Professor Dr. Zoltan Nagy
https://nagy.caee.utexas.edu

Introduction to Programming & The Internet of Things

Module 4: Data Acquisition & The Internet of Things

1 Modules Background

There is a pervasive need to study the performance of built structures and their abilities to provide occupants with comfort in an efficient manner. The Internet of Things (IoT) provides methods of obtaining and studying data about our built environments. As such, these modules introduce civil, architectural, and environmental engineers to tropics of electrical engineering. Students will gather the abilities needed to program and deploy necessary sensors in the Internet of Things (IoT), as well as gain the ability to gather and analyze data collected.

Contents

2 Introduction

In this module, we're going to explore ways to collect data and analyze it. From a high level, we're going to talk about where you should look when starting to use a sensor, how to incorporate libraries into your Arduino code, and how to format this data in a way we can use with something like excel. Likewise, we'll touch on a few different protocols and Often you'll find that there are already a lot of great utilities to use that will expedite the process of connecting sensors and make our lives a lot easier. Without further ado, let's get coding!

3 Protocols

3.0.1 Analog Voltage / Variable Resistance

The easiest and most basic form of collecting data is analog voltage. Many sensors like photo resistors (light sensitivity) or hall effect sensors (magnetic field) usually output analog voltage. This means that the sensor has been able to convert some physical system like pressure or light into a change in resistance. Putting this into a voltage divider, where it's visible that as one of the R values changes the Vout changes proportionally). The Arduino comes with a few analog voltage reading inputs, which makes reading analog voltages extremely easy.

Pros: Extremely easy, super rapid prototyping.
Cons: Not totally supported, requires more pins, perhaps voltage regulation.

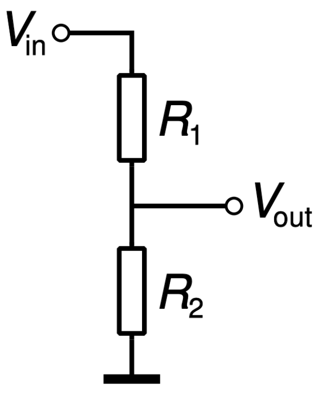


Figure 1: Analog Voltage Example Setup

3.0.2 I2C

There are a few protocols with are a lot more popular in industry than others, and the crowd favorite for ease of use and functionality is I^2C (Eye - Squared - C). This is also called "Two wire interface", and one of the benefits of this protocol is that it can be daisy chained to contain a number of devices using just two main wires for transmission.

This configuration uses "pull-up" resistors, which basically tells us that our transmission voltage will be high unless it is pulled low by the slave devices. In English? Think about how a city bus works. There are a bunch of people who need to communicate to the bus driver when they need to get off. To do so, they reach up and give a tug on the cable. This tug communicates to the bus driver some kind of information. After the person tugs the cable, it is restored to the original position of being relatively taught near the top of the bus ceiling. It might seem like this explanation is a bit drawn out, but there are a lot of parallels in I^2C communication.

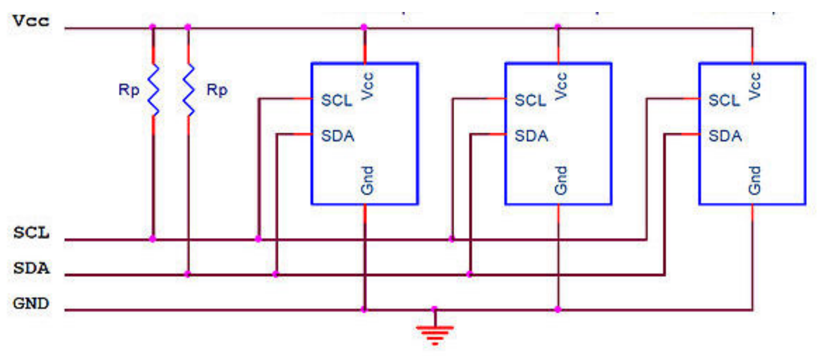


Figure 2: Example of I2C Configuration

When you connect a peripheral device (any kind of sensor or communication system) to the main device (what is called the "master" node), it's like the sensor is getting on the bus (which is conveniently also called the "bus" line). When it needs to communicate to the master node, it also pulls the line down. But instead of pulling the cable low, it's pulling the voltage line low. To restore the voltage to the original "high" voltage, a "pull-up" resistor is used. This allows the line to be restored to the original high voltage when our peripheral isn't asserting it low. Sometimes the device has these pull-up resistors built into a circuit board (as is our case for the tsl2561) so we don't need to think about this too much.

