Getting started with PlatformIO

Initial draft
V 0.0.1

Andy Powell – 2023-08-01

Setting Up

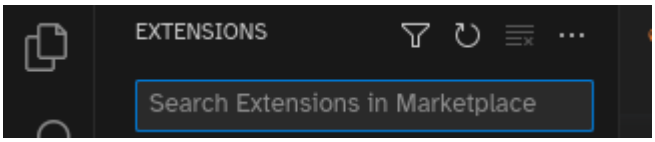
PlatformIO under linux needs pyhton3-venv to be installed. OSX doesn't have this issue and just works. I have no experience of the Windows version since I don't have Windows.

```
$ sudo apt-get install python3-venv
```

This is just a basic install of an extension in VSCode, if you know what you're doing you can just go ahead and install the PlatformIO extension then skip to the next section.

Install VSCode, then select the extensions icon from the left hand toolbar.

In the search box at the top of the screen type Platformio and hit enter. Select the Platformio extension to install it. VSCode might need to restart.



Once installed you should see a little alien head (it might be an ant, I'm unsure) in the left toolbar, this is PlatformIO. Click it to start PlatformIO.

There are also some icons along the bottom status bar. The little house icon is the PlatformIO home page – you can use it to jump there at any time. Until you're used to using PlatformIO this is a good place to start for your projects. It should automatically display when you start VSCode unless you remove the tick from the 'Show at startup' checkbox (top right or the home screen page).

The tick in the status bar is 'compile' and the arrow to the right is send to device which will also compile if required. The dustbin/trashcan is to clean builds.

First Project

Setting up a project is quick and simple, you'll just need a project name, know the board you are using and the path where you want to store it.

From the PlatformIO home page (click the little house if the tab isn't open) and on the right, you'll see a "Quick Access" menu.

Select New Project and you'll be presented with the Project Wizard, asking for a **Project name** and board type. Give your project a name, eg My Test Project then tab to the **Board** selection.

This is a dropdown with a list of over 1400 items. Luckily you don't have to scroll down the entire list to find your board, you can start typing its name, eg esp32 dev which should narrow the list down. For ESP32 Development I tend to just opt for the Espressif ESP32 Dev Module – even though my boards are custom.

Below the board is a dropdown that allows you to select which **framework** you want to use. This list will show all of the available frameworks for the board you have selected. In the case of the ESP32 it's going to be Arduino or ESPIDF but other boards will list other frameworks. Everything will be downloaded as and when needed so you don't have to setup or preinstall the framework yourself.

Once you've completed filling out the wizard click the 'finish' button and wait for PlatformIO to complete setting up your project. If this is the first time you've created a project with the framework selected it will take longer to complete the initial setup of your project as the framework will be downloaded and installed. Subsequent setups with the framework will be much quicker.

Once complete you should see a folder/file list down the left hand side of the VSCode window. The ones we're primarily interested in at this point are **include**, **lib** and **src** and the configuration file **platformio.ini**.

Normally libraries are downloaded, cached and added to your project when you configure your platformio.ini file using the *lib_deps* flag. The lib folder is where you put libraries that you have modified or have built yourself. You will still need to specify these in your platformio.ini file.

The src/include folders should be self explanatory. For includes in your code, you don't need to worry about the path unless it's a sub-folder. I find it odd that it's not actually in the src folder but there you go. The src folder will have a stub/example file in it.

platformio.ini

This is where things get interesting. Your basic platformio.ini file will be something like

```
[env:esp32dev]
platform = espressif32
framework = arduino
board = esp32dev
```

The env sections are essentially a *build-for-this* definition. The platformio.ini file can contain multiple env sections and unless you specify one to build, they'll all be built each time you compile. So for example if you are using the Arduino framework and working on a Teensy and an ESP32 version you can have an env section for each and have them built at the same time sharing the same source, but building with different parameters.

