Getting Started with PlatformIO

It's time to look at a more advanced development environment for programming our microcontrollers, so today we'll take a look at PlatformIO.

It's a bit of a learning curve, but well worth it, as PlatformIO has many advantages over using the Arduino IDE. I'll show you how to set it up and how to use it with the Arduino Uno, ESP32, and Seeeduino XIAO.

Introduction

When we begin working with the Arduino one of the first steps is to install the Arduino IDE (Integrated Development Environment). It's a fundamental piece of software that runs on Linux, Windows, or Mac OSX and it allows us to program our little microcontroller wonders to do just about anything.

The Arduino IDE has a lot going for it. It's very easy to use, especially for beginners, and it comes with a great assortment of sample sketches to get you going. By adding additional Boards Managers you can use it for more than just Arduino boards. And,

because it's so popular, you'll find an abundance of information to assist you on the web and on YouTube (and, of course, here on the DroneBot Workshop!).

But for all of its glory, the Arduino IDE does have many shortcomings.

- It lacks a debugger, a tool that allows you to insert breakpoints into your code and then observe the state of key variables when these points are reached.
- It requires you to manually determine which USB or serial port your microcontroller is connected to, sometimes (especially with Linux or Mac OSX) this is not obvious.
- It does not provide help such as auto-complete or built-in references to allow you to catch errors before you compile.
- It cannot be integrated with a code repository, such as GitHub.

For beginners, many of those features, like the debugger and Git integration, are not essential. But beginners can still benefit from features like auto-complete and an integrated code reference, as they help anyone write code with fewer errors.

Enter PlatformIO.

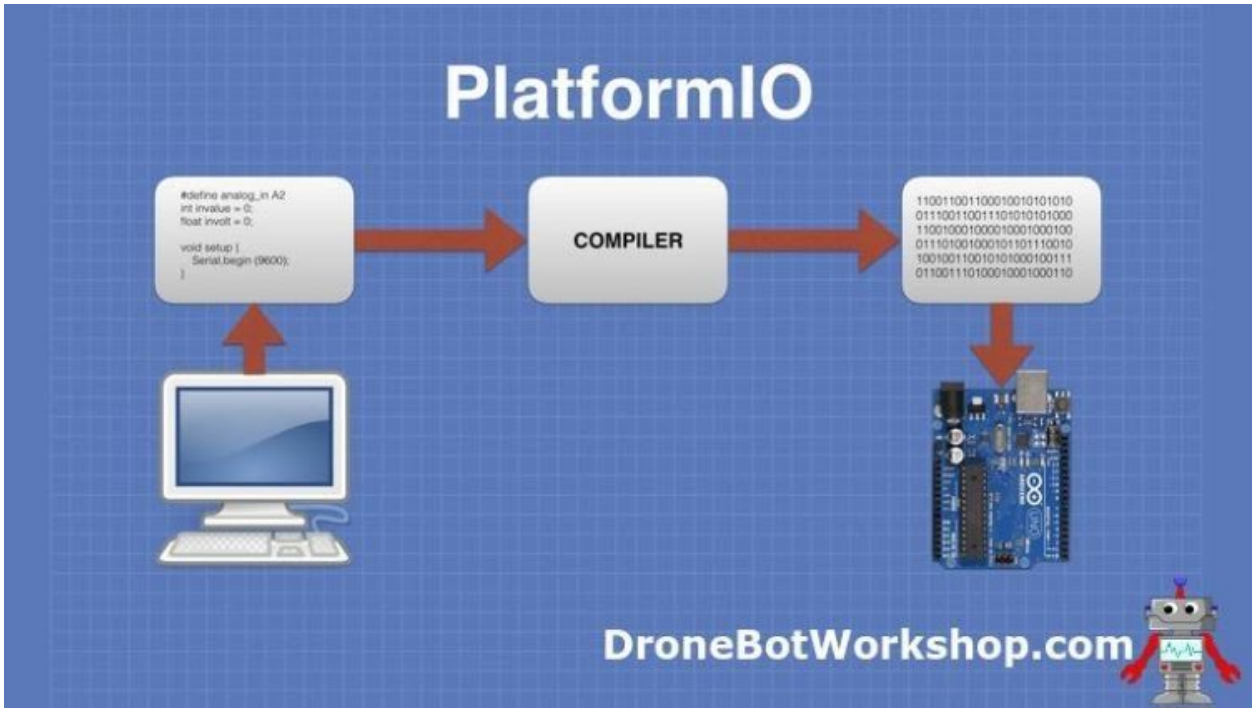
PlatformIO has those missing features, along with many more. So it is certainly worth taking a look at, no matter what your level of coding experience.

Before we begin, let's run over a few features common to many IDE's.

Integrated Development Environments

Developing code, whether it be for microcontrollers, mobile devices or desktop platforms, requires a number of common steps:

- You need to use some form of text editor to write the code in the desired programming language. The most common languages for writing microcontroller code are C++ and Python (or microPython).
- That text needs to be converted into machine-readable code, suitable for your target device. This is a job for either a *compiler* or an *interpreter*, depending upon which language you are using,
- That machine-readable code is then uploaded to the target device. In the case of interpreted languages, such as Python, the editor interfaces directly with the device and an interpreter translates the code every time it is run.

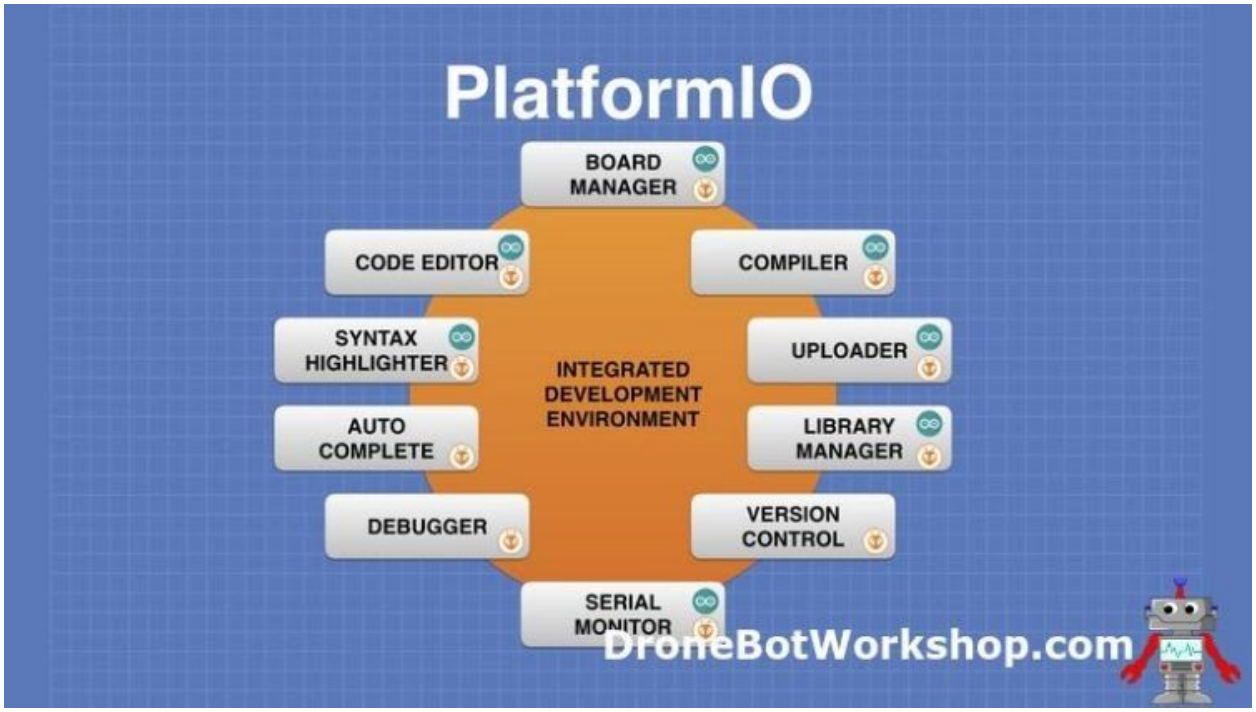


You can use a basic text editor and a command-line utility to do all of the above, but most people find it easier to use specialized GUI-based tools.

The Arduino IDE has all of the features you need to compose, compile, and upload code to your target microcontroller. It also has a Serial Monitor to observe activity on the microcontroller and to exchange commands with it. It can manage libraries and you can add alternate microcontrollers to it.

Sounds like it has all we need. So why switch?

Well despite all of its features the Arduino IDE is really just a basic IDE, and it is missing a lot of features that advanced editors and IDEs have. Things like auto-complete, which can save you some typing, inline error checking to catch your mistakes as you make them, and an onboard reference to help you understand your code.



PlatformIO has more features than the Arduino IDE, features that make it much easier to create and troubleshoot your code.

PlatformIO

According to the PlatformIO documentation "*PlatformIO is a cross-platform, cross-architecture, multiple framework, professional tool for embedded systems engineers and for software developers who write applications for embedded products*."

Put another way, this is a development tool that can run on most operating systems and under many different code editor packages. A few of the editors it runs under are:

- Visual Studio Code (VS Code)
- Atom
- Codeblocks
- Eclipse
- Netbeans
- Sublime Text

It can also be run on cloud-based packages like Codeanywhere and Eclipse Che.

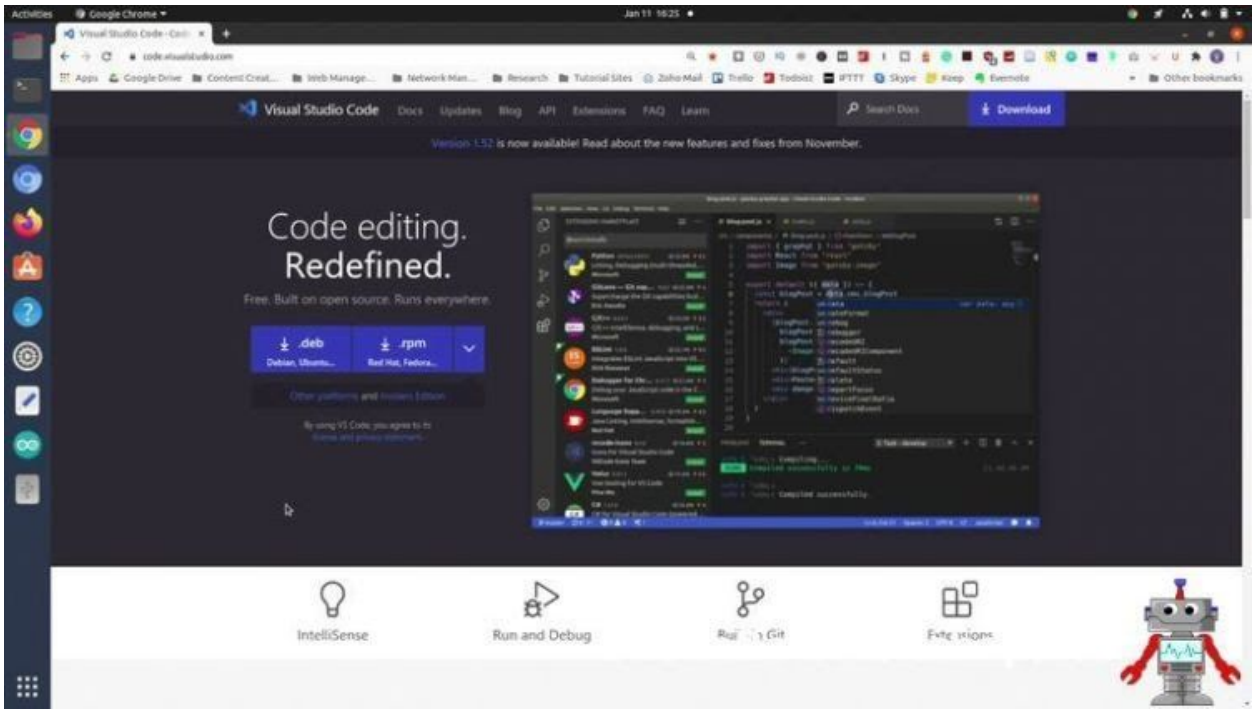
We will be running it under Microsoft Visual Studio Code, a free development platform available for Linux, Windows and Mac OS X.