If you'd like to do more research on how I^2C works, here's a good place to start [1]. Luckily, most of the necessary code for communicating with sensors using I^2C is already written, you just need to know where to find it!

Pros: Easy, generally universally supported with sensors, only two wires for interfacing,
Cons: Small learning curve, requires compatible sensors (although they generally are), and consistent voltage source, sometimes requires pull-up resistor.

3.0.3 SPI

SPI (S - P - Eye) is good when you need a high data transmission. Think something like a camera, screen, or lidar sensor. The way it can achieve this high data transfer, however, is through an increased number of wires.

Pros: Great for high data transmission systems, generally supported.
Cons: Requires more pins, higher learning curve, not as commonly supported, **requires microcontroller that's compatible**.

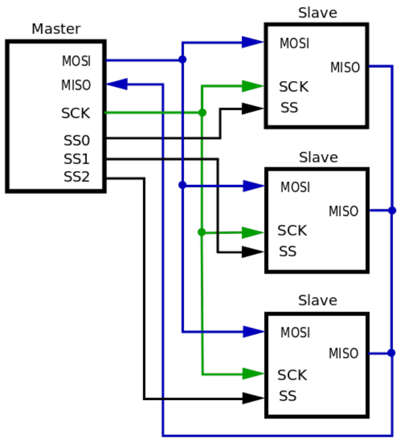


Figure 3: Example of SPI Configuration

3.1 Example

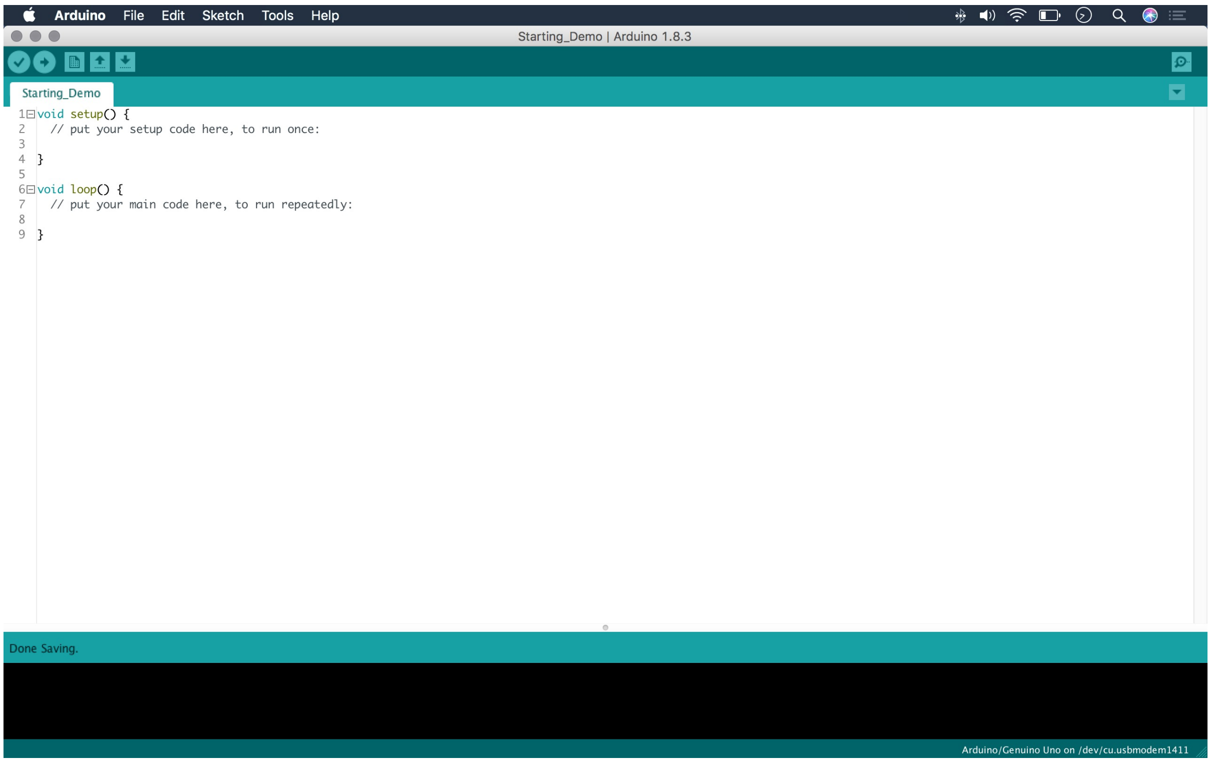
3.1.1 Problem and Research

Let's say you want to hook up a light sensor. The tsl2561 for example! Where you you start? Often, the first place you'll look is on github. By searching "tsl2561" in Google, a github repository is usually the first thing that pops up. "Github" is a great resource that will come up a lot when coding, so it's good to have a basic idea of what it does. Github is a place for sharing and working collaboratively on code, and is used widely in both industry and in academia. It's quite famous for its version control system, which saves every iteration of the code over time (so if you mess up, there's an old copy). In this case, we're capitalizing on the availability of public code that github provides.

I found this github library using google: [2], which we're going to use in our system as a primary reference and tool. Likewise, you can reference this spec sheet if you'd like to find information about limitations and operating principles for this sensor: [3].

4 Simulating Tools

Once you have all of the necessary resources gathered, it's time to start building our system. Often it's useful to simulate a circuit before building it. A great tool is provided by Autodesk [4]. At the time of writing, there is a reasonably sized library of prebuilt sensor code from which you can find inspiration. Likewise, there's a program called Fritzing [5] which offers much of the same simulation setup and perhaps even more supported devices.


Arduino File Edit Sketch Tools Help
Starting_Demo | Arduino 1.8.3
Starting_Demo

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

```
Done Saving.
```

```
Arduino/Genuino Uno on /dev/cu.usbmodem1411
```

Figure 4: Blank Arduino IDE.

5 Arduino IDE

The Arduino platform was chosen for these modules in large part because of their accessibility, ease of use, and free software.

The Arduino software [6] contains all of the software you'll need, with support for every platform. It is **highly** recommended that you download the software if you would like to work within the Ardunio platform. After we've downloaded the software, it should look something like figure 4!

