Documentation for PyVIEW