Visual Studio Code

While many people don't associate Microsoft with free software they actually have created several free products, many of them development environments.

- Visual Studio Community is a free IDE for developing iOS, Android, Windows and web applications.
- Visual Studio Dev Essentials is a collection of tools, cloud services and software trials.

And, of course, Visual Studio Code, the free open-source code editor that can run on any operating system. This is the product that we will be running PlatformIO under.



Visual Studio Code includes IntelliSense, an advanced auto-complete and syntax highlighting system that can assist you in creating better code without errors. This allows you to catch and correct coding errors before you compile your code.

It also has a debugger, a software diagnostic tool that allows you to troubleshoot code that isn't working the way you expect it to. This is a more advanced feature that we won't be examining today.

One of the greatest features of Visual Studio Code, or VS Code, is that it supports extensions. These extensions allow you to add additional functionality to VS Code, enabling you to use VS Code for virtually any platform and coding language.

PlatformIO is an extension to VS Code.

Installing PlatformIO with VS Code

As we will be using the PlatformIO extension for VS Code the first thing we will need to do is get Visual Studio Code installed.

Another requirement for using PlatformIO is to have Python version 3.5 or higher installed.

The installation procedure differs depending upon your operating system, but it's pretty easy. Follow the instructions for your OS.

Linux Installation

There are two ways (at least) to install VS Code and Python on Linux.

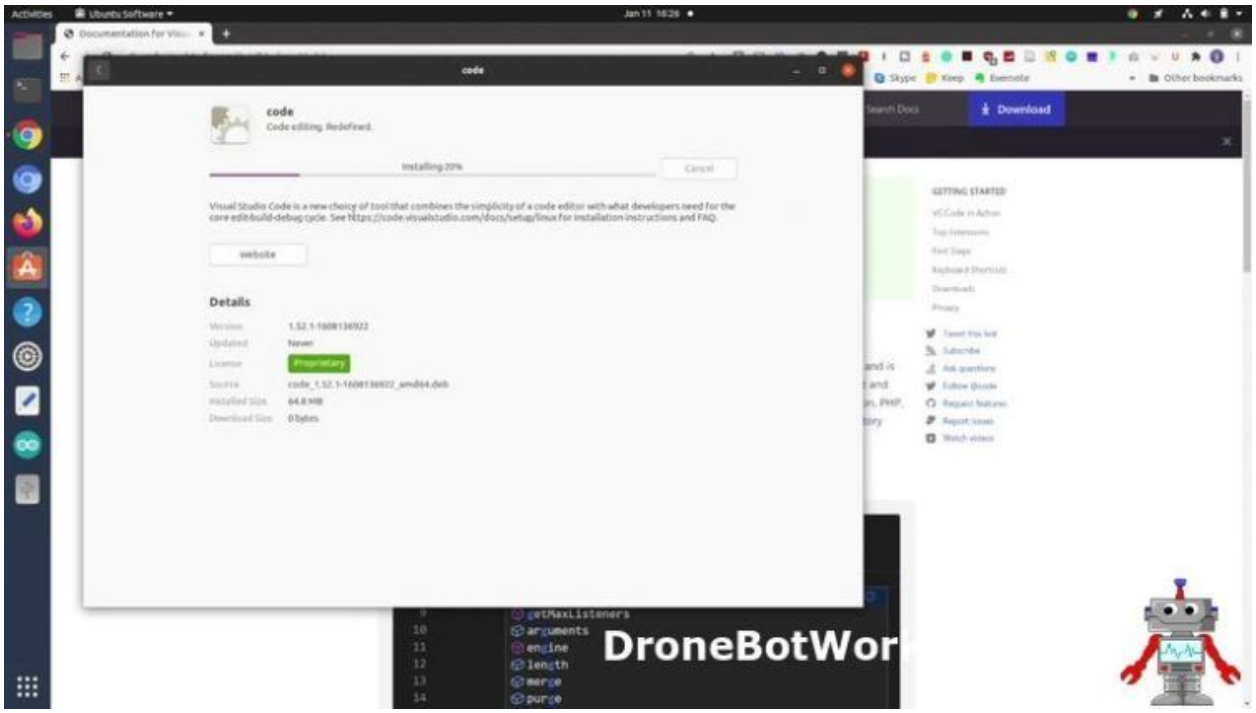
- Use the Snap Store snap. This also requires the installation of a Python Virtual Environment.
- Install from the file downloaded from the VS Code website. In most cases, the native Python on your Linux distribution will work without updating.

I described the first installation method, using the Snap store, in the article and video I published for Building a Developers Linux Workstation. So rather than repeat myself I'll refer you to that article if you want to use the Snap store. This method has the advantage of the VS Code updates being applied automatically when you do a system update.

The second method is to grab the installation file from the Visual Studio Code website. The site should defaut to the correct installation files for your operating system, but if you want a different version you can also go to the downloads section at the bottom of the page.

For Ubuntu and other Debian-based distributions of Linux, you'll want to grab the .deb file. If you're running a derivative of RedHat Linux then the .rpm file is what you need.

If you're running Ubuntu then you can just click on the .deb file once it finishes downloading. The Software Installer will open and you can click Install to begin the installation process.



You'll need to authenticate first, then the installer will add VS Code to your system.

With this type of installation method that's really all there is to it, as you likely already have Python 3. To verify this open a Terminal and type the following:

```
python3 --version
```

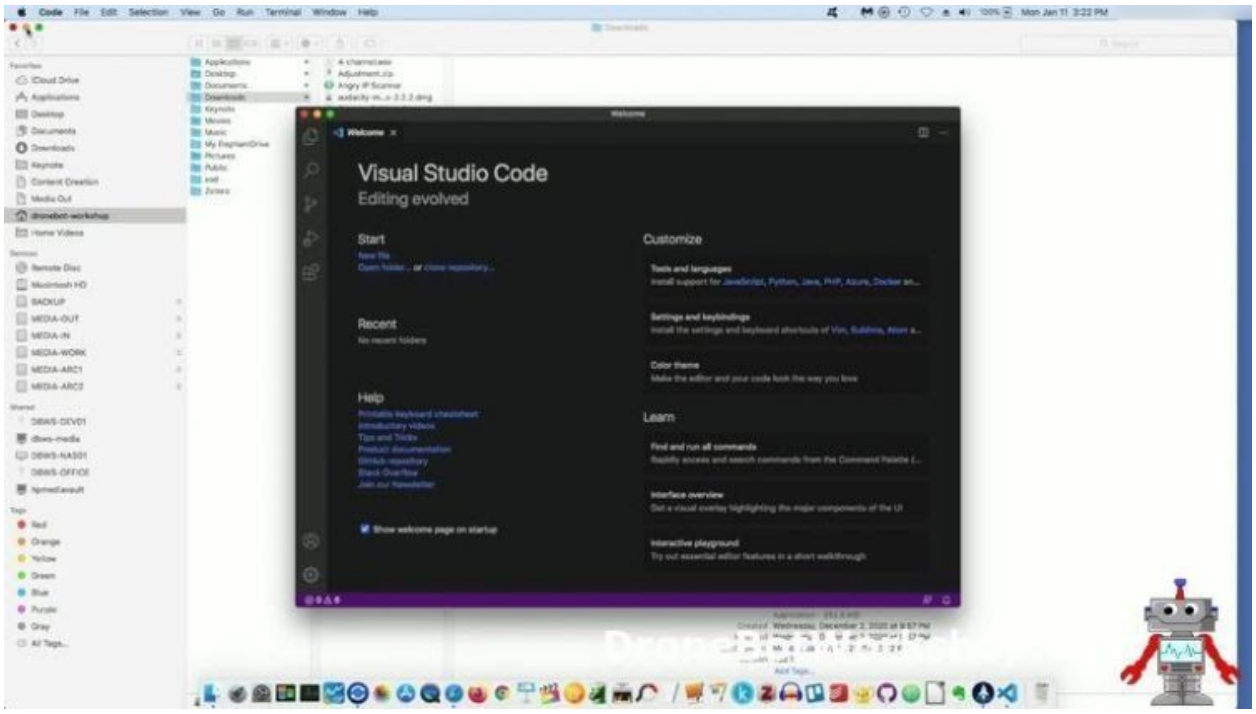
This will display your current version of Python 3. As long as it is at least version 3.5 you are good to go.

You can now skip ahead to the PlatformIO plugin installation procedures.

Mac OS X Installation

Installation of VS Code on Mac OS X is pretty simple, you'll be installing both VS Code and Python 3.

The first step is to visit the VS Code Website and download the application for Mac OS X, it should default to that when you load the page.

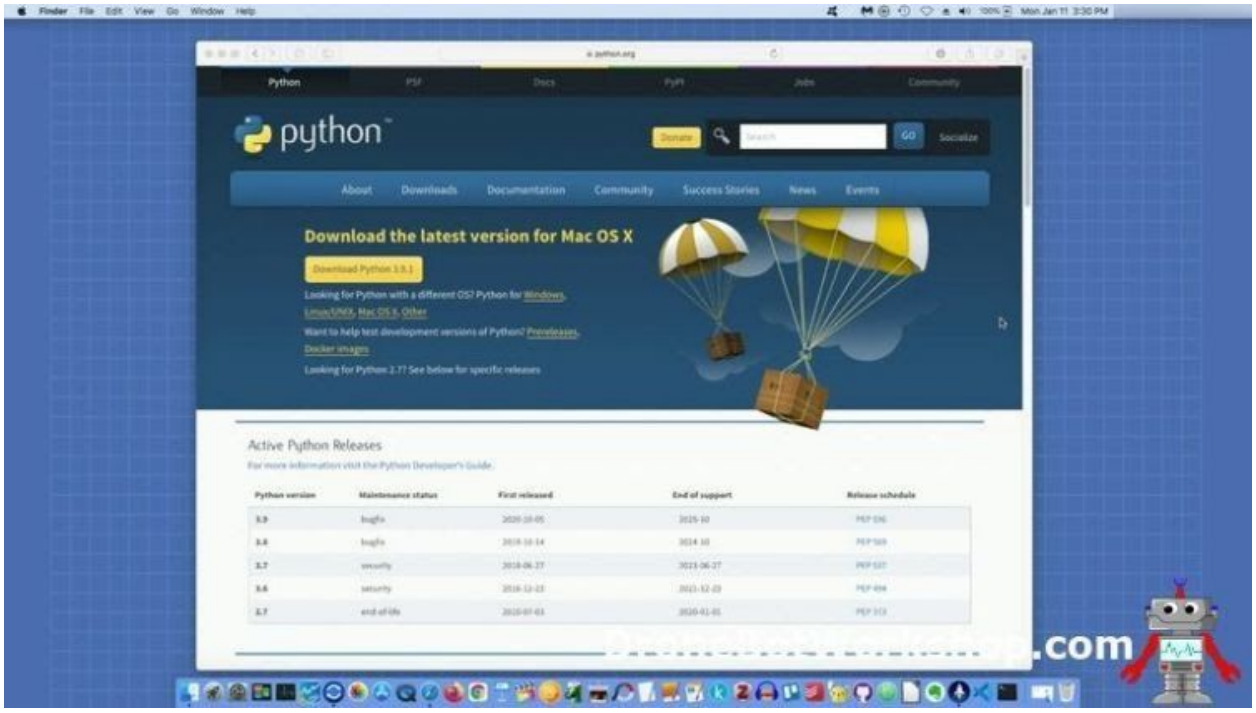


The download is a complete application, and you can run it directly as soon as it is installed. It would probably be a good idea to move it from your downloads folder into a more appropriate folder, like Applications.

The first time you run VS Code you'll need to agree to trust it, as it is a file downloaded from the Internet.

After getting Visual Studio Code you'll need to install Python.

Head over to the Python website downloads page. As with the VS Code website you should see a big button to download an installation package with the latest version of Python for the Mac.



This time you are downloading a package that you will need to install. Follow the instructions and accept the license agreement and the installer will run.

When the installer has finished you have an option to put the original install package into the trash, which is probably a good idea.

