is the REPL session:

```
>>> from thingflow import *
>>> from tsl2591 import Tsl2591
>>> tsl = Tsl2591('lux-1')
>>> sched = Scheduler()
>>> sched.schedule_sensor(tsl, 2.0, Output())
<closure>
>>> sched.run_forever()
('lux-1', 344, 294.9023)
('lux-1', 345, 294.9023)
('lux-1', 347, 294.9023)
('lux-1', 349, 288.2113)
('lux-1', 351, 245.6161)
('lux-1', 352, 214.1184)
('lux-1', 354, 48.14401)
('lux-1', 356, 50.75521)
('lux-1', 358, 294.9023)
```

The `schedule_sensor()` call takes three parameters: the sensor object to be schedule, the sample interval in seconds, and the downstream data flow. In this case we are just calling the `Output` "thing" to print the messages.

The tuples being printed have three elements: the sensor id, a timestamp, and the sensor reading.

In the *next section*, we'll see how we can get these samples off the ESP2866 using its WiFi radio and the MQTT protocol.

CHAPTER 6

6. Messaging with MQTT

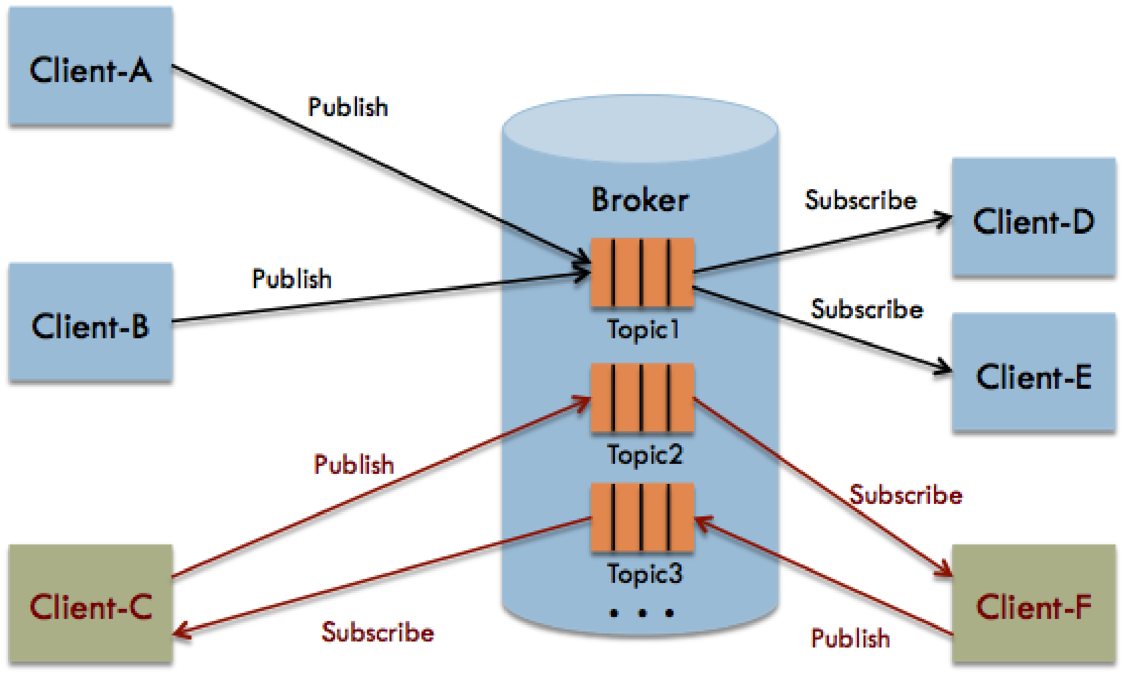
MQTT (MQ Telemetry Transport) is a lightweight publish/subscribe messaging protocol frequently used in IoT applications. It is a very thin layer over TCP/IP, and has many implementations. MQTT is even an OASIS standard[1]. The Micropython software for ESP8266 includes a client implementation in the `umqtt` module[2].

MQTT Basics

An MQTT-based application will include two or more *clients*, which are applications exchanging messages, and a *broker*, which is a server that accepts incoming messages and routes them to the appropriate destination client. As with most *publish-subscribe* systems, message sends involve *publishing* on a specified *topic*. The broker then forwards the message to all *subscribers* of that topic. These primitives can be used to build different interaction patterns. The picture below shows an example:

[1] http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

[2] The `umqtt` module is not in the official Micropython documentation, but module is definitely present in the firmware image. The API is simple enough that you can understand it by a quick read of the source code: https://github.com/micropython/micropython-lib/tree/master/umqtt.simple and https://github.com/micropython/micropython-lib/tree/master/umqtt.robust.



Here, we see a broker with three topics: *topic1*, *topic2*, and *topic3*, which interact with clients using two different interaction patterns. *Client-A* and *Client-B* are publishing their messages to *topic1*. When a message is received at the broker, it is passed on to any current subscribers to the topic. In this case, both *Client-D* and *Client-E* will receive each message. We will call this a *push* pattern, as it is one directional, and the interaction is initiated by the output_thing. In an IoT context, the output_thing is usually a sensor node and the subscriber(s) are servers that process and store the sensor data.

Client-C and *Client-F* are interacting in a request-response or *pull* pattern. *Client-C* sends request messages to *topic2*. These are received by *Client-F*, which sends responses on *topic3*. In an IoT context, this pattern may be useful when the server wishes to poll the sensor nodes for data or for situations where the sensor nodes need to query a server for configuration data.

In our application, we will be using the simpler push pattern.

Quality of Service

Networked systems are never 100% reliable – systems may crash, connections can be lost, or parts of an application taken down for maintenance. This is even more true for IoT systems, where sensors may be in harsh physical environments. Middleware like MQTT can improve reliability by storing messages and implementing handshake protocols. However, this has a cost in terms of resources and system complexity. MQTT gives you flexibility by specifying a *Quality of Service* (QoS) with each message.