We use a [platformio] section to specify which env sections are built. Leaving the default_envs empty, or commenting it out with a semi-colon (;) will build them all.

```
[platformio]
description=My test program on my test board
default_envs= esp32dev
```

If you want to build parameter sections, sections that will be used by the env sections, you omit the env: For example:

```
[wifi]
ssid="mySSID"
password="myPassword"
```

defines the [wifi] section with 2 variables. To reference those variables we use

```
${sectionname.variable}
```

so using the [wifi] section ssid above, that'd be

```
${wifi.ssid}
```

We can use these parameters in any section to build our code for a device.

These are a number of keywords we can use to build the firmware for our device. The most important being *build_flags*, *lib_ignore* and *lib_deps*.

build_flags : Used to pass flags to the compiler for example -D for adding defines

lib_ignore : Used if you're building for multiple devices that vary where you don't want libraries that aren't required – or that wouldn't work – to compile.

lib_deps : These are libraries that your code requires in order to compile and function.

Some examples of each are:

```
build_flags=   -DMY_LED=14
               '-DWIFI_NETWORK_SSID=${wifi.ssid}'
               '-DWIFI_NETWORK_PASSWORD=${wifi.password}'
               -DDEBUG
```

This will pass 4 defines which can be handled in your code. *Note the quoted defines for passing string for the ssid and password.*

lib_ignore and lib_deps both work in the same way, the only difference being one is a list of ignored libraries and the other is a list of required libraries.

```
lib_deps       =    adafruit/Adafruit Unified Sensor @ ^1.1.11
lib_ignore     =    SHT
                    knolleary/PubSubClient@^2.8
```

Two things to note here

- The formatting of the library names
- Multiple library names in a list.

Lets deal with multiple libraries in a list. These are formatted, one per line no separator other than white space (i.e. no comma separation). Tabs and spaces are fine if you like tidy build files.

The formatting of library names depends on whether the library is being read from the libs folder or the downloaded cache/fetched library.

For libraries in the libs folder use the defined name field in the library.properties file for the library concerned – literally the bit after the = in the name= line.

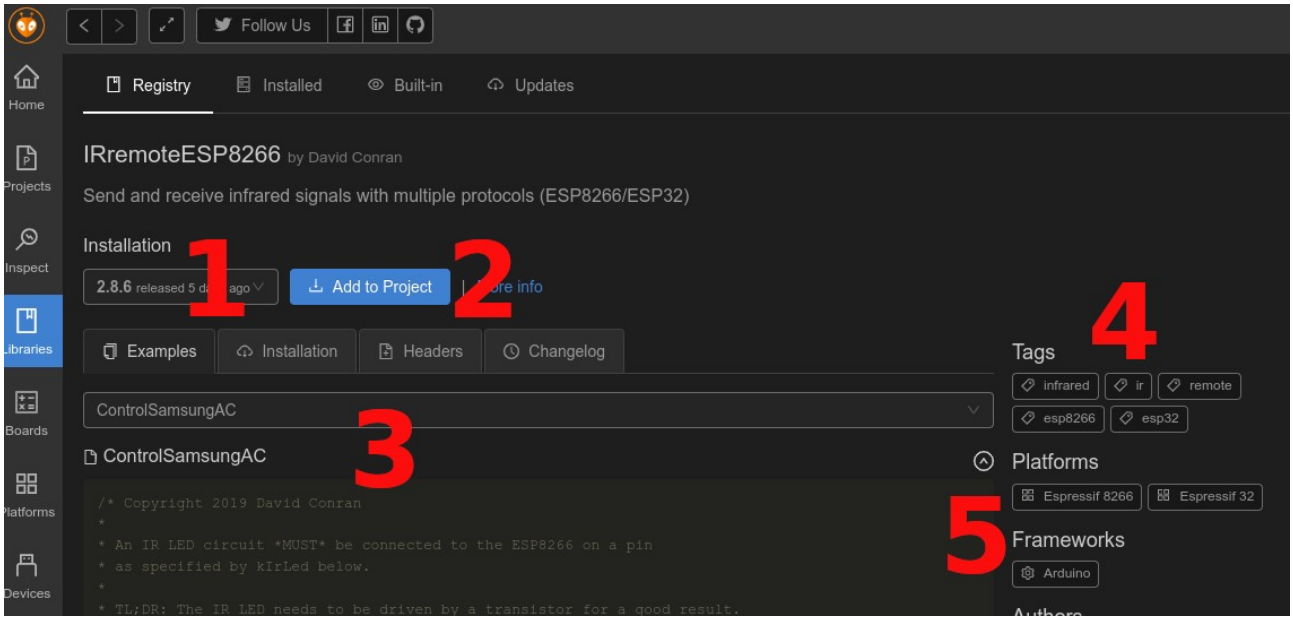
Fetched libraries are cached, you can see in the lib_deps example that we've requested version 1.1.11 of Adafruit's Unified sensor library. The **^** indicates you will accept backwards compatible versions with patches, using a **~** means you'll accept any <major>.<minor> version provided the patch version is greater than or equal to the one you specified. Finally omitting both the **~** and **^** means you will only accept the version you specified.