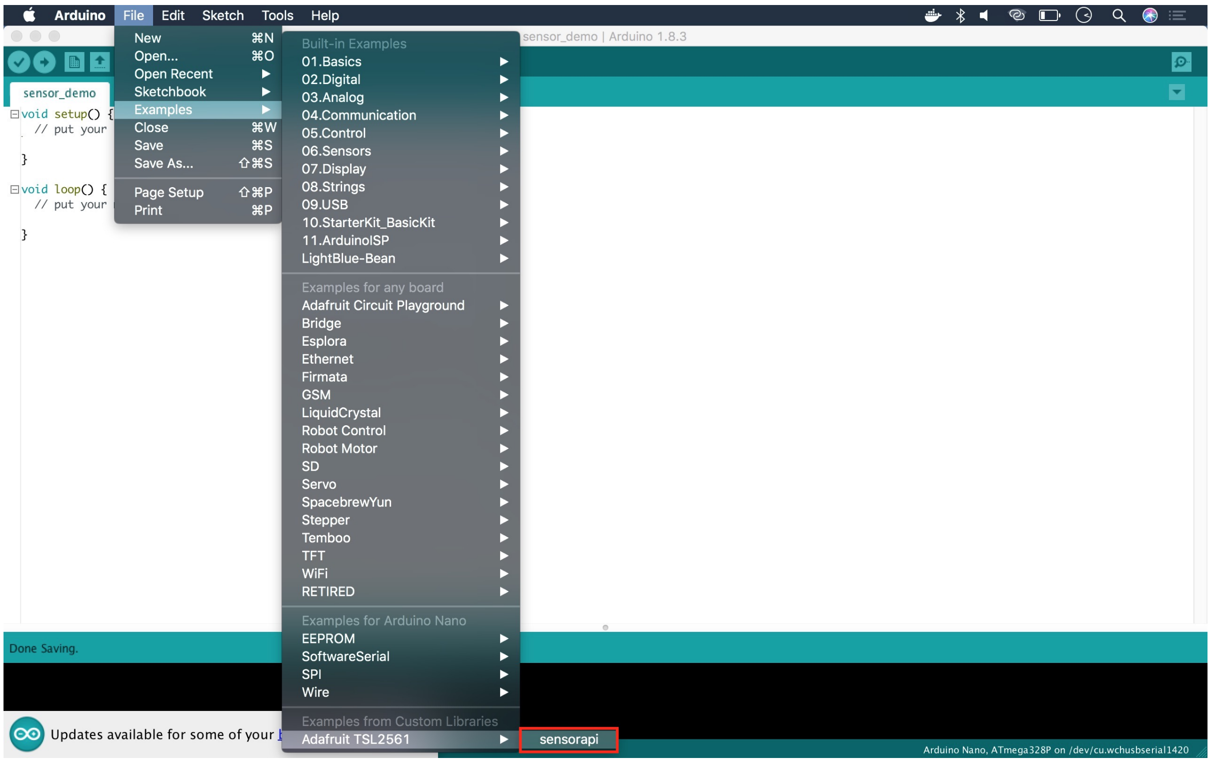
5.0.1 Libraries using the Arduino IDE

Some of the more popular and common libraries have local support for them from within the Arduino IDE. The hookup guide on Adafruit's website actually shows this process quite clearly: [7]. **This is the easiest way** to get a sensor up and running if available. If it's not a highly supported sensor like those on Adafruit, the next place to look is if the sensor has code up and running on Github.

5.0.2 From Github to the Ardunio

Downloading a zip file of the repository from Github will give you a folder full of files. See figure 6 for where to do this on github's website. Once you download your file, it will be found in your "Downloads" folder as "Adafruit_TSL2561-master.zip". Here are the steps we need to take in order to have our Ardunio IDE see this and allow us to run this code on the Ardunio:

1. Double clicking this on a mac will "unzip" the file, and the same can be done with Windows by right clicking and selecting "Extract All".