`qos` is a parameter available on each publish call. It is one of three levels:

- `0` – at most once. This means that the system will make a best effort to deliver the message, but will not store it and will drop the message in the event of an error. This is typically the behavior you would use in a sensor application: if a message is lost, the next sensor sample will be coming soon, anyway.
- `1` – at least once. This means that the system will use storage and handshaking to ensure that the message is delivered. However, in doing so it may send the same message multiple times, resulting in duplicates.

- 2 – exactly one. This means that each message will be delivered, and the handshaking protocol will ensure that duplicates are not passed to clients. This is the behavior you want if you are implementing a banking system (but not so much in the IoT world).

For our application, we will use QoS level 0.

Security

MQTT provides security, but it is not enabled by default. For our experiments, we will rely on the encrypted WiFi connection to provide a basic level of security. MQTT has the option for Transport Layer Security (TLS) encryption, just as used with HTTPS. We recommend using that for any system you put into production.

MQTT also provides username/password authentication. Note that the password is transmitted in clear text. Thus, be sure to use TLS encryption if you are using authentication.

Host-side Setup

Now, let us install an MQTT broker on our host system. We will use Mosquitto, an open source broker from the Eclipse Foundation. It is implemented in C, and can run well on fairly constrained systems (e.g. a Raspberry Pi). The documentation and other other resources may be found at http://mosquitto.org. The downloads are hosted at the main Eclipse Foundation Website, and may be found at https://www.eclipse.org/mosquitto/download/. That page has instructions for installing on most platforms.

You will want to install both the broker and the client utilities (in particular, mosquitto_sub). On Debian-based Linux distributions (Debian, Ubuntu, Raspbian, etc.) these are in two packages, so you will install them as follows:

```
sudo apt-get install mosquitto mosquitto-clients
```

Once installed, make sure that your Mosquitto broker is running and listening to port 1883 (the default). A simple test can be done with the mosquitto_pub and mosquitto_sub clients. Open two terminal windows. In one, type:

```
mosquitto_sub -t sensor-data
```

It should hang and not print anything immediately. The process is subscribing to the topic sensor-data and will print to standard output any messages that it receives. In the second window, run the following:

```
mosquitto_pub -t sensor-data -m "hi, there"
```

You should now see the message printed in the first (subscriber) window. This means that the broker is working. You can now kill the subscriber with a Control-C.

Finally, if your system has a firewall, make sure that port 1883 is open. Otherwise, connections from the ESP8266 system will be blocked.

ESP8266 Setup

MicroPython already has an MQTT client in its standard library, so we do not need to do much on the ESP8266-side. We will just copy over some convenience modules provided by ThingFlow.

On your host machine, go to the micropython subdirectory of your ThingFlow repository. Run mpfshell and copy the scripts for WiFi configuration and MQTT as follows (substituting your tty device name for TTYDEVICE):

```
open TTYDEVICE
put wifi.py
put mqtt_writer.py
```

Interactive Testing

Now, go to the MicroPython REPL (via either the `repl` command of `mpfshell` or through `screen`). We will first run our import statements:

```
>>> from thingflow import *
>>> from tsl2591 import Tsl2591
>>> from wifi import wifi_connect
>>> from mqtt_writer import MQTTWriter
```

Next, we configure the WiFi connection and then connect to the MQTT broker. Here is the code in the REPL (replace `my_wifi_sid`, `my_wifi_password`, and `mqtt_broker_ip` with values for your environment):

```
>>> SID='my_wifi_sid'
>>> PASSWORD='my_wifi_password'
>>> MQTT_HOST='mqtt_broker_ip'
>>> wifi_connect(SID, PASSWORD)
network config: ( ... )
>>> m = MQTTWriter('esp8266', MQTT_HOST, 1883, 'sensor-data')
Connecting to xxx.xxx.xxx.xxx:1883
Connection successful
```

We can now create a sensor and connect two downstream components: `Output`, which prints events to the standard output, and `m`, our MQTTWriter instance. Here is the REPL session:

```
>>> sensor = SensorAsOutputThing(Tsl2591('lux-1'))
>>> sensor.connect(Output())
<closure>
>>> sensor.connect(m)
<closure>
```

Finally, we instantiate an ThingFlow scheduler and schedule our sensor to be sampled once every two seconds:

```
>>> sched = Scheduler()
>>> sched.schedule_periodic(sensor, 2.0)
<closure>
>>> sched.run_forever()
('lux-1', 611, 284.1312)
('lux-1', 613, 284.1312)
('lux-1', 615, 284.1312)
...
```

To verify that these messages are being sent to our broker, we can use the utility `mosquito_sub` on the host machine. It takes one command line argument, the topic name (in our case `sensor-data`). We should see something like the following when we run it:

```
$ mosquitto_sub -t sensor-data
["lux-1", 624, 284.1312]
["lux-1", 626, 288.2113]
["lux-1", 627, 77.0304]
```

```
["lux-1", 629, 35.90401]
...
```

Great, now you have gotten live sensor data off your ESP8266 board!

Putting it all Together

Now, we will set up the ESP8266 to run our sample/send loop upon startup. We will also run a script on the host to subscribe to our topic and write the events to a CSV (spreadsheet) file. The source code for this section may be found on GitHub in the repository for this tutorial. Specifically, look in the `example_code` folder (https://github.com/jfischer/micropython-iot-hackathon/tree/master/example_code). The program `client.py` will run on the ESP8266 and the program `server.py` will run on our host.

client.py

First, open an editor and create a file `config.py` that contains configuration variables needed for your network and system. It should look something like this:

```
SENSOR_ID='lux-1'
WIFI_ESSID='my_wifi_sid'
WIFI_PASSWORD='my_wifi_password'
MQTT_HOST='mqtt_broker_ip'
MQTT_TOPIC='sensor-data'
SLEEP_TIME=5.0
```

You will definitely need to change the values for `WIFI_ESSID`, `WIFI_PASSWORD`, and `MQTT_HOST`. The others can be left as-is.