After you complete the Python installation, open your Terminal and use the same command we used in Linux to check the current version of Python:

```
python3 --version
```

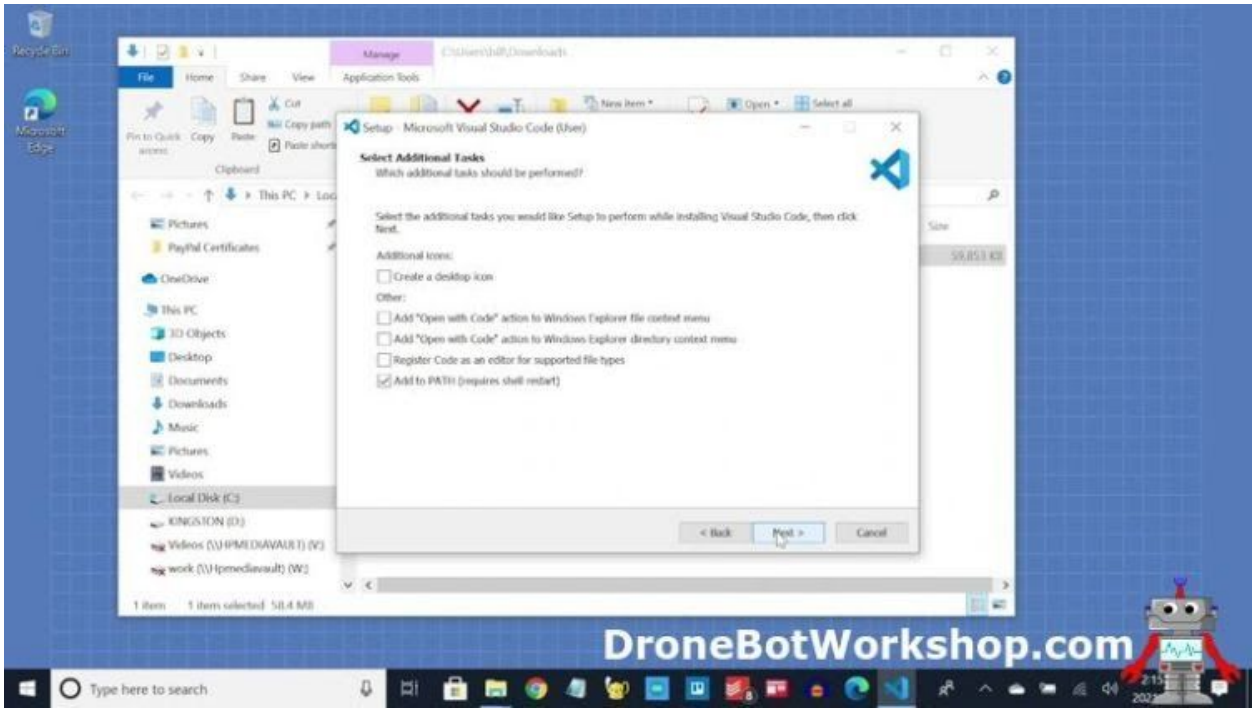
This will print back the version of Python, which should (hopefully) be above version 3.5. You are now ready to proceed to the installation of the PlatformIO plugin, detailed in a bit.

Microsoft Windows 10 Installation

The Windows 10 installation is also very simple. As with the Mac, you'll need to install both VS Code and Python 3.

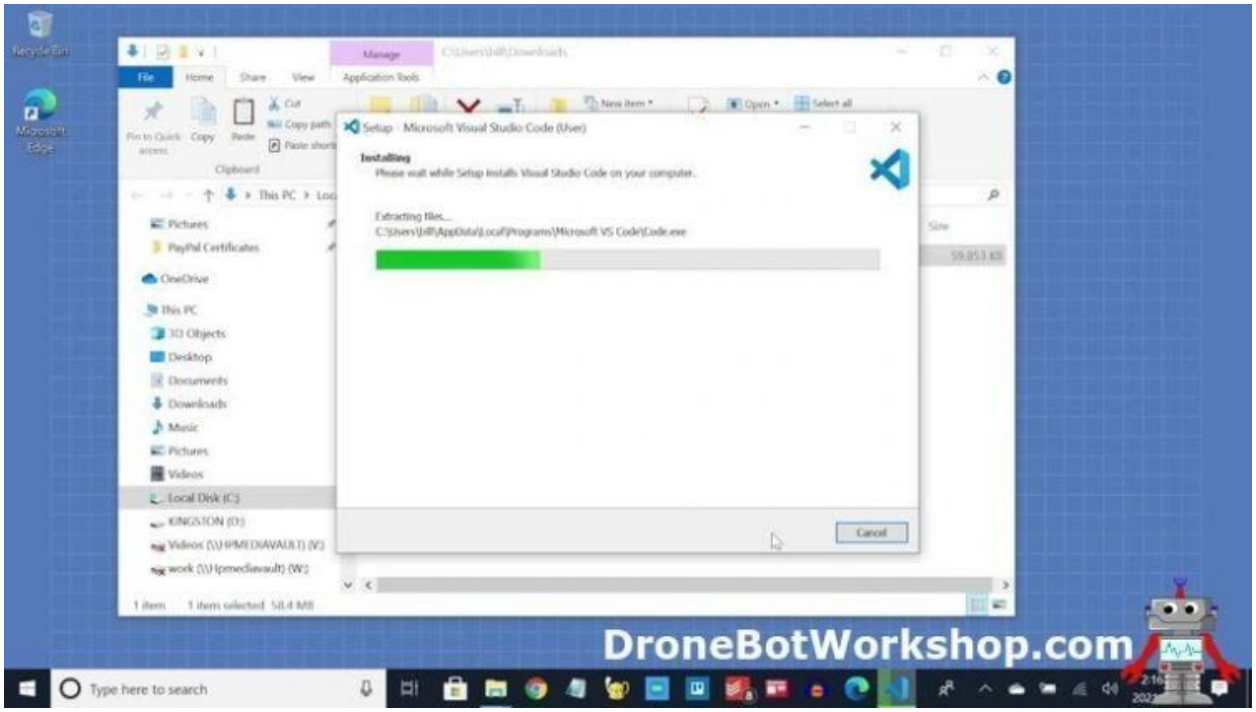
Once again you need to grab the installation program from the Visual Studio Code website. Click on the file you downloaded to start installing VS Code.

As with most Windows programs you'll need to select a location to install your new software and decide if you want it placed in the Start menu, unless you have a specific reason for changing these you can just accept the default settings. You'll also need to accept a license agreement.



One thing that is very important is to leave the box for “Add to PATH” checked on the *Select Additional Tasks* dialog box. You may select other choices here as well.

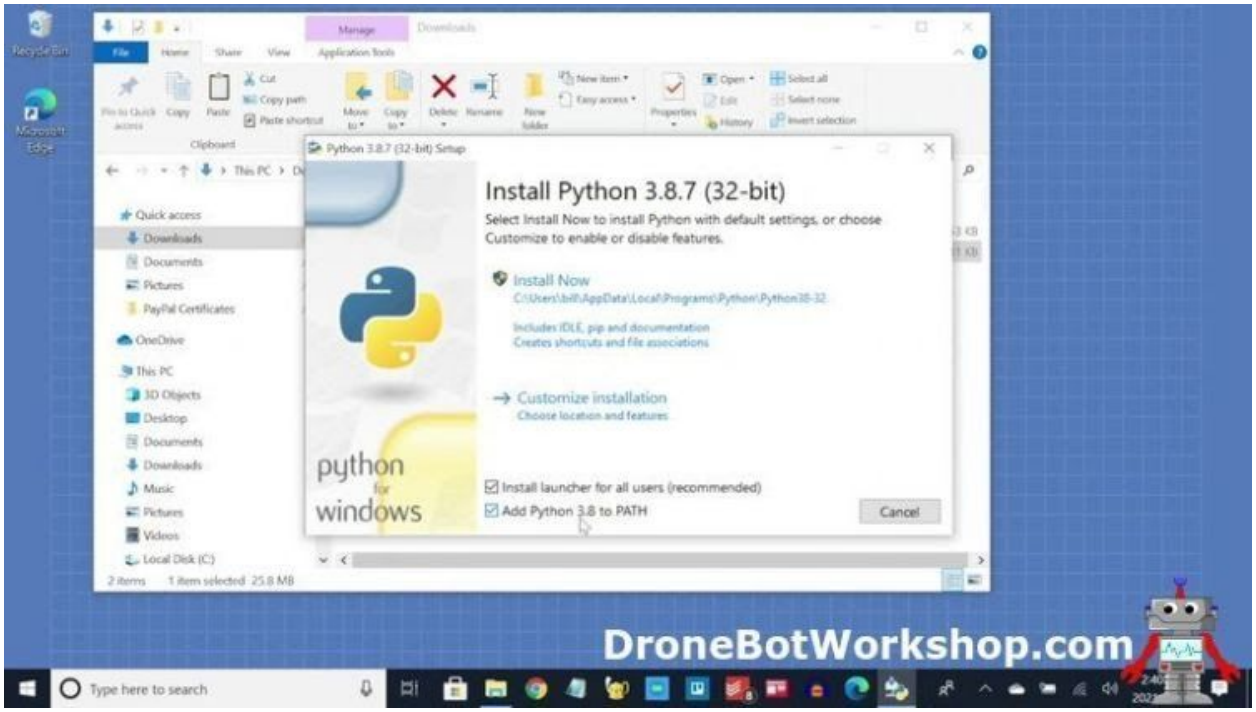
After making all your selections VS Code will begin to install. The installation can take a bit of time, so please be patient!



Once the installation is finished you have the option of opening Visual Studio Code, you may wish to do that just to ensure that everything installed correctly.

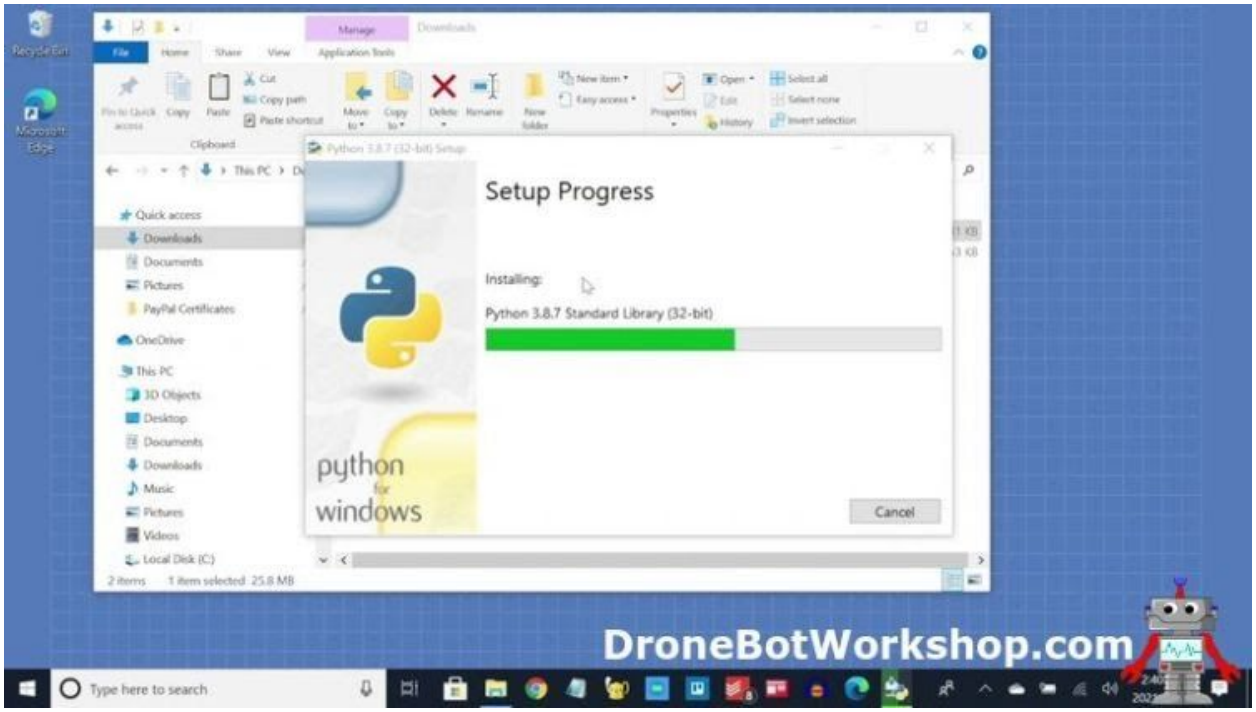
But we aren't done yet, we still need to install Python.