2. You'll now find that there's a file called "Adafruit_TSL2561-master" in the same location on your computer which is a folder (or "directory"). Let's rename this to remove the "-master", so our final folder name is "Adafruit_TSL2561".



```
void setup() {
  // put your
}

void loop() {
  // put your
}
```

Done Saving.

Updates available for some of your

Arduino Nano, ATmega328P on /dev/cu.wchusbserial1420

Figure 5: TSL2561 example file location.

3. The final step is to move this folder into the place our Arduino app will be expecting to see it. On a mac, this location can be found in your "Documents" folder. Opening your finder, you'll find this on the left sidebar. See figure 7 for a guide. On a PC, this libraries folder will be located in "Documents/My Documents". In either case, if the folder doesn't exist then make it in this location.

4. Drag this folder (Adafruit_TSL2561) into the Arduino/libraries folder. Now you'll be able to find the script in your Ardunio IDE!

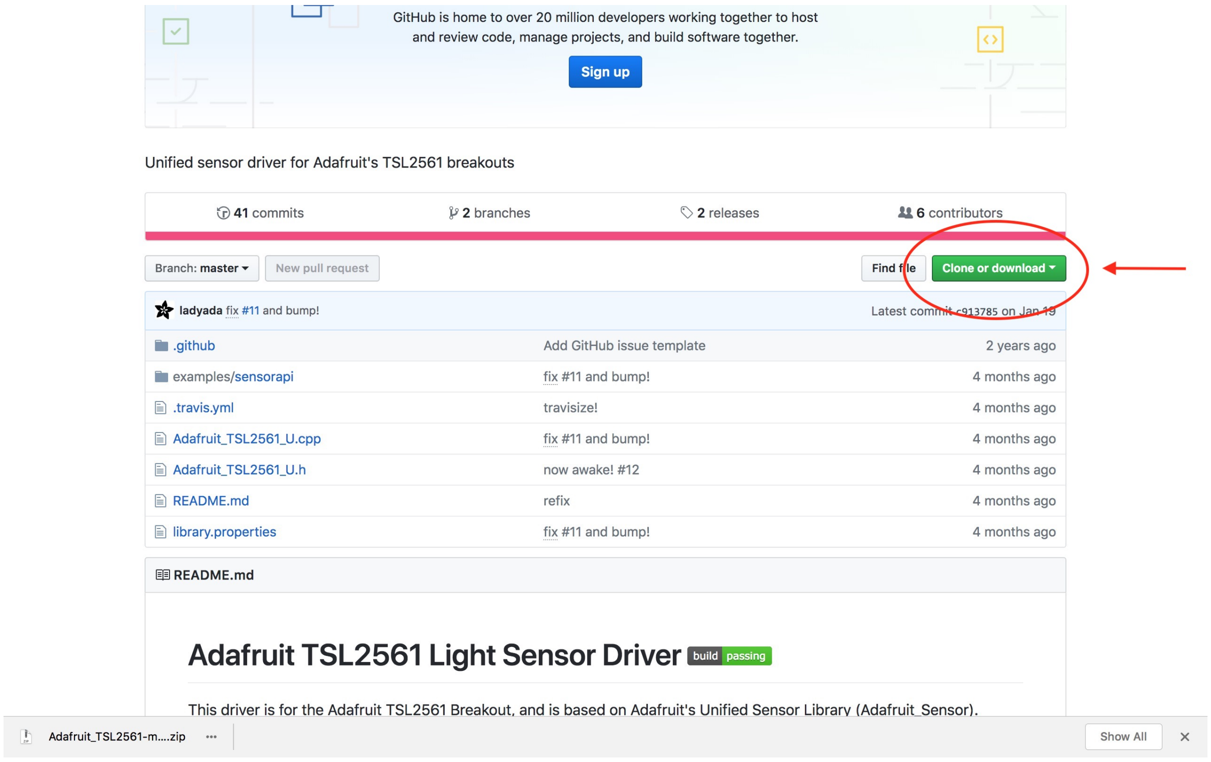
If these steps didn't work for you, then I'd recommend consulting this guide [8]. In general, Adafruit is going to be a great resource for troubleshooting and step by step guides for your sensors.