Now, use `mpfshell` to copy `config.py` and `client.py` to your ESP8266 (substituting for TTYDEVICE or use Control-] to exit the repl if `mpfshell` is already running):

```
open TTYDEVICE
put config.py
put client.py
```

Next, open a MicroPython REPL session. To start our main loop, we just need to import the `client` module. Here is what the REPL session looks like:

```
>>> import client
Disabled access point, network status is -1
network config: (...)
Connecting to xxx.xxx.xxx.xxx:1883
Connection successful
Running main loop with sample every 5.0 seconds...
```

The REPL should hang at this point because the ESP8266 is in its main loop. Messages should be sent to the MQTT broker once every 5 seconds.

Now that we have verified the `client.py` script, we will configure it to start upon boot. While still in your REPL session, enter Control-C to break out of the loop. You should see a `KeyboardInterrupt` exception. We will now rename `client.py` to `main.py` using `os.rename()`. Upon completion of its boot procedure, MicroPython will always run the script `main.py` if it is present. Here is the REPL:

```
>>> import os
>>> os.rename('client.py', 'main.py')
>>> os.listdir()
['boot.py', 'tsl2591.py', 'thingflow.py', 'wifi.py', 'mqtt_writer2.py', 'mqtt_writer.
↪py', 'config.py', 'main.py']
>>>
```

Finally, press the reset button of your ESP8266 board. It will reboot. You should see some garbage data followed by the same sequence of messages that you saw when you imported `client` from the REPL.

Now, let us turn our attention to the host side of things.

Verifying messages at the server

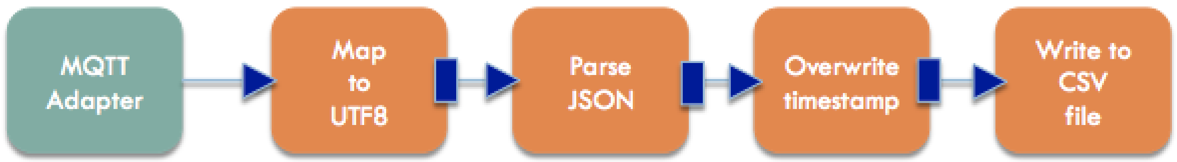
First, we will verify that we are getting the messages on the host. From your command line run:

```
mosquitto_sub -t sensor-data
```

You should see the sensor events printed once every five seconds.

server.py

We will next use the `server.py` script to read these events and write to a CSV file. It is an ThingFlow script that subscribes to messages on a specified topic, parses the messages, overwrites the timestamps with the server timestamp[3], and writes the events to a CSV file. Here is a graphical view of the dataflow:



Here is what the core part of the script looks like:

```
mqtt.select(lambda m:(m.payload).decode('utf-8'))\
    .from_json(constructor=SensorEvent)\
    .select(lambda evt: SensorEvent(sensor_id=evt.sensor_id,
                                    ts=time.time(),
                                    val=evt.val))\
    .csv_writer(filename)
```

Since it is running on a PC or server, this script uses the full CPython version of ThingFlow. You will need to have an installation of Python 3. You will also need the `paho-mqtt` package (installable via pip) and the `thingflow` package in your Python environment. Installing ThingFlow can be done in one of three ways:

1. Install ThingFlow via pip: `pip install thingflow`

[3] The ESP8266 does not have a realtime clock and the timestamps we get from it are only seconds since startup time. To work around this, we overwrite the timestamps on the server. This introduces some inaccuracy, but it should not be significant at our sample rates.

2. Install from your local repository by going to the `thingflow-python` directory and running `python setup.py install`.
3. Just set your PYTHONPATH environment variable to the full absolute path of the repository directory `thingflow-python`.

Once this is done, you should be able to run the following:

```
$ python3
>> import thingflow.base
```

If this succeeds, you have ThingFlow properly set up. We are now ready to run the `server.py` script. It takes two command line arguments: the topic to which it will subscribe and the name of the out CSV file. We'll run it as follows:

```
python3 server.py sensor-data test.csv
```

It should print a message about connecting successfully and then, once every five seconds, print the latest sensor event like this:

```
SensorEvent(sensor_id='lux-1', ts=1484535480.613611, val=371.6063)
SensorEvent(sensor_id='lux-1', ts=1484535485.6078472, val=371.6063)
SensorEvent(sensor_id='lux-1', ts=1484535490.4335377, val=371.6063)
SensorEvent(sensor_id='lux-1', ts=1484535495.4575906, val=371.6063)
...
```

If you look at the file test.csv, you should see four data values for each row:

1. The timestamp in Unix format (seconds since 1970)
2. The timestamp in human readable format
3. The sensor id.
4. The sensor value.

Congratulations! You have gotten the entire system working!

If you are interested, you can look at some more *projects* to do with your board.