You can head over to the Python Download Page on python.org to get the installation file. One thing to note is that the processor you are using may not accept the 64-bit version of Python, even if you are running 64-bit Windows. You can then choose to grab the latest version of the 32-bit version of Python, it will work fine with PlatformIO.



On the dialog box with the “Install Now” selection, you’ll see a checkbox allowing you to “*Add Python 3.x to Path*”. You must have that selected for PlatformIO to function correctly.

After selecting the checkbox you can click “Install Now” and the installation will begin.



Once Python is installed you are all set to add the PlatformIO plugin to Visual Studio Code.

Install PlatformIO Plugin for VS Code

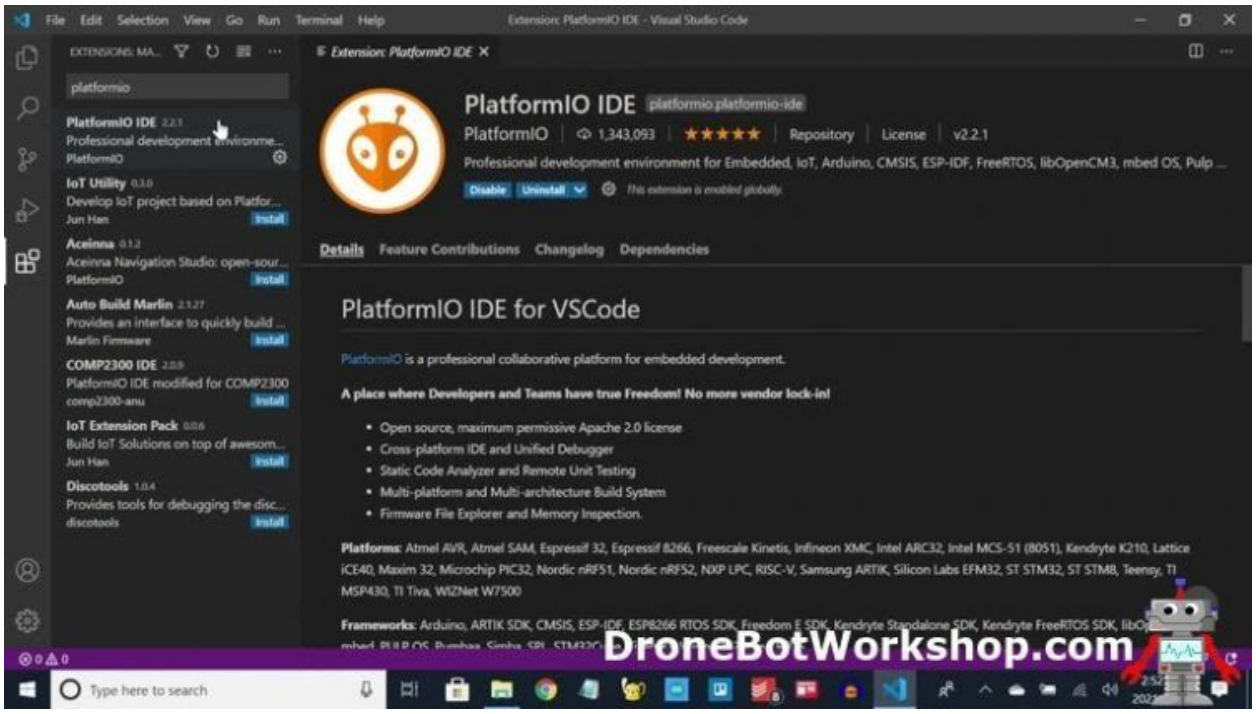
The instructions here apply to any operating system.

Open Visual Studio Code. You'll initially be greeted with a welcome screen that outlines some of the features of VS Code. You can close this screen after you finish examining it.

On the left side of the VS Code interface you'll see a number of icons. As you move your mouse over these icons their functions will be displayed.

The bottom icon (assuming you haven't installed any plugins yet) is shaped like a group of squares, this is the Extensions icon. Click on it, this will open another pane where you can search for extensions.

TYpe “platformio” in the search box. One of the items in the results will be “PlatformIO IDE”. There will be a blue “Install” button beside the search result. Click on it to install PlatformIO.



The PlatformIO plugin installation will take a little while, so be patient and don’t do anything in VS Code while it is installing. You can monitor the installation progress on the lower right side of the screen, at one point during the installation the Terminal area will also display some information.

Once the installation is finished you will need to close and reopen Visual Studio Code. When you open VS Code it will check all of its extensions, including PlatformIO. After it is done you are ready to use PlatformIO.

PlatformIO Basics

When you first start Visual Studio Code with the PlatformIO extension you'll be greeted by the PlatformIO Home screen. If you don't see it look for a small icon on the bottom taskbar shaped like a "house" and click on it.

The Home screen displays the version of PlatformIO and also has a *Quick Access* section, which we will use to start our first project.

Before we get to that let's examine some of the other icons down the side of the PlatformIO screen.

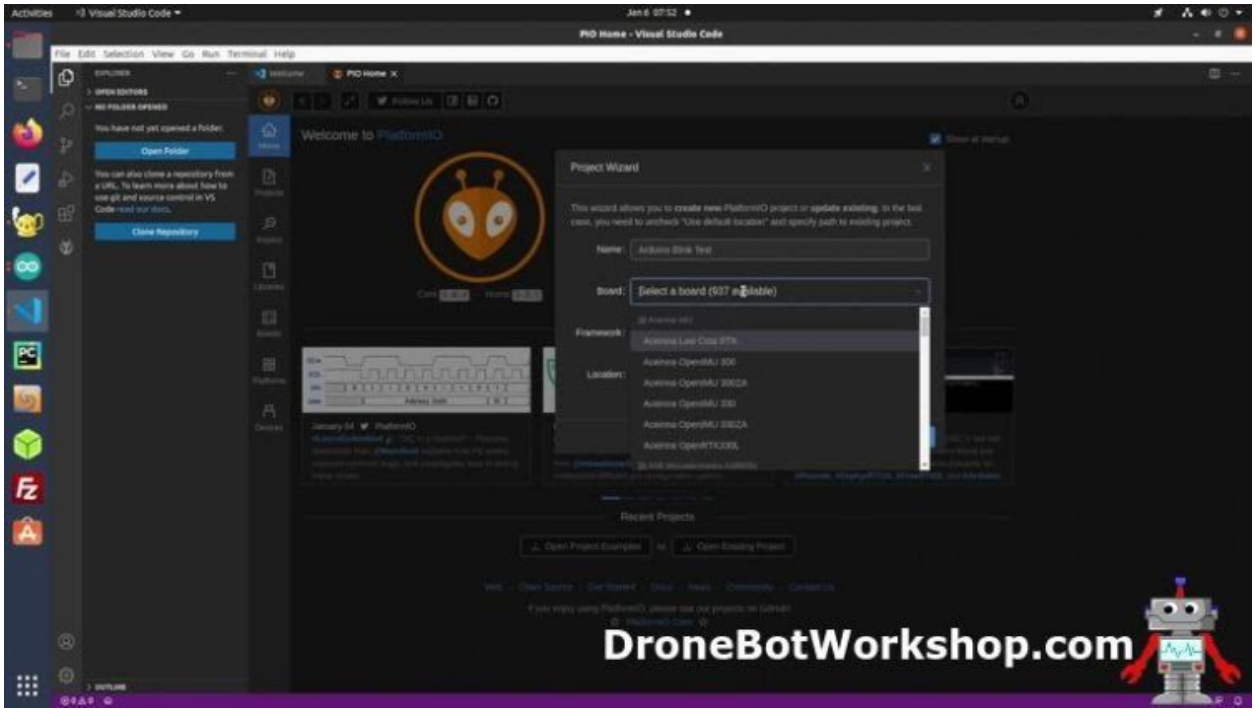
- **Home** – You have already seen this, it has the current version and the *Quick Access* box that allows you to create new projects.
- **Projects** – A list of all the projects you have created. You can edit these to add descriptions.
- Inspect – This allows you to inspect a project for statistics like memory utilization.
- **Libraries** – This is the Library Manager, which we will describe in detail later on in this article.
- **Boards** – A list of the boards supported by PlatformIO. As of this writing, there are over 900.
- **Platforms** – Platforms like the ArduinoAVR, Espressif ESP32, and others are listed here. The list will grow as you build projects with new boards.
- **Devices** – A list of the boards that are currently attached to your computer. This is built up automatically so you don't need to select the port, unlike the Arduino IDE.

Creating Your First Project

In PlatformIO your "sketches" are actually part of a "project", the term "sketches" is not used here.

All of the resources required for your project are contained in one place, this includes libraries and code files.

Go back to your PlatformIO Home screen and click the New Project button. This will launch the Project Wizard.



The Project Wizard makes it very easy to create all of the initial files required for a PlatformIO project. You will need to supply the following information:

- **Project Name** – Obviously the name of your project!
- **Board** – The type of microcontroller board you are using.
- **Framework** – The framework must match the board, and PlatformIO will determine this automatically for you once you select a board.
- **Location** – Where you want your files stored. You can just leave this checked to accept the default location or uncheck it to choose a specific one.

Selecting a board may appear to be quite daunting, as PlatformIO has over 900 of them! But you certainly don't need to scroll through the list manually, just start typing the bard name and the search will be narrowed down to one or more matching selections.

After you make the selections for your board PlatformIO will set up the files for your new project. If this is the first time you have used this type of board it will need to grab several files from the Internet, so you may have to wait a minute for it to finish. Subsequent selection of the same board will be much quicker, as it only has to do this once per board.

main.cpp File

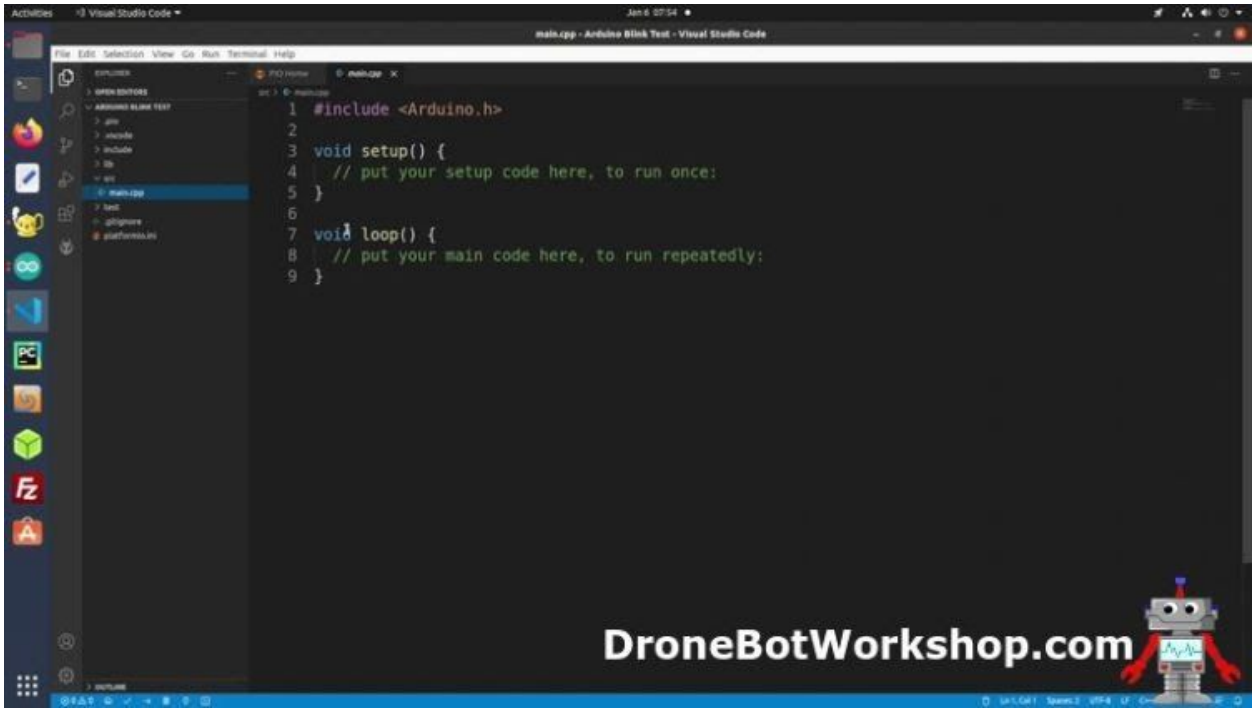
One of the first big differences between PlatformIO and the Arduino IDE is the type of files you'll be coding.