6 Running and Monitoring Data

The example code given in sensorapi is feeding values to the serial monitor (See figure 8), which basically means it's sending every value it gets to your computer through the cable. We can see this by clicking on the "Serial Monitor" button in the top right of the Arduino IDE (it looks like a magnifying glass). Run the code and then look at the serial monitor to verify that your sensor is up and running and the values seem appropriate. You'll know the code has run correctly when you see the two lines Sketch uses... and Global variables... that you see in figure 8.

6.1 Common Errors

The Arduino IDE needs to make sure it's looking for your physical Arduino computer in the right place. After you plug an Arduino Uno into your computer via USB, your computer will recognize that there's something in the USB socket. Likewise, it needs to know which kind of computer you're plugging in so it can upload the right way. Both of these issues are explained in this walk through [9].


GitHub is home to over 20 million developers working together to host and review code, manage projects, and build software together.

Sign up

Unified sensor driver for Adafruit's TSL2561 breakouts

41 commits | 2 branches | 2 releases | 6 contributors

Branch: master ▾ | New pull request | Find file | Clone or download ▾

ladyada fix #11 and bump! | Latest commit c913785 on Jan 19

.github	Add GitHub issue template	2 years ago
examples/sensorapi	fix #11 and bump!	4 months ago
.travis.yml	travisize!	4 months ago
Adafruit_TSL2561_U.cpp	fix #11 and bump!	4 months ago
Adafruit_TSL2561_U.h	now awake! #12	4 months ago
README.md	refix	4 months ago
library.properties	fix #11 and bump!	4 months ago