So now you're probably thinking "Well how am I supposed to know what a library is called?" and you'd be right. Platformio helps us here, and helps us quite a lot.

Caution: always save your platformio.ini changes before using the library add feature of PlatformIO.

Click the Home icon on the status bar to get to the PlatformIO home page, or switch to the tab if it's already open. Down the left side there's a series of icons Home, Projects, Inspect. Click the Libraries one.

You should see a search box at the top, type in **esp32 irremote** and you should see a number of libraries appear, just click on the name of the first one that appears – it doesn't really matter what's there, we're just looking at the interface right now.



1) A dropdown with the available versions of the library. Select the version of the library you want to use.

2) A quick 'Add to Project' button that will insert the appropriate lines in your platformio.ini file. **Caution: always save your platformio.ini changes before using the library add feature of PlatformIO. Note that this will add the library to *ALL* of your env sections.**

3) The four tabs show examples of usage, how to manually add the library, a list of all the header files and a changelog for the repo.

4) Some tags to help find alternative libraries that have the same or similar functionality.

5) A list of the supported platforms and frameworks for this library.

Loading firmware

By default PlatformIO will try and find your attached boards and program them. If, like me, you tend to use OTA Updates then adding the following to your env should trigger an OTA update when you try and push your code to the device.

```
upload_protocol = espota
upload_port = 10.120.3.12
```

I don't really do it like that because that'd mean remembering all those device IP addresses and well, I don't really have time for that. So again, I have a section setup similar to this:

```
[otadevices]
office = 10.120.3.12
workshop = 10.120.3.44
kitchen= 10.120.3.46
hall = 10.120.3.40
```

and then I just change the port to match the device I'm updating/modifying.

```
upload_protocol = espota
upload_port = ${otadevices.office}
```

If your device is attached and you want to output from its serial port you can pre-configure the PlatformIO serial port to be the speed you've set on your micro controller using:

```
monitor_port=115200
```

in the env section of the board it applies to.

So that's a basic guide to getting you up and running. There's far more to PlatformIO than covered here but hopefully this will get you started enough for it not to be a hindrance to building your projects. On the next few pages I've included some device specific notes and one of the platform.ini files I started out using which shows many of the elements previously explained in a single file.

Platform Notes

Teensy

Useful build_flags for the Teensy boards. Note you can only have one of these flags defined at a time:

```
build_flag               | Description (as in Arduino IDE)
-------------------------|---------------------------
USB_SERIAL               | Serial
USB_DUAL_SERIAL          | Dual Serial
USB_TRIPLE_SERIAL        | Triple Serial
USB_KEYBOARDONLY         | Keyboard
USB_TOUCHSCREEN          | Keyboard + Touch Screen
USB_HID_TOUCHSCREEN      | Keyboard + Mouse + Touch Screen
USB_HID                  | Keyboard + Mouse + Joystick
USB_SERIAL_HID           | Serial + Keyboard + Mouse + Joystick
USB_MIDI                 | MIDI
USB_MIDI4                | MIDIx4
USB_MIDI16               | MIDIx16
USB_MIDI_SERIAL          | Serial + MIDI
USB_MIDI4_SERIAL         | Serial + MIDIx4
USB_MIDI16_SERIAL        | Serial + MIDIx16
USB_AUDIO                | Audio
USB_MIDI_AUDIO_SERIAL    | Serial + MIDI + Audio
USB_MIDI16_AUDIO_SERIAL  | Serial + MIDIx16 + Audio
USB_MTPDISK              | MTP Disk (Experimental)
USB_RAWHID               | Raw HID
USB_FLIGHTSIM            | Flight Sim Controls
USB_FLIGHTSIM_JOYSTICK   | Flight Sim Controls + Joystick
USB_EVERYTHING           | Everything (only teensy 3.1 3.5 3.6)
USB_DISABLED             | No USB
```