In the Arduino IDE, most of your files use the *.ino* extension. When you start a new project in the Arduino IDE you typically name the main file something like "mycode.ino". When you first save it the IDE will place it in a folder called "mycode".

In PlatformIO there are a number of files created for a project. The principal one is called *main.cpp*. The *.cpp* extension means "C++", the language you are coding in.

You will keep all of your code files in a subdirectory called *src*, which is an abbreviation for "source".

On the left side pane, you should see your project listed, and beneath it, you'll see a number of subdirectories. Expand the *src* subdirectory by clicking on it and you should see a *main.cpp* file. Click on this file to open it in the editor window.



You will notice that the main.cpp file already has a line in it that reads as follows:

```
1  #include "Arduino.h"
```

This line must be present in every program you write for microcontrollers using the Arduino framework, including non-Arduino boards like the ESP32. PlatformIO will automatically insert this line into the main.cpp file for you. If you are copying some existing Arduino code you can paste it underneath it.

Arduino Uno Blink Test

Let's start with the "Hello World" program for microcontrollers – the infamous Blink! Connect an Arduino Uno to one of your USB ports to follow along.

Here is Blink, modified for PlatformIO with the inclusion of the *Arduino.h* library:

```
#include <Arduino.h>

/*
  Simple Blink sketch
  simple-blink.cpp
  Use for PlatformIO demo

  From original Arduino Blink Sketch
  https://www.arduino.cc/en/Tutorial/Blink

  DroneBot Workshop 2021
  https://dronebotworkshop.com
*/

// Set LED_BUILTIN if undefined or not pin 13
// #define LED_BUILTIN 13

void setup()
{
  // Initialize LED pin as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop()
{
  // Set the LED HIGH
  digitalWrite(LED_BUILTIN, HIGH);

  // Wait for a second
  delay(1000);

```

```
32    // Set the LED LOW
33    digitalWrite(LED_BUILTIN, LOW);
34
35     // Wait for a second
36    delay(1000);
37  }
```

The code hardly needs any description, it's basically the Blink sketch you know and love (or at least know), using the constant LED_BUILTIN to represent the Arduino's onboard LED, which is connected to pin 13. Note that the definition of LED_BUILTIN is remarked out, as the Arduino Uno framework already knows it.

As with the Arduino IDE, you'll need to compile the code, and then upload it to your Arduino Uno.

Compiling the code is done using the PlatformIO Build button, which is a checkmark on the lower toolbar. Click on the checkmark and observe the progress in the terminal window. The code should compile successfully.

The next step is to upload it to the Arduino. Click on the key next to the Build key, the one shaped like an arrow. This is the Upload key. Once again I'll point out that we never had to tell PlatformIO which USB port we had our Arduino connected to, it figures it out by itself.

PlatformIO will upload the compiled code to the Arduino, and you should see the familiar flashing LED as a result.

ESP32 Blink

Now that we have seen how to compile and upload code using PlatformIO let's change our microcontroller board.

Remove the Arduino Uno and hook up an ESP32 board, pretty well any ESP32 board will do. After you have done that go back to the PlatformIO Home page and start a new project with the Quick Access *New Project* button.

Give your project a name and type “esp32” into the *Board* textbox. A list of ESP32 boards will be displayed. Scroll through the list until you find a board that matches yours.

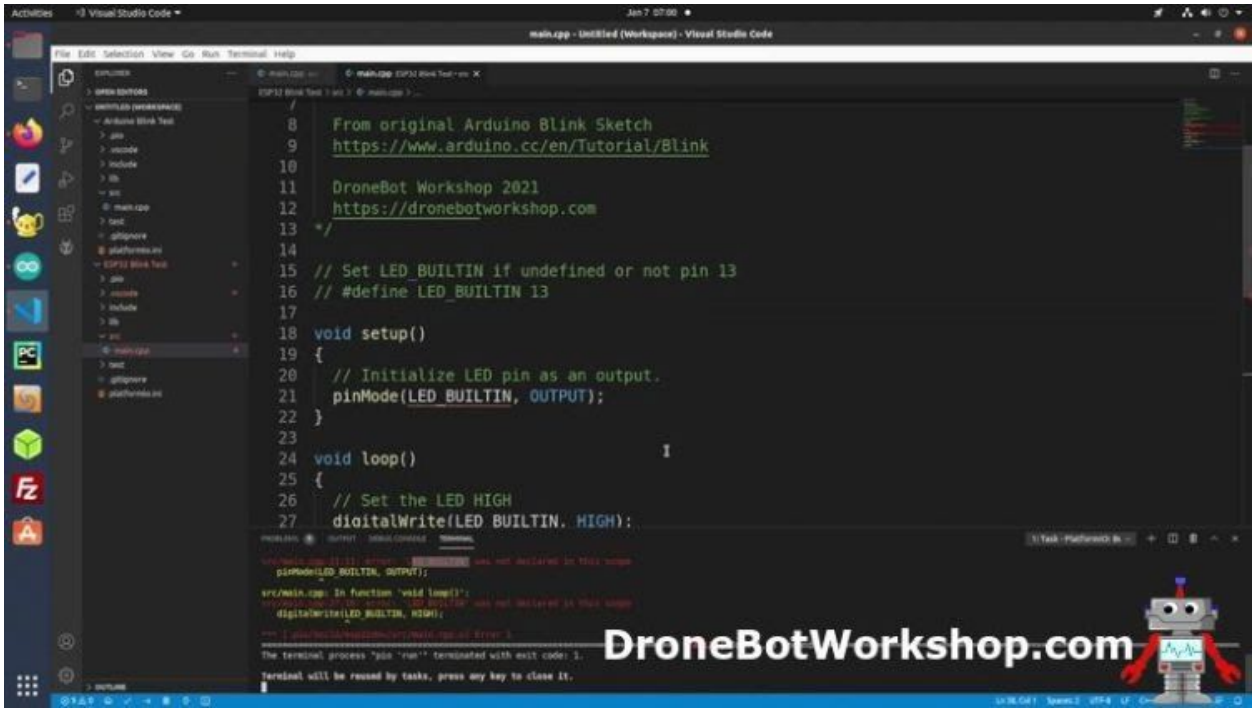
Note that the Framework will be filled in as “Arduino”, this is correct as despite our board using an ESP32 processor we are still using the Arduino framework to program it.

Once again you’ll need to wait while the required support files are downloaded and installed. When that task is completed you will see a new project on the left pane, right below the first project.

Expand the src subdirectory and open the main.cpp file for editing. It will look identical to the one we opened for the Arduino.

Now paste the same Blink code into the main.cpp file in your editor. After that hit the Build button (checkmark) to compile it.

This time you’ll get an error, and the code will fail to compile. And there is a good reason for that!



The error is produced because the constant LED_BUILTIN is not predefined for the ESP32. In addition, the ESP32 board uses a different pin for the built-in LED.

In order to fix this, you'll need to unremark the definition of LED_BUILTIN, as well as change its value from "13" to an appropriate pin number. On my ESP32 that is pin 2.

Once you make those changes you can Build and Upload the code, and your ESP32 will start flashing its approval!

Seeeduino XIAO with Serial Monitor

I repeated the Blink code, this time with a project for the Seeeduino XIAO. LED_BUILTIN is already a predefined constant on the XIAO, so it will run correctly.

I then modified the code, as just blinking was getting a bit boring, and I added statements to print the LED state to the Serial monitor.

You can follow the steps I took in the video accompanying this article, starting at the 21:09 mark. Pay attention to how PlatformIO “assisted” me when writing the code for the serial monitor. This is actually part of the Visual Studio Code Intellisense feature I mentioned earlier.

My final code looked like this:

```
#include <Arduino.h>

/*
  Simple Blink sketch
  simple-blink.cpp
  Use for PlatformIO demo

  From original Arduino Blink Sketch
  https://www.arduino.cc/en/Tutorial/Blink

  DroneBot Workshop 2021
  https://dronebotworkshop.com
*/

// Set LED_BUILTIN if undefined or not pin 13
// #define LED_BUILTIN 13

void setup()
{
  // Initialize LED pin as an output.
  pinMode(LED_BUILTIN, OUTPUT);
  Serial.begin(9600);
}

```

```
25  void loop()
26  {
27    // Set the LED HIGH
28    digitalWrite(LED_BUILTIN, HIGH);
29    Serial.println("HIGH");
30
31    // Wait for a second
32    delay(1000);
33
34    // Set the LED LOW
35    digitalWrite(LED_BUILTIN, LOW);
36    Serial.println("LOW");
37
38     // Wait for a second
39    delay(1000);
40  }
```

Compile and upload the code to your Seeeduino XIAO, and as you might expect the onboard LED will start blinking. But how do we see the Serial Monitor?

On the taskbar on the bottom of the IDE there is an icon that looks like a “plug”, this is the PlatformIO Serial Monitor. Click the icon and the Terminal area will switch to a serial monitor, and you’ll be able to observe the LED state here.

And if you’re using the Seeeduino XIAO keep in mind that a HIGH turns off the LED, which is backward from the Arduino Uno!

To close the Serial Monitor hit Ctrl-C (command-C on a Mac). Then hit your Enter key (or any other key, it really doesn’t matter). This will close the Serial Monitor and display the Terminal again.

Using Libraries with PlatformIO

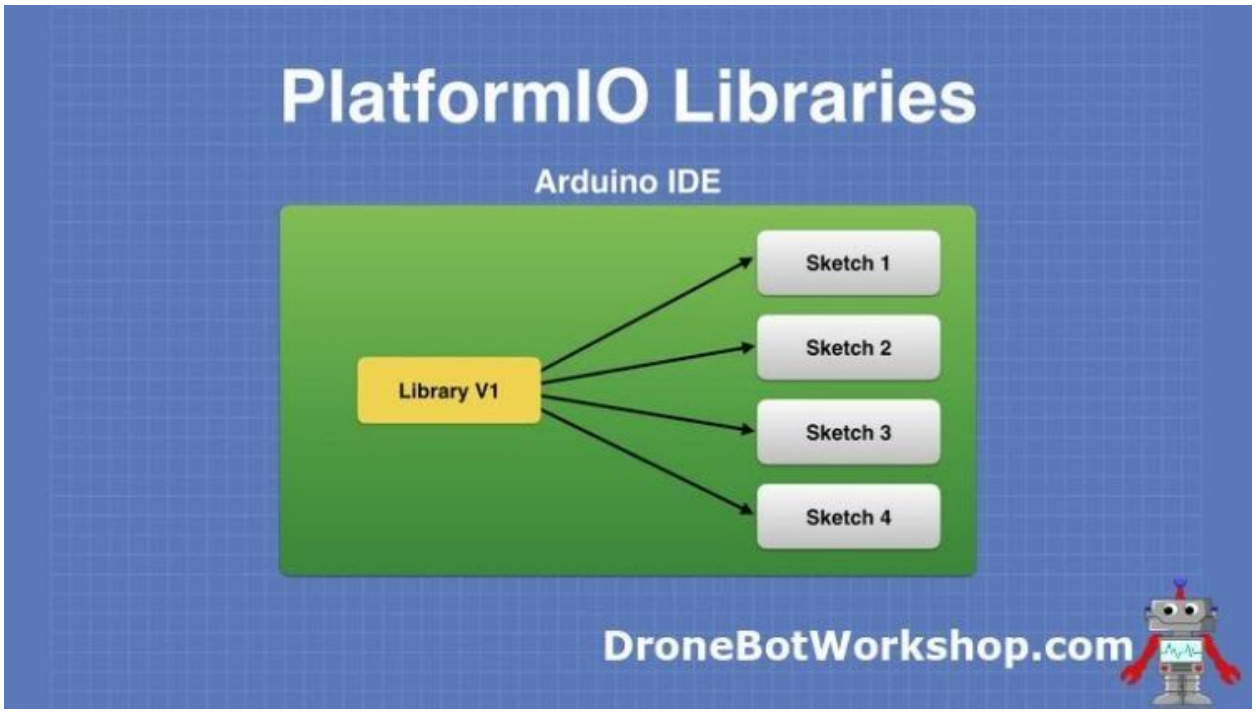
One task we need to know how to perform is to use libraries with PlatformIO.