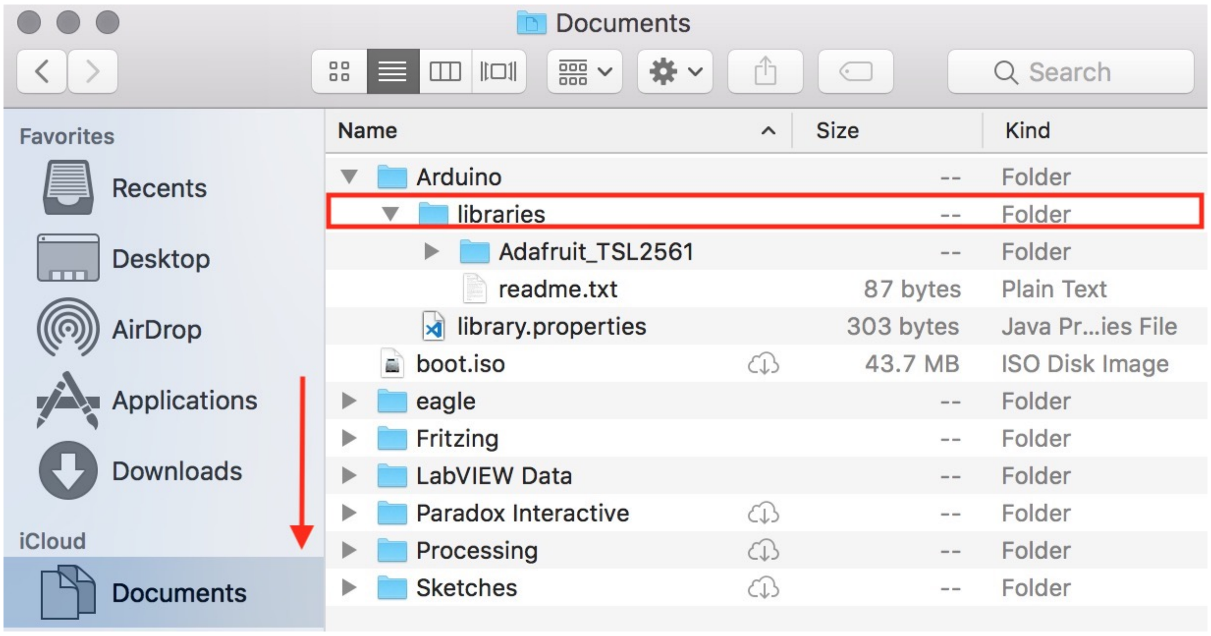
README.md

Adafruit TSL2561 Light Sensor Driver build passing

This driver is for the Adafruit TSL2561 Breakout, and is based on Adafruit's Unified Sensor Library (Adafruit_Sensor).

Adafruit_TSL2561-m....zip ⋯ | Show All ×

Figure 6: Download Here.


Documents

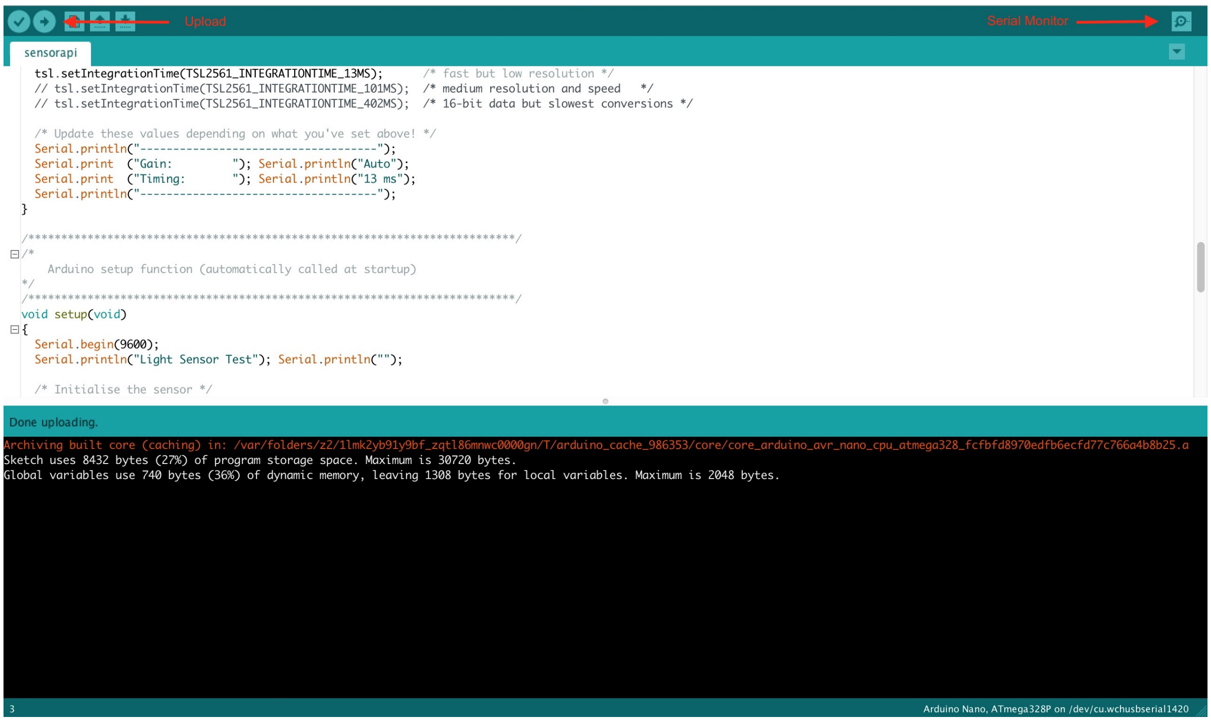
Search

Favorites: Recents, Desktop, AirDrop, Applications, Downloads

iCloud: Documents

Name	Size	Kind
▼ Arduino	--	Folder
▼ libraries	--	Folder
▶ Adafruit_TSL2561	--	Folder
readme.txt	87 bytes	Plain Text
library.properties	303 bytes	Java Pr...ies File
boot.iso	43.7 MB	ISO Disk Image
▶ eagle	--	Folder
▶ Fritzing	--	Folder
▶ LabVIEW Data	--	Folder
▶ Paradox Interactive	--	Folder
▶ Processing	--	Folder
▶ Sketches	--	Folder