CHAPTER 7

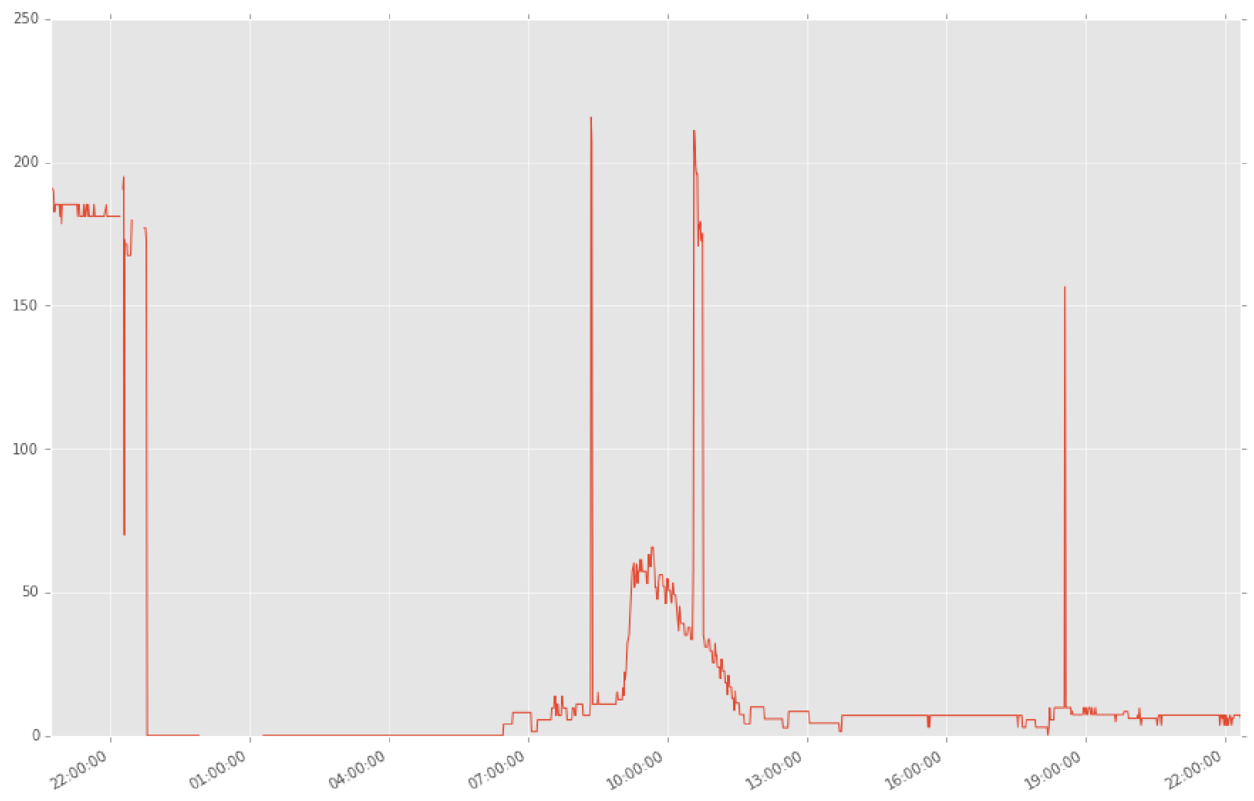
7. Projects

We now sketch out several projects you can build starting with your system. These include interfacing additional sensors, software projects, as well as incorporating the system in a larger application. Currently, this section is just a sketch. Over time, we will fill out more details.

Sensor Data Visualization

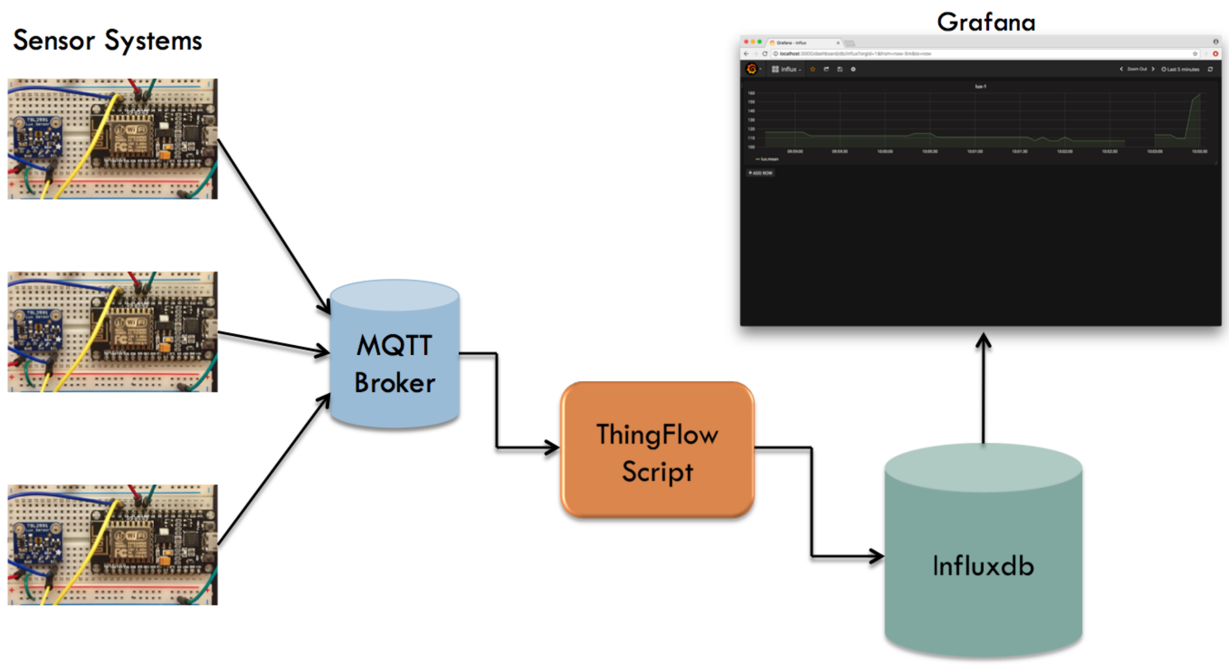
Graphing Light Sensor Data in Jupyter

Given the CSV files we have generated, it is quite easy to use Jupyter and matplotlib to graph the light sensor data over time. Here is an example plot of light sensor data gathered using the system we have described:



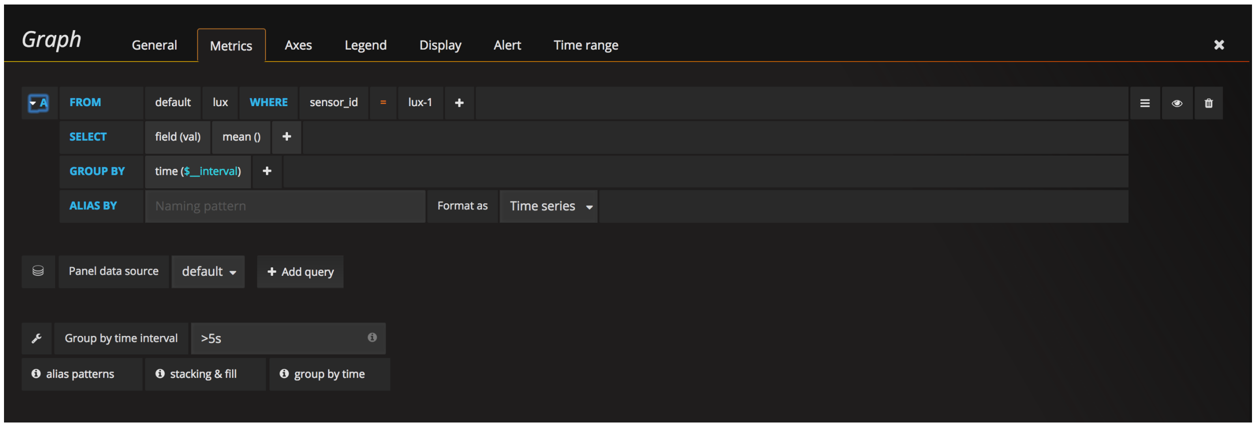
Realtime Dashboard of Light Sensor Data

Grafana is an open source web application for displaying time series data. You can use it with Influxdb (a time-series database) to store and visualize the light sensor data. Here is a picture of the workflow:



The setup for Influxdb and Grafana is pretty straightforward. You can find a ThingFlow script to read sensor events from a MQTT broker and write them to Influxdb in the `example_code` directory of this documentation's Git repository. The script is called `server_mqtt_to_influx.py`. Run the script with `--help` to see the command line options for configuring the connections to the broker and Influxdb.

The only non-intuitive part of the setup was the Grafana chart configuration. Be sure to set the "Group by time interval" parameter. For a five second sensor sample interval, use ">5s". Here is a screen shot of the full configuration I used:

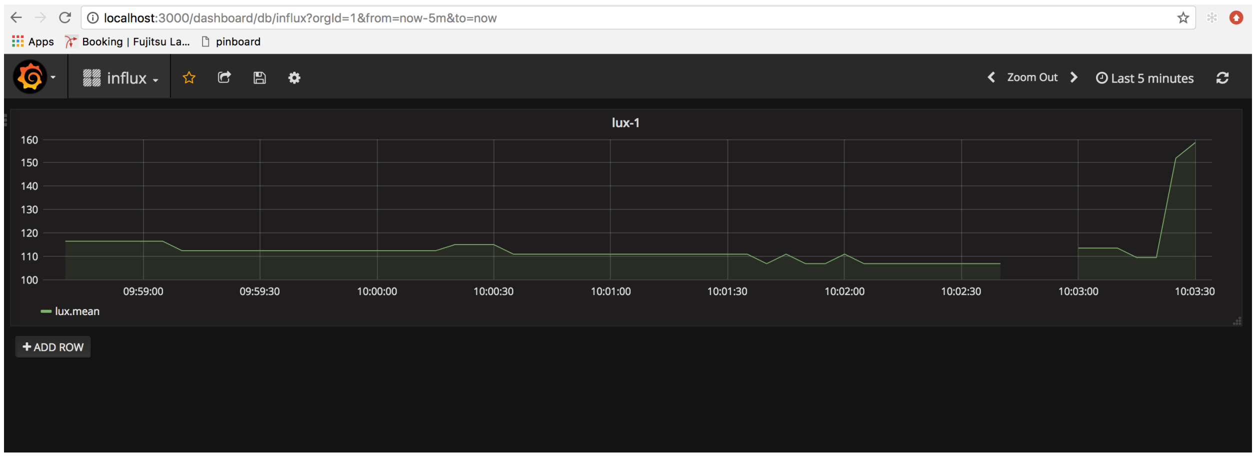


FROM default lux WHERE sensor_id = lux-1
SELECT field (val) mean ()
GROUP BY time ($__interval)
ALIAS BY Naming pattern Format as Time series

Panel data source default
Add query

Group by time interval >5s

A Grafana screenshot showing a chart of the light sensor data:



Lighting Replay Application

See https://github.com/mpi-sws-rse/thingflow-examples/tree/master/lighting_replay_app. This is an application which uses captured light sensor data to train a Hidden Markov Model. This model then is replayed when you are not home, to turn lights on and off in a realistic manner.

More Sensors

Temperature Sensor

There are many temperature sensors on the market. One well-documented sensor is the MCP9808 breakout board from Adafruit. Like our light sensor, it uses the IC2 bus for communication. To use the sensor, you need four connections: ground, power, SDA, and SCL. Power and ground can be connected to the power and ground lines of your breadboard or directly to the associated pins on the ESP8266. For the SDA and SCL pins, you can either connect them directly to the ESP8266 or daisy-chain them (connect in series) through the Lux sensor. Assuming you already have the TSL2591 sensor wired to your ESP8266, here are the connections you will need:

Signal Name	Location of Source	Location of Destination
GND	Bottom row of breadboard	MCP9808 board pin #2 from left
3V	Top row of breadboard	MCP9808 board ping #1 from left
SDA	TSL2591 board pin #5 from left	MCP9808 board pin #4 from left
SCL	TSL2591 board pin #6 from left	MCP9808 board pin #3 from left

ThingFlow for MicroPython has a sensor class for the MCP9808 located in `micropython/sensors/mcp9808.py`. Here is an example MicroPython REPL session where we sample both the lux and temperature sensors using ThingFlow:

```
>>> from thingflow import *
>>> from tsl2591 import Tsl2591
>>> from mcp9808 import Mcp9808
>>> lux = Tsl2591()
>>> temperature = Mcp9808()
Read manufacturer ID: 0054
Read device ID: 0400
>>> sched = Scheduler()
>>> sched.schedule_sensor(lux, 10, Output())
<closure>
>>> sched.schedule_sensor(temperature, 10, Output())
<closure>
>>> sched.run_forever()
('tsl2591', 108, 162.221)
('mcp9808', 108, 26.1875)
('tsl2591', 117, 162.221)
('mcp9808', 117, 26.1875)
('tsl2591', 126, 1.5096)
('mcp9808', 126, 26.1875)
```

Finally, Adafruit has a tutorial about using this sensor with MicroPython here.