In PlatformIO libraries work a bit differently than they do with the Arduino IDE, although you also have the option of using them in the same way.

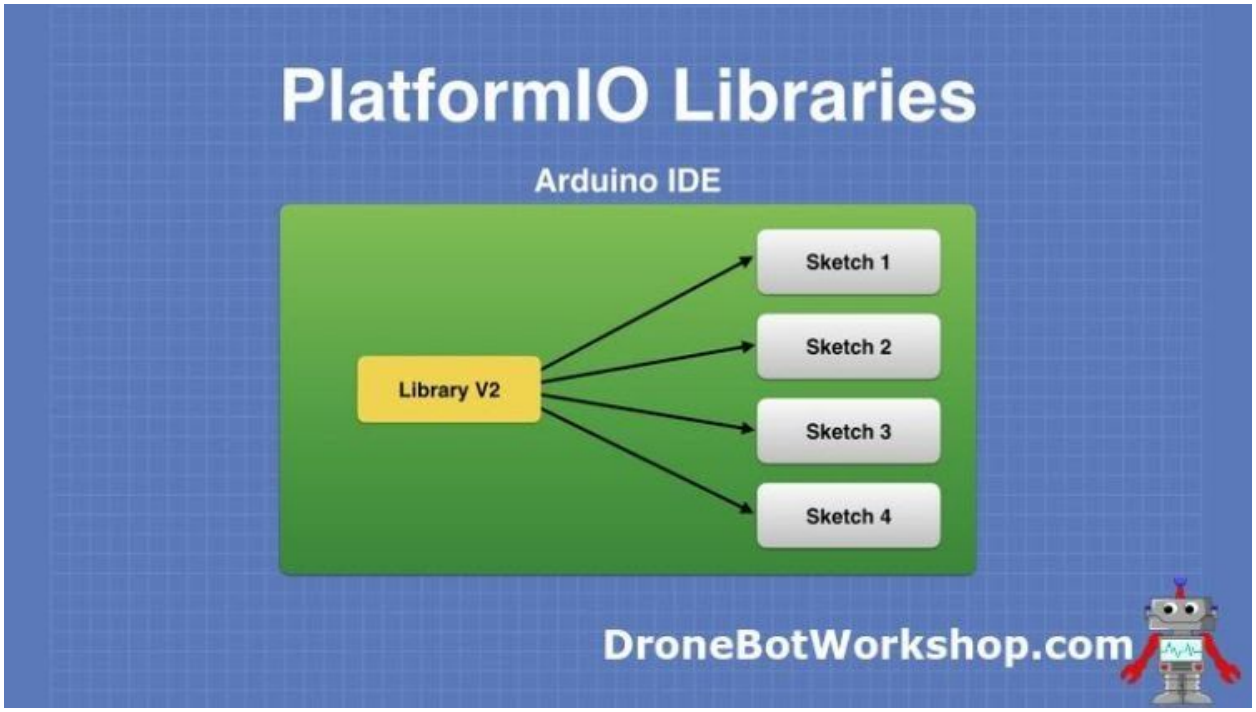
Library Management – Arduino IDE

In the Arduino IDE we have a Library Manager. This has access to thousands of Arduino libraries, and we can search for them and install them into our IDE.

Libraries installed in the Arduino IDE are available for every Arduino sketch. So, for example, if we install Version 1 of our library in the IDE and we have four sketches ALL of the sketches have access to that library.



If we upgrade that library to Version 2 then all of our sketches have Version 2.



While this may seem to be a good thing it sometimes can cause problems:

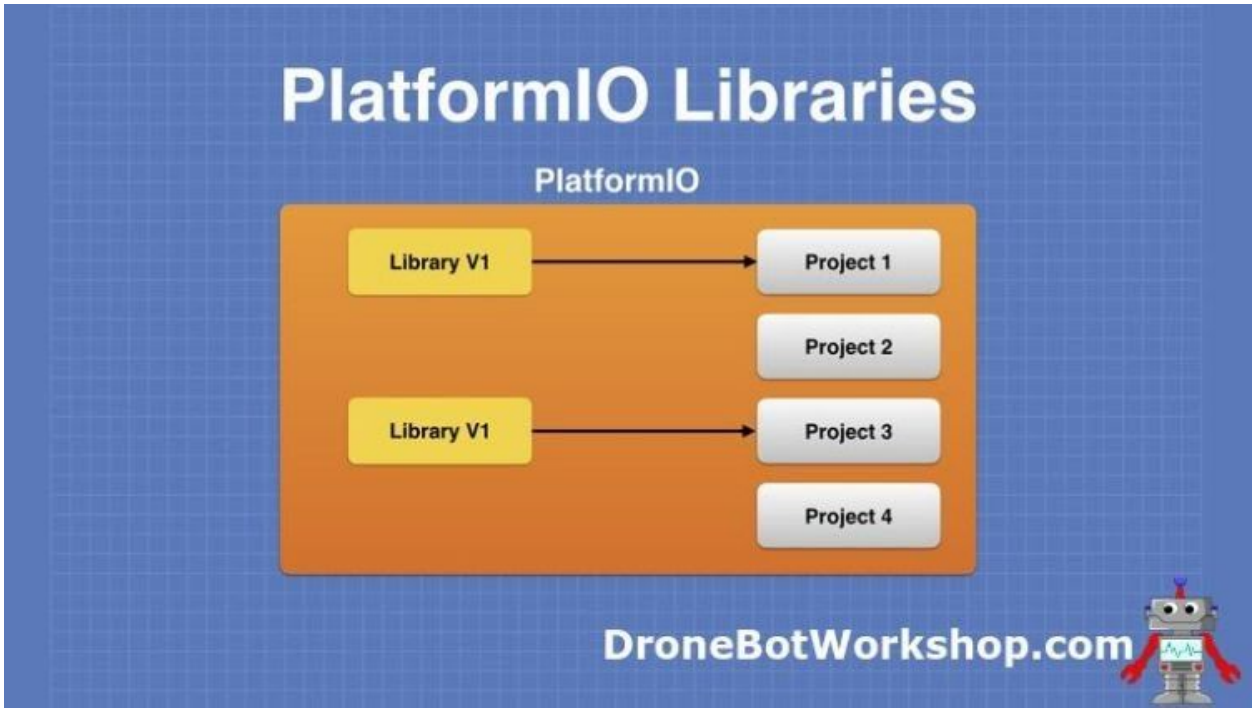
- If we copy our code to another computer that doesn't have the required library the code will fail.
- If we base our code on Version 1 it is possible that Version 2 will break it.
- If we find another library that has the same name as our first one we can't use it unless we remove the original one.

PlatformIO handles libraries in a different fashion.

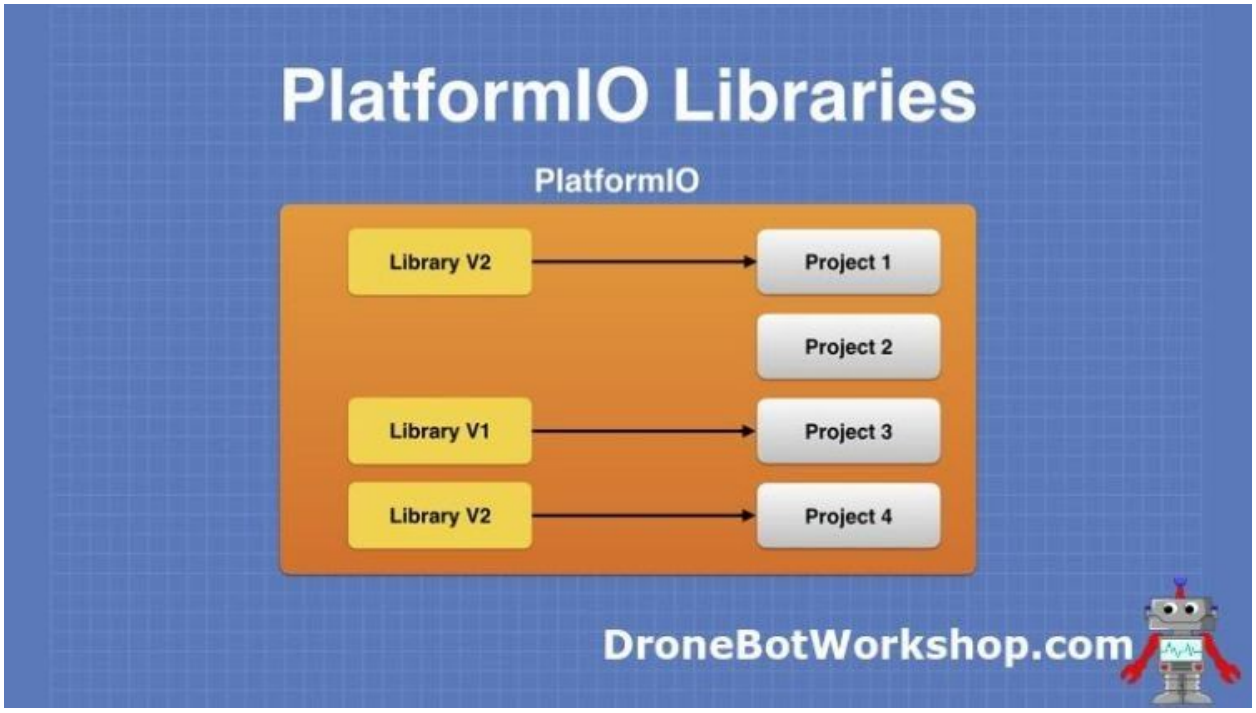
Library Management – PlatformIO

In PlatformIO libraries are managed on a per-project basis. You install your libraries into your project, not into the whole IDE.

So, to repeat our last example, we have PlatformIO with four projects.



In the above example, we have Version 1 of our library bound to both Project 1 and Project 3. The other two projects do not need this library.



Now we have updated Project 1 to use Version 2 of our library. We have also added Version 2 to Project 4. But Project 3 has not been updated, perhaps we are concerned that it may break due to changes in the new version. PlatformIO allows you to do this.

So the big difference in PlatformIO is that you add your libraries to the Project, not to the whole IDE.

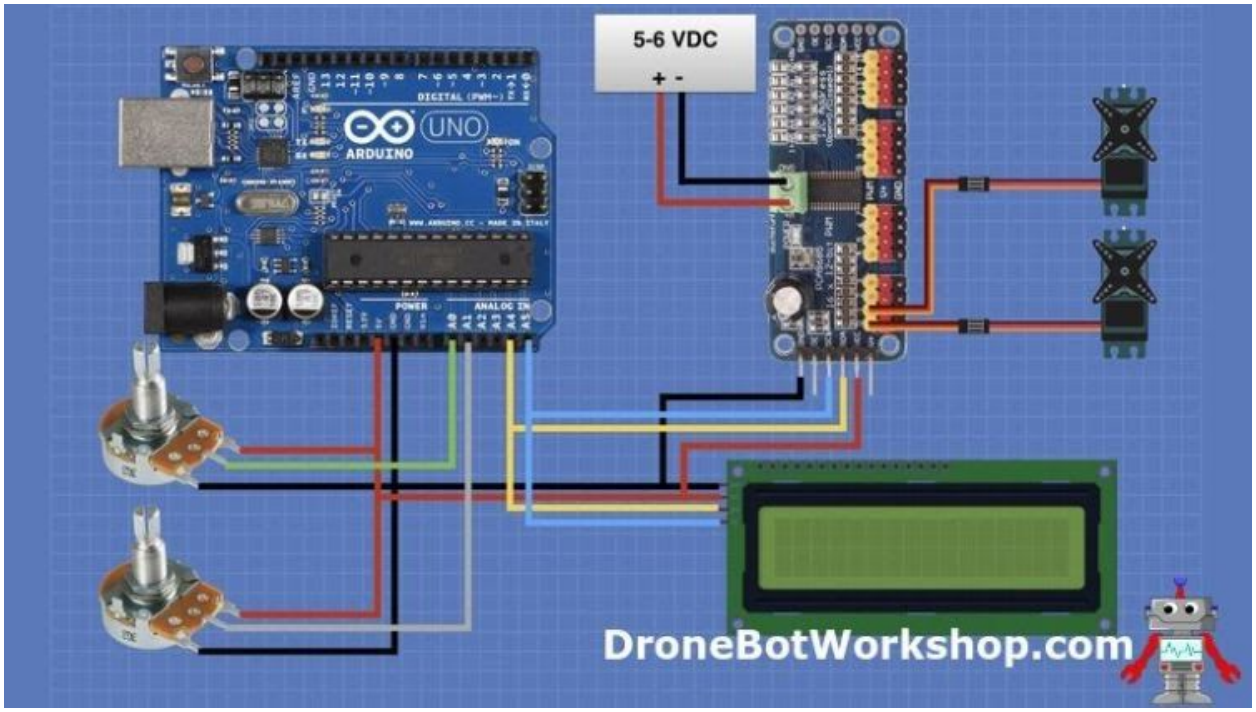
BTW, when you use the Arduino framework you already get all the built-in libraries that the Arduino has. And, like the Arduino IDE, they are available for all of your projects.