Figure 7: Ardunio Folder Location - Mac.



```
  tsl.setIntegrationTime(TSL2561_INTEGRATIONTIME_13MS);      /* fast but low resolution */
  // tsl.setIntegrationTime(TSL2561_INTEGRATIONTIME_101MS);  /* medium resolution and speed   */
  // tsl.setIntegrationTime(TSL2561_INTEGRATIONTIME_402MS);  /* 16-bit data but slowest conversions */

  /* Update these values depending on what you've set above! */
  Serial.println("------------------------------------");
  Serial.print  ("Gain:         "); Serial.println("Auto");
  Serial.print  ("Timing:       "); Serial.println("13 ms");
  Serial.println("------------------------------------");
}

/**************************************************************************/
/*
    Arduino setup function (automatically called at startup)
*/
/**************************************************************************/
void setup(void)
{
  Serial.begin(9600);
  Serial.println("Light Sensor Test"); Serial.println("");

  /* Initialise the sensor */
```

```
Archiving built core (caching) in: /var/folders/z2/1lmk2yb91y9bf_zqtl86mnwc0000gn/T/arduino_cache_986353/core/core_arduino_avr_nano_cpu_atmega328_fcfbfd8970edfb6ecfd77c766a4b8b25.a
Sketch uses 8432 bytes (27%) of program storage space. Maximum is 30720 bytes.
Global variables use 740 bytes (36%) of dynamic memory, leaving 1308 bytes for local variables. Maximum is 2048 bytes.
```

Figure 8: Location of run button and serial monitor.

7 Recording Data

The easiest and quickest way to record data is by copying what's in the serial monitor and pasting it into excel. This is probably the way you'll want to go if you're testing a sensor for a system. If you're recording long-term data, this might be less ideal. For those cases, you'll more than likely want to use SD card storage. This offers more permanent and larger data capturing ability.

Given what you've learned already with I2c, Github, and the Arduino IDE, I think it's safe to say you're ready to build a data logger! This guide might come in handy for that project [10]. Likewise, there are some prebuilt arduino boards with SD card support [11], which could be a good option when long term storage is necessary. Remember that you're probably not the first person to run into any technical issues so look online if you get stuck!

8 Example Questions

8.1 Light

Use a photo-resistor [12], build a system capable of measuring light intensity. [13]

8.2 Air Quality

While Volatile Organic Compounds (VOCs) [14] aren't exactly the perfect metric for measuring air quality, they're primary components in smells, plant communication, and might give some indication of harmful chemicals. Hook up a VOC sensor using I^2c. [15].

References

[1] i^2c overview and more information. https://learn.sparkfun.com/tutorials/i2c.

[2] Github page for tsl2561. https://github.com/adafruit/Adafruit_TSL2561.

[3] Spec sheet for the tsl2561. https://cdn-shop.adafruit.com/datasheets/TSL2561.pdf.

[4] Autodesk circuit simulation software. https://www.tinkercad.com/.

[5] Fritzing simulation software. http://fritzing.org/download/.

[6] Arduino software. https://www.arduino.cc/en/Main/Software.

[7] Arduino wiring guide for tsl2561. https://learn.adafruit.com/tsl2561/arduino-code.

[8] Arduino libraries guide. https://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use/how-to-install-a-library.

[9] Arduino common errors. https://www.arduino.cc/en/Guide/Arduino101#toc4.

[10] Arduino data logger tutorial. https://learn.adafruit.com/adafruit-data-logger-shield/overview.

[11] Arduino board with sd card. https://learn.adafruit.com/adafruit-feather-m0-adalogger/using-the-sd-card.

[12] Photoresistor. https://en.wikipedia.org/wiki/Photoresistor.

[13] Photoresistor walkthrough. https://learn.adafruit.com/photocells/using-a-photocell.

[14] Voc wiki. https://en.wikipedia.org/wiki/Volatile_organic_compound.

[15] Air quality tutorial. https://learn.adafruit.com/adafruit-ccs811-air-quality-sensor.