Example File

```
[platformio]
description = Low Power ESP32 Board for reading SHT31 or
DHT11/22 temperature and humidity sensors

[otadevices]
office = 10.120.3.12
workshop = 10.120.3.44
kitchen= 10.120.3.46
hall = 10.120.3.40

[wifi]
ssid = "mySSID"
password = "myPassword"

[common]
build_flags =
     -DBUILTIN_LED=2
     -DNOTIFY_LED=16
     -DVOLTAGEMONITOR=33
     -DA1=4
     -DD1=13
     -DD2=14
     -DD3=15
     -DSTAY_AWAKE=15
     '-DWIFI_NETWORK_SSID=${wifi.ssid}'
     '-DWIFI_NETWORK_PASSWORD=${wifi.password}'

debug_flags =
     -DDEBUG
     -DSERIALLIVE
     -DUSETELNET

debug_libs = TLog

production_ignore_libs = TLog

dht_libs = knolleary/PubSubClient@^2.8
     adafruit/Adafruit Unified Sensor @ ^1.1.11

dht_ignore_libs = SHT31

sht_libs = knolleary/PubSubClient@^2.8

sht_ignore_libs = DHT sensor library
```

```
[env:lowpower-sht]
platform = espressif32
board = esp32dev
framework = arduino

upload_protocol = espota
upload_port = ${otadevices.office}

build_flags =
     ${common.build_flags}
     -DPIN_SCL=14
     -DPIN_SDA=13
     -DUSESHT
     '-DOTA_HOST_TYPE="OTA-LPSHTEMP"'

lib_ignore = ${common.sht_ignore_libs}
     ${common.production_ignore_libs}

lib_deps =
     ${common.sht_libs}


[env:lowpower-sht-DEBUG]

platform = espressif32
board = esp32dev
framework = arduino

monitor_speed = 115200

build_flags =
     ${common.build_flags}
     ${common.debug_flags}
     -DPIN_SCL=14
     -DPIN_SDA=13
     -DUSESHT
     '-DOTA_HOST_TYPE="OTA-LPSHTEMP"'
lib_ignore = ${common.sht_ignore_libs}

lib_deps =
     ${common.sht_libs}
     ${common.debug_libs}
```

```
[env:lowpower-dht]
platform = espressif32
board = esp32dev
framework = arduino

upload_protocol = espota
upload_port = ${otadevices.workshop}

build_flags =
     ${common.build_flags}
     -DDHT_PIN=14
     -DDHT_TYPE=DHT11
     -DUSEDHT
     '-DOTA_HOST_TYPE="OTA-LPTEMP"'

lib_ignore = ${common.dht_ignore_libs}
     ${common.production_ignore_libs}

lib_deps =
     ${common.dht_libs}


[env:lowpower-dht-DEBUG]
platform = espressif32
board = esp32dev
framework = arduino

monitor_speed = 115200

build_flags =
     ${common.build_flags}
     ${common.debug_flags}
     -DDHT_PIN=14
     -DDHT_TYPE=DHT11
     -DUSEDHT
     '-DOTA_HOST_TYPE="OTA-LPTEMP"'

lib_ignore = ${common.dht_ignore_libs}

lib_deps =
     ${common.dht_libs}
     ${common.debug_libs}
```