Dual Servo Library Demo

To illustrate this I'm going to put together a small project, one with an Arduino Uno, a couple of servo motors, a PCA9685 16-channel I2C servo driver, and a 2×16 LCD display with an I2C backpack.

The PCA9685 and the LCD display will require libraries, and we will also need the Arduino Wire library as well. So it's a good demo for using libraries, plus it's a cool little project.

You can hook everything up as shown here:



Once you have everything hooked up we will need some code.

We will start another new project using the Project Wizard and choose an Arduino Uno for our board. The code we will need to put into *main.cpp* is as follows:

```
/*
  Servo Motor Controller Demo for PlatformIO
  servo-control-demo.cpp
  Controls 2 servo motors, uses PCA9685 PWM Controller
  Displays status on 16x2 LCD
  Uses LiquidCrystal PCF8574 LCD Library
  Uses Adafruit PWM library
  Uses 2 potentiometers
  Uses I2C LCD Display

  DroneBot Workshop 2021
  https://dronebotworkshop.com
*/

// Include Arduino framework
#include <Arduino.h>

// Include PCF8574 Library for I2C LCD
#include <LiquidCrystal_PCF8574.h>

// Include Wire Library for I2C Communications
#include <Wire.h>

// Include Adafruit PWM Library
#include <Adafruit_PWMServoDriver.h>
// Define I2C Address - change if required
const int i2c_addr = 0x3F;

// Define LCD object
LiquidCrystal_PCF8574 lcd(i2c_addr);

```

```
// PWM Parameter Definitions
#define MIN_PULSE_WIDTH    650
#define MAX_PULSE_WIDTH    2350
#define FREQUENCY          50

// Define PWM object
Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();

// Define Potentiometer Inputs
int control0 = A0;
int control1 = A1;

// Define Motor Outputs on PCA9685 board
int motor0 = 0;
int motor1 = 1;

// Define Motor position variables
int mtrDegree0;
int mtrDegree1;

// Define Motor previous position variables
int mtrPrevDegree0 = 0;
int mtrPrevDegree1 = 0;

// Variable to determine if display needs updating
boolean updatedisplay = 0;


// Function to move motor to specific position
void moveMotorDeg(int moveDegree, int motorOut)
{
```

```
  int pulse_wide, pulse_width;

  // Convert to pulse width
  pulse_wide = map(moveDegree, 0, 180, MIN_PULSE_WIDTH, MAX_PULSE_WIDTH);
  pulse_width = int(float(pulse_wide) / 1000000 * FREQUENCY * 4096);

  //Control Motor
  pwm.setPWM(motorOut, 0, pulse_width);
}

// Function to convert potentiometer position into servo angle
int getDegree(int controlIn)
{
  int potVal,srvDegree;

  // Read values from potentiometer
  potVal = analogRead(controlIn);

  // Calculate angle in degrees
  srvDegree = map(potVal, 0, 1023, 0, 180);

  // Return angle in degrees
  return srvDegree;

}

void setup()
{
  // Setup PWM Controller object
  pwm.begin();
  pwm.setPWMFreq(FREQUENCY);
```

```
94
95    // Set display type as 16 char, 2 rows
96    lcd.begin(16,2);
97
98    //Turn on the LCD Backlight
99     lcd.setBacklight(255);
100
101    // Clear the display
102    lcd.clear();
103    lcd.home();
104
105    // Print on first row of LCD
106   lcd.setCursor(0,0);
107   lcd.print("Servo 0: 0");
108
109   // Print on second row of LCD
110   lcd.setCursor(0,1);
111   lcd.print("Servo 1: 0");
112
113   // Start Serial Monitor
114   Serial.begin(19200);
115
116 }
117
118 void loop() {
119   //Control Servo Motor 0
120
121   // Get desired position
122   mtrDegree0 = getDegree(control0);
123
124   // Move motor only if control position has changed
```

```
125   if (mtrDegree0 != mtrPrevDegree0) {
126     // Move motor
127     moveMotorDeg(mtrDegree0,motor0);
128     // Update motor moved variable
129     updatedisplay = 1;
130     // Update previous position
131     mtrPrevDegree0 = mtrDegree0;
132   }
133
134   //Control Servo Motor 1
135
136   // Get desired position
137   mtrDegree1 = getDegree(control1);
138
139   // Move motor only if control position has changed
140   if (mtrDegree1 != mtrPrevDegree1){
141     // Move motor
142     moveMotorDeg(mtrDegree1,motor1);
143     // Update motor moved variable
144     updatedisplay = 1;
145     // Update previous position
146     mtrPrevDegree1 = mtrDegree1;
147   }
148
149   // Update display if required
150   if (updatedisplay == 1) {
151
152     // Clear the display
153    lcd.clear();
154    lcd.home();
155
```

```
156     // Print on first row of LCD
157     lcd.setCursor(0,0);
158     lcd.print("Servo 0: ");
159     lcd.print(mtrDegree0);
160
161     // Print on second row of LCD
162     lcd.setCursor(0,1);
163     lcd.print("Servo 1: ");
164     lcd.print(mtrDegree1);
165
166   }
167
168   // Print to Serial Monitor
169   Serial.print("Motor 0: ");
170   Serial.print(mtrDegree0);
171   Serial.print("\t");
172   Serial.print("Motor 1: ");
173   Serial.println(mtrDegree1);
174   // Reset the motor moved variable
175   updatedisplay = 0;
176
177   // Add short delay
178   delay(100);
179
180 }
181
182
183
```

You can try to compile the code right now with the Build button, however, you won't be successful. The reason is probably pretty obvious, you haven't installed the libraries.

To do that you'll need to use the PlatformIO Library Manager.

Using The Library Manager

Open the PlatformIO Home screen, remember you can ghetto it with the little "house-shaped" icon on the bottom taskbar. You can also find it by clicking the PlatformIO icon on the left panel of VS Code and looking in *Quick Access* for *Home*.

The library manager is the fourth icon down, you can't miss it as it's labeled Libraries! Click on the icon to open it.

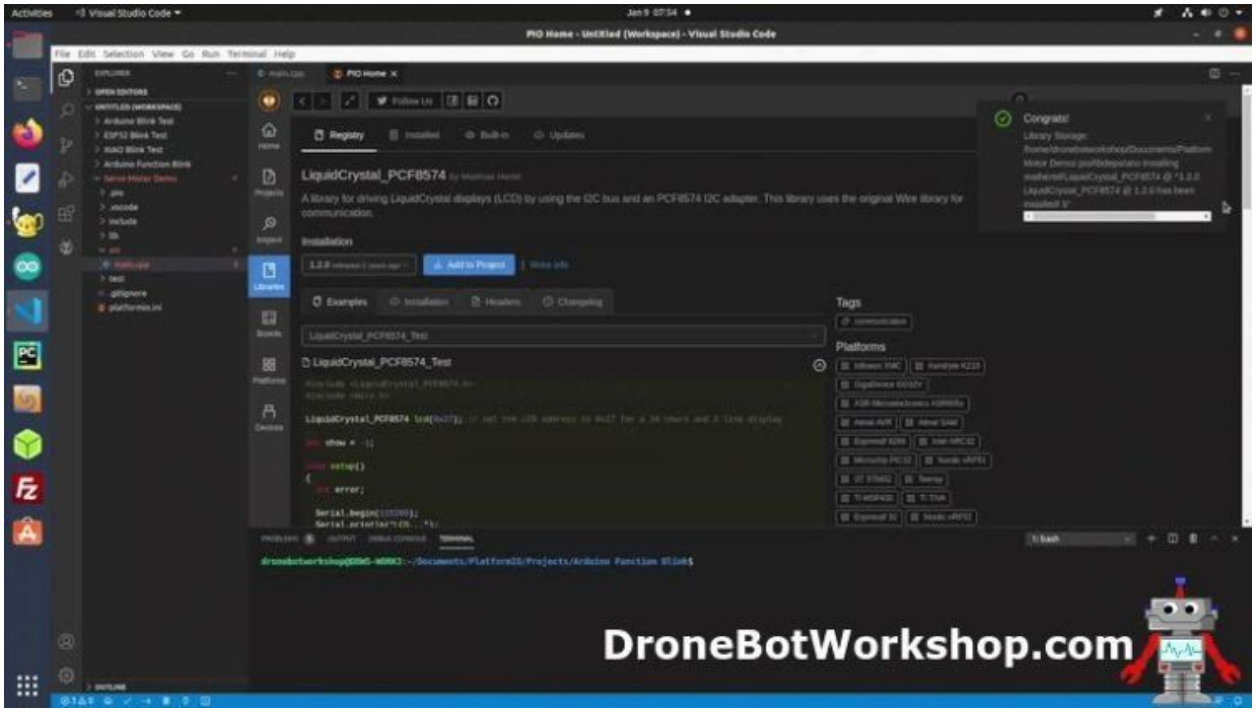
You'll see a search box where you can search for your required library. It's a pretty advanced search, you can input the library name, its file name, or the name of the component you need a library for.

Our servo project requires the following libraries:

- LiquidCrystal PCF8574 LCD Library
- Adafruit PWM library

Search for the first library. You'll see it displayed prominently among the results. Click on the desired library, and its details will be displayed. Note that you also get sample code and versioning information with the library.

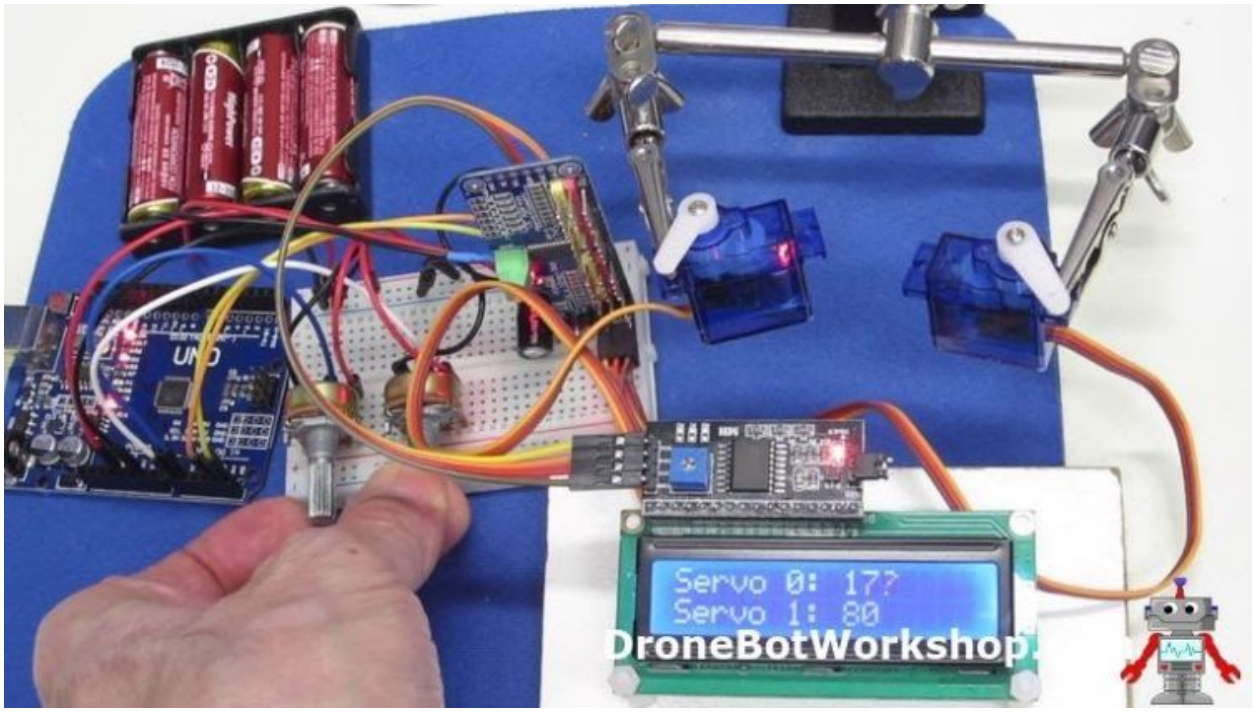
There is an Add to Project button that you can use to add the library to your project. Click on it and find the project you need to add it to, note that the project you are currently working on will be displayed first.