Door Open/Close Detector

Adafruit describes a project to interface the ESP8266 to a door open/closed switch here: https://learn.adafruit.com/using-ifttt-with-adafruit-io/overview. It is Arduino based, but should be easily adaptable to MicroPython.

Accelerometer

An accelerometer opens up a number of interesting project ideas. For example, a student at the Max Planck Institute for Software Systems built a machine learning-based gait analysis application. She used an ESP8266, the ADXL345 breakout board from Adafruit, MicroPython, and ThingFlow. A ThingFlow sensor for the ADXL345 is located in the ThingFlow repository at `thingflow-python/micropython/sensors/adxl345_upy.py`. The machine learning was done in Python with scikit-learn.

Actuators

Turning on an external LED

Perhaps the simplest hardware interfacing is to an LED. There are many tutorials online about how to do this. You will need three connections: from a GPIO pin to the LED, from the LED to a resistor, and from the resistor to GND.

NeoPixel Strips

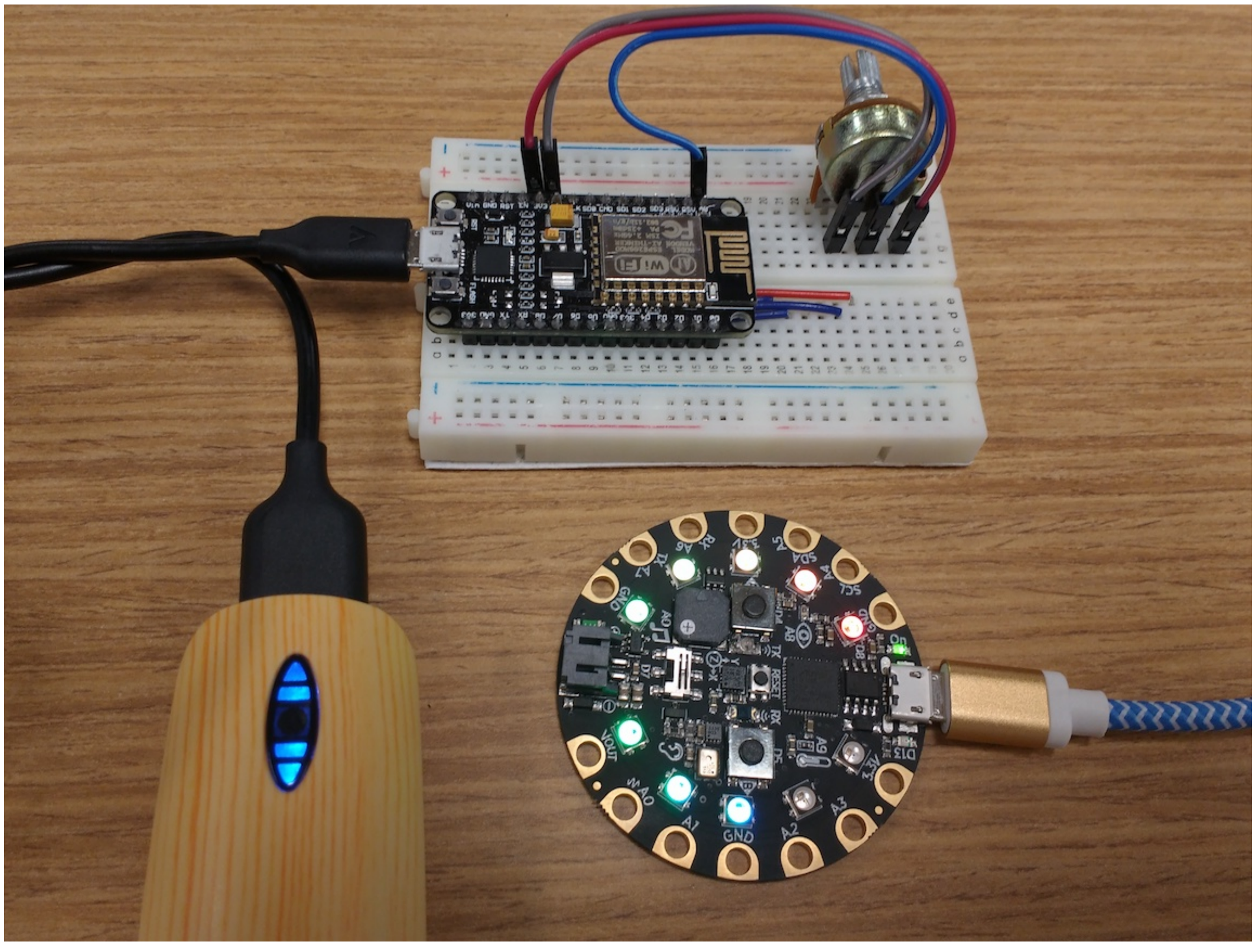
NeoPixels are addressable LED light sets available from Adafruit and others. MicroPython provides a NeoPixel "driver" library. ThingFlow for MicroPython provides a convenient interface on top of this – see `thingflow-python/micropython/neopixel_writer.py`.

Open-ended Projects

Adafruit Circuit Playground Express

The Circuit Playground Express (CPX) is a board from Adafruit that includes a Cortex M0 processor, temperature, motion, sound, and light sensors, NeoPixel lights, a speaker, and more. It runs CircuitPython, which is Adafruit's port of MicroPython to their system.

There are a lot of applications possible with this kind of board. In the directory `example_code/cpx`, Daniel Mizyrycki has graciously provided code from a cool demo. He uses a potentiometer connected to an ESP8266 to wirelessly control the NeoPixel lights on the Circuit Playground Express. Here is a picture of the demo in action:



Power Management

The ESP8266 has various power saving modes. If you have a simple sensor application that samples once a minute or less, you should be able to turn off the WiFi radio and put the board into a deep sleep mode. Some people claim that you can Run your ESP8266 for Years on a Battery, but there probably are complications in the real world. For example: How much power does your sensor consume? Can you turn it off programmatically? Can you avoid DHCP initialization when the radio is turned back on?