When the library is added a message will be displayed, congratulating you on accomplishing your task!

Click the Library Manager icon again to search for the second library, and repeat the same steps to add it to the project.

Now go back into the *main.cpp* for the project and try compiling it again. This time it will compile successfully, and you can upload it to the Arduino.

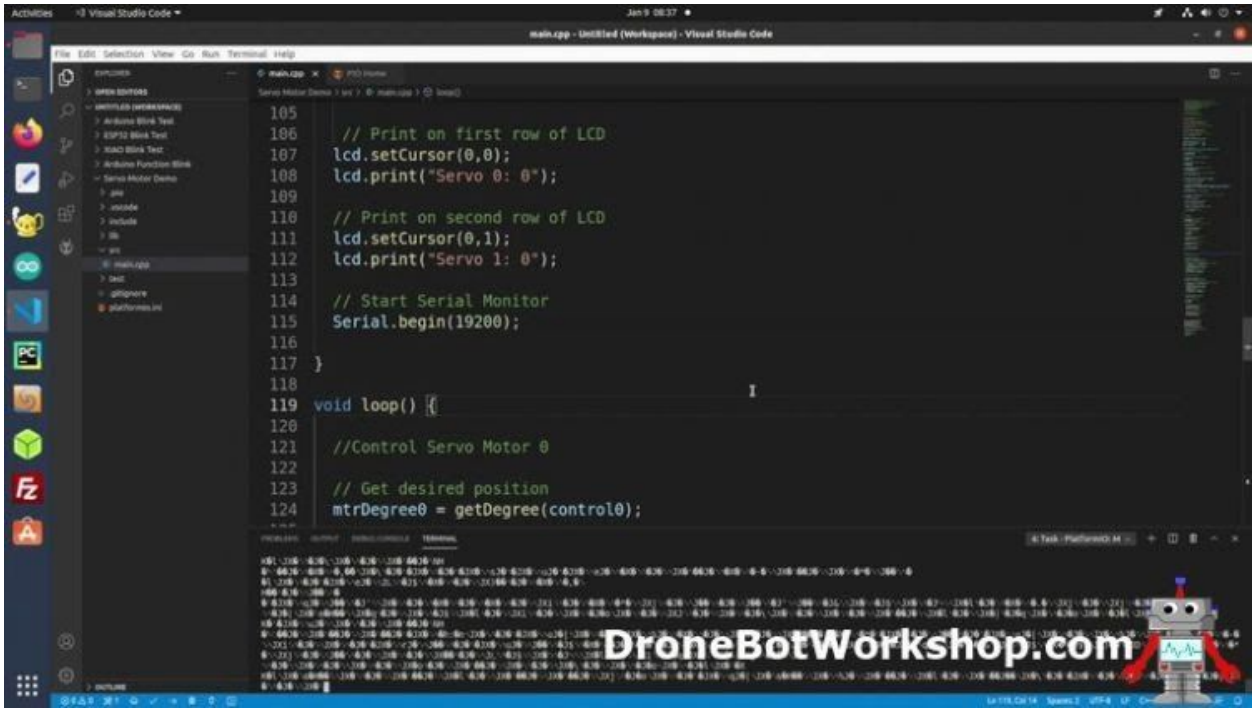


Try out the project, you should be able to move both servo motors and observe their position on the display.

platformio.ini File

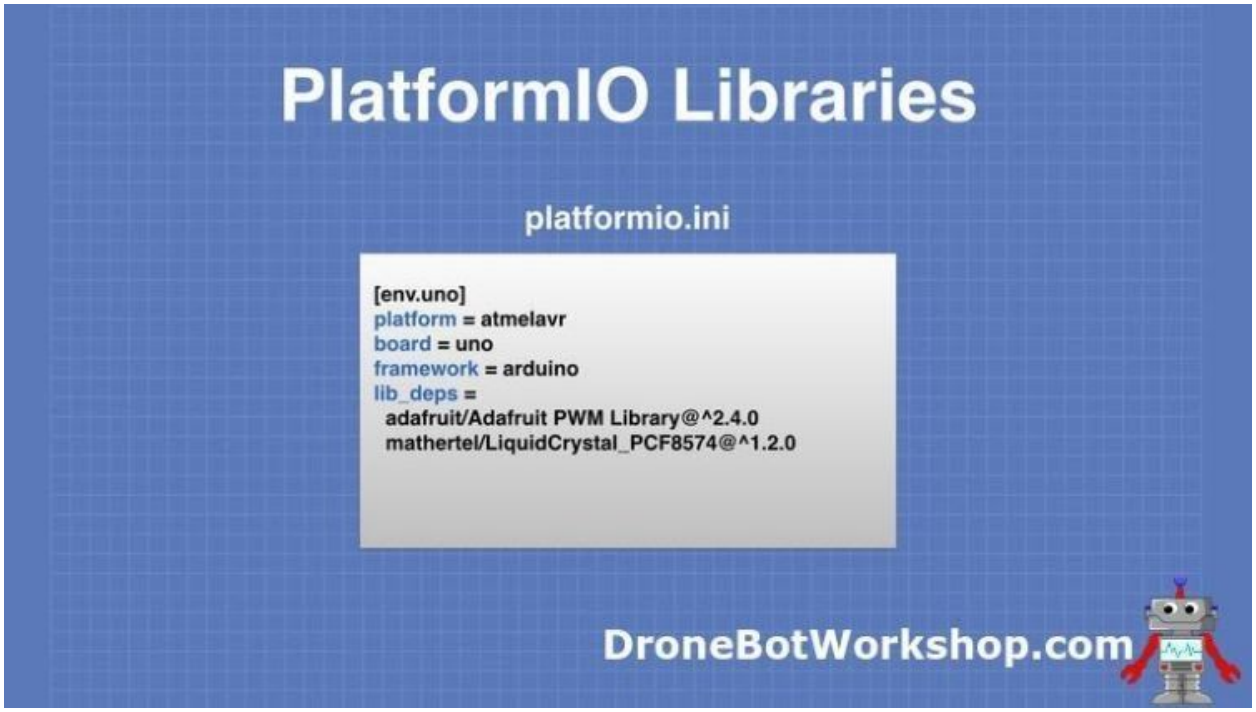
You will notice that the code also makes use of the Serial Monitor. Click the “plug” icon on the bottom taskbar to open the serial monitor.

You probably won’t like what you see in the Serial Monitor, it’s just a bunch of random characters. Go back into the code and you’ll see why we aren’t getting a proper display.



In this code, I set the serial output to 19,200 baud, instead of 9600. But since our Serial Monitor defaults to 9600 baud, it is not displaying the text correctly. We need to change the speed.

We do this by editing the *platformio.ini* file. You'll find this file at the bottom of every project, in the left pane.



The platformio.ini file contains the parameters for your project. It is a basic text file that you can edit.

You'll notice lines for the environment you are working in, the board and the framework.

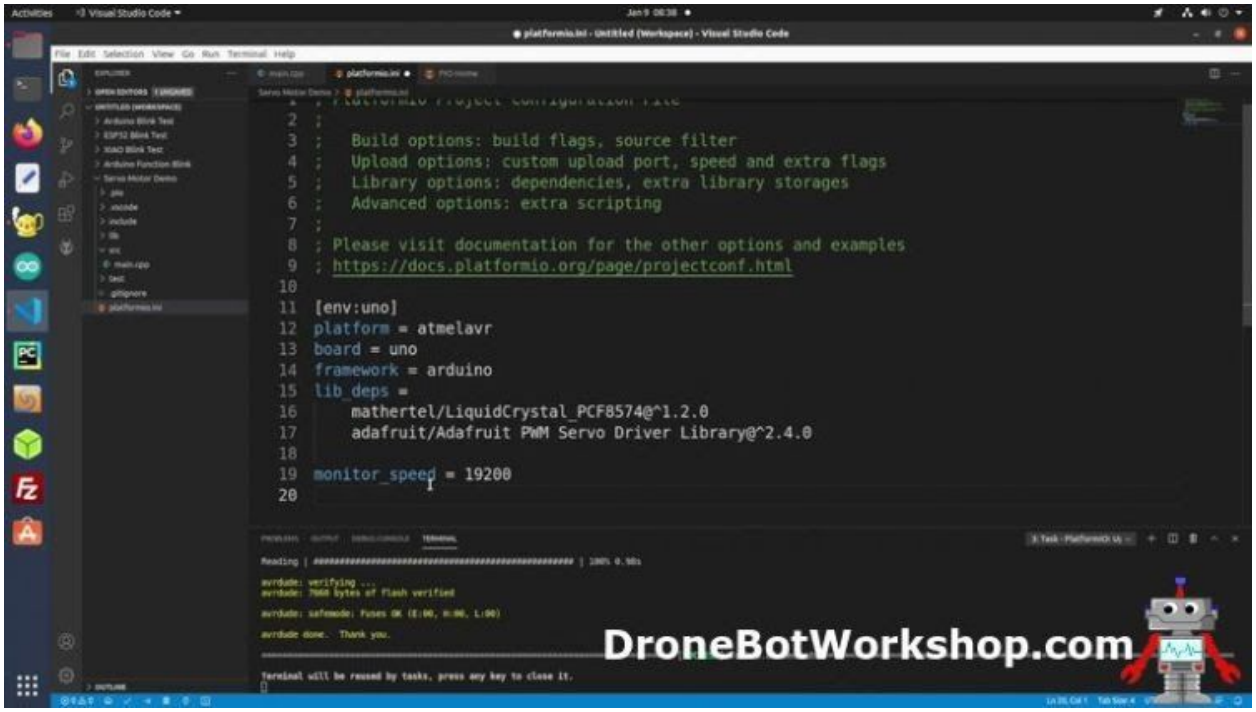
There are also lines for library dependencies, in fact you can just edit this file to install libraries and PlatformIO will install any missing ones automatically.

The platformio.ini file makes it possible to easily move your project to another computer.

We will need to edit our *platformio.ini* file to set the speed of the serial monitor. In the editor add the following line to the file:

```
1  Monitor_speed = 19200
```

Now save the file (Ctrl-C or command-C on a Mac) and go back into the Serial Monitor. You will notice that the text is now readable and the monitor displays the servo position.



You can examine the *platformio.ini* files for our other projects if you wish to get a better understanding of how it all works.

Conclusion

The learning curve for PlatformIO is a bit steeper than it is for the Arduino IDE, but the effort is well worth it. PlatformIO is a more advanced code editor that will help you write better code for a multitude of microcontrollers.

I'll be using PlatformIO for many of my upcoming projects, and we will also revisit PlatformIO in the near future to learn more about the advanced features of this wonderful develop[ment environment.

Happy coding!

Resources

Code for this article – All of the code used here and on the video in a ZIP file.

PlatformIO – The home page for PlatformIO.

PlatformIO Documentation – Complete documentation for PlatformIO.

Visual Studio Code – Download Visual Studio Code from Microsoft.

Visual Studio Code Documentation – Getting started with VS Code.