You can use a voltage divider to reduce the battery voltage below the 1.0V limit of the ESP8266 Analog to Digital converter. That would enable you to measure the voltage over time. Here are some instructions at Adafruit: https://learn.adafruit.com/using-ifttt-with-adafruit-io/wiring#battery-tracking.

On the software side, the ThingFlow scheduler optimizes samples to maximize the sleep time. The sleep can be easily changed to one of the deep sleep modes.

Using the ESP8266 as a WiFi Modem

The ESP8266 was originally developed to used in conjunction with larger microcontrollers. The intent was for the ESP8266 to provide WiFi capabilities, while the primary microcontroller provided the main processing. In fact, the original ESP8266 firmware included a Hayes-style ("ATDT") modem command set over the serial port.

There are still hobbyist use cases for the ESP8266 as a WiFi modem. For example, you might have a larger Arduino system that does not already support wireless operation. Can you connect the ESP8266 to these kind of systems via its second serial port? Can you emulate some modem functionality in MicroPython?

Home Assistant Integration

Home Assistant is a Python3 open-source application framework for automating your home. Since Home Assistant already supports MQTT, it should be easy to integrate remote ESP8266 nodes into the application. Can you do it? What cool applications can you build?

Another Micropython/ESP8266 Tutorial

We recently found another tutorial about Micropython and the ESP8266. It is more focused on lower level sensors and has some interesting hardware projects. It is called “MicroPython on ESP8266 Workshop” and is available here: http://micropython-on-esp8266-workshop.readthedocs.io/en/latest/index.html.

Finally, we conclude with some *advice* for hosting a hackathon.

CHAPTER 8

8. Teachers' Notes

In this section, we provide some advice to teachers of the class.

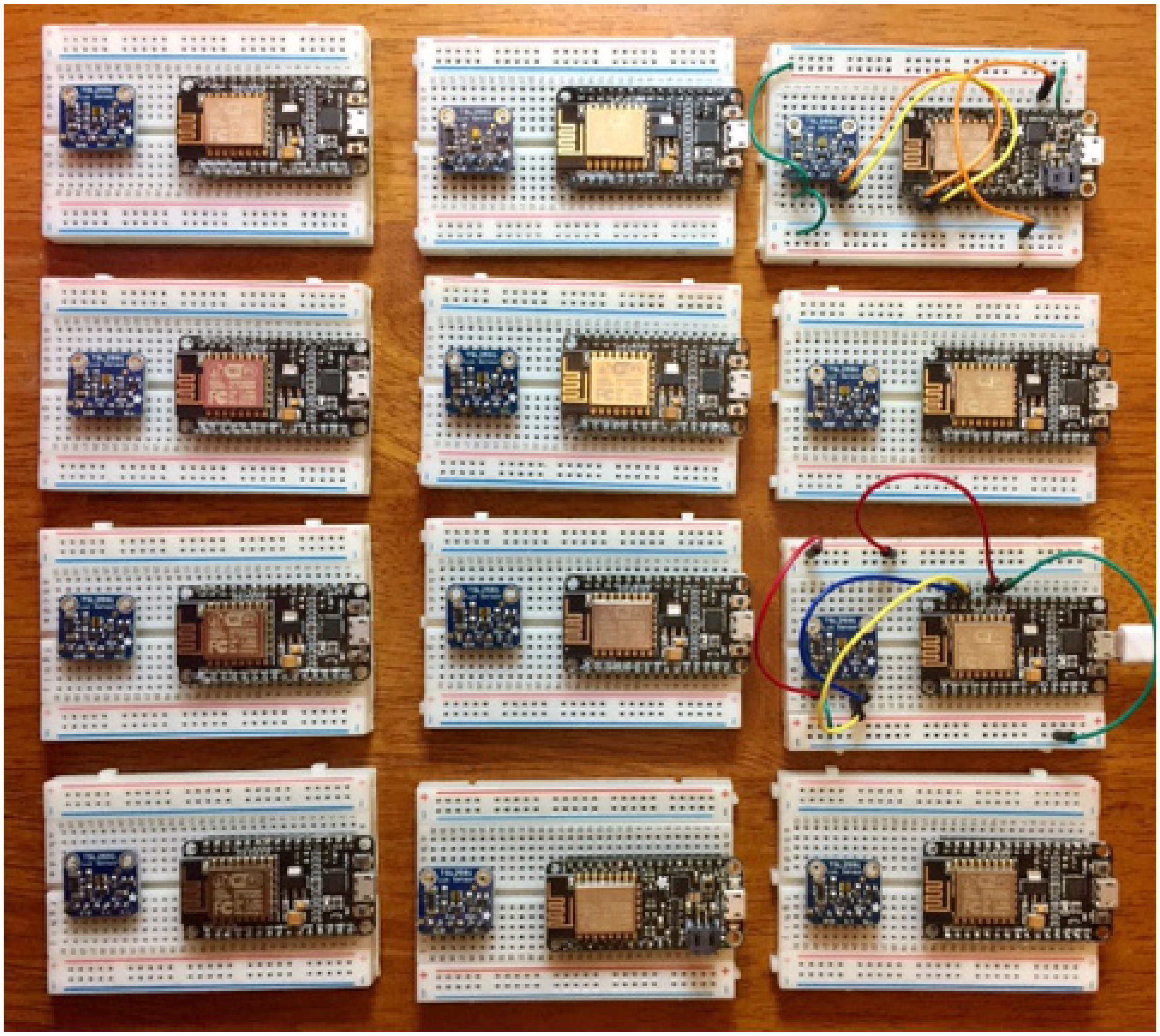
Announcement

Here is a sample announcement that can be used for the Hackathon:

> Due in large part to the availability of cheap, low-power, internet-connected microcontrollers, the Internet of Things is taking off. Python developers can get in on the excitement with MicroPython, an implementation of Python 3 that runs on very small devices with no operating system. MicroPython provides the standard Python REPL (read-eval-print-loop) interface, so you can interactively develop and debug applications on these small devices.
>
> In this session, you will learn some basic electronics, wire up some sensors to a low-power wireless controller board (based on the ESP8266 microcontroller), load the MicroPython firmware, and interactively write simple applications to read from the sensors. We will also discuss how to connect to other systems via the MQTT messaging protocol and exchange ideas on larger projects that can be built at home for low cost with beginner-level knowledge.
>
> Hardware is provided, the only prerequisite is to bring a laptop with a USB port. Here is a picture of the completed project: https://data-ken.org/images/lighting-app-esp8266.png

Soldering and Hardware Preparation

The breakout boards from Adafruit (e.g the Feather HUZZAH with ESP8266 or the TSL2591 light sensor) need to have their headers soldered to the boards. Unless your facility is a hardware lab set up for this kind of thing, we recommend doing the soldering in advance. For the first running of the hackathon, we soldered the boards and seated them on the breadboard. Here's what the set of boards looked like (10 student boards, 1 demo board, 1 spare):



I verified that each board was functional by flashing the latest version of MicroPython to it. I did not test the light sensors, but did have a spare.

Lecture

The repository containing this documentation also contains a lecture presentation (`lecture.pptx` and `lecture.pdf`). We recommend walking through this presentation before the hands-on section. The goal is to provide students with some context and an overview of the project.

CHAPTER 9

9. Quick Software Setup

This set of steps is designed as a fast alternative initial software and firmware setup. They are particularly recommended when time is a concern, or blocking issues arise.

The script make_zip.sh at the root of this repo was used to build micropython-iot-software.zip, removing network dependency and improving reliability. The instructions below assume that you have this zip file available, either created through make_zip.sh or given to you by your instructor.

To check that the light sensor is working, you will need to wire up your board and connect it to your USB port, as described in the *hardware assembly* chapter.

We include two version of the instructions, one for Debian 9 Linux and one for MacOS. If you have another system, please follow the full instructions in *Firmware and Testing*. You should still be able to use the files from the zip, as most are portable across OSs. Look at *here* for OS-specific needs.

Debian 9 Linux

We copy-and-paste tested these instructions using a modern Debian 9 with python 3.6.2 (from sid) and the screen program.

```
# Create project directory and virtual environment
mkdir -p ~/micropython-iot-hackathon
cd ~/micropython-iot-hackathon
python3.6 -m venv venv

# Activate python virtual environment [Notice the (venv) prompt indicating
# you are now within the virtual environment]
source venv/bin/activate

# Install software
unzip micropython-iot-software.zip
pip install --upgrade micropython-iot-software/python-tools/*whl

# Upload to latest firmware. (Use /dev/tty.SLAB_USBtoUART for OSX)
```

```
esptool.py --port /dev/ttyUSB0 erase_flash
esptool.py --port /dev/ttyUSB0 --baud 460800 write_flash --verify \
  -fm dio 0 micropython-iot-software/esp8266-20170612-v1.9.1.bin

# Enter the micropython REPL using the terminal  (and press Enter key to
# see the promt ">>>".  To exit, use Ctrl-a + \ + y)
screen /dev/ttyUSB0 115200
>>> print('Hello world')
[Exit screen, using: Control-A + k + y]

# Upload libraries to micropython
export AMPY_PORT=/dev/ttyUSB0
ampy put micropython-iot-software/micropython lib
ampy put micropython-iot-software/micropython/client.py main.py

# Check data can be read
screen /dev/ttyUSB0 115200
>>> from tsl2591 import Tsl2591
>>> tsl = Tsl2591('lux-1')
>>> tsl.sample()
[Exit screen, using: Control-A + k + y]
```

MacOS

First, download and install the Anaconda for Python 3.6 distribution from https://www.continuum.io/downloads. We will use the `conda` virtual environment tool instead of the standard Python 3 `venv`.

To communicate with the ESP8266 over your Mac's USB port, you will also need to install a serial driver (found in your zip distribution at `micropython-iot-software/drivers/Mac_OSX_VCP_Driver.zip`).

This was tested on MacOS Sierra.

```
# Create project directory and virtual environment
mkdir -p ~/micropython-iot-hackathon
cd ~/micropython-iot-hackathon
# Create virtual environment. Enter "y" when asked whether to proceed.
conda create -n micropython-iot-hackathon python=3.6 anaconda

# Activate python virtual environment [Notice the (micropython-iot-hackathon)
# prompt indicating you are now within the virtual environment].
source activate micropython-iot-hackathon

# Install software
unzip micropython-iot-software.zip
pip install --upgrade micropython-iot-software/python-tools/*whl

# Upload to latest firmware. (Use /dev/tty.SLAB_USBtoUART for OSX)
esptool.py --port /dev/tty.SLAB_USBtoUART erase_flash
esptool.py --port /dev/tty.SLAB_USBtoUART --baud 460800 write_flash --verify \
  -fm dio 0 micropython-iot-software/esp8266-20170612-v1.9.1.bin

# Enter the micropython REPL using the terminal  (and press Enter key to
# see the promt ">>>".  To exit, use Ctrl-a + \ + y)
screen /dev/tty.SLAB_USBtoUART 115200
>>> print('Hello world')
[Exit screen, using: Control-A + k + y]
```

```
# Upload libraries to micropython
export AMPY_PORT=/dev/tty.SLAB_USBtoUART
ampy put micropython-iot-software/micropython lib
ampy put micropython-iot-software/micropython/client.py main.py

# Check data can be read
screen /dev/tty.SLAB_USBtoUART 115200
>>> from tsl2591 import Tsl2591
>>> tsl = Tsl2591('lux-1')
>>> tsl.sample()
[Exit screen, using: Control-A + k + y]
```

Have fun!

CHAPTER 10

Copyright and License

CHAPTER 11

Acknowledgements

The following people have contributed to this repository (listed in chronological order):

- Jeff Fischer
- Simeon Franklin
- Eric Theise
- Daniel Mizyrycki

We thank all the contributers for their help! Thank you also to those who have participated in or facilitated past IoT Hackathons.

CHAPTER 12

Indices and tables

- genindex
- modindex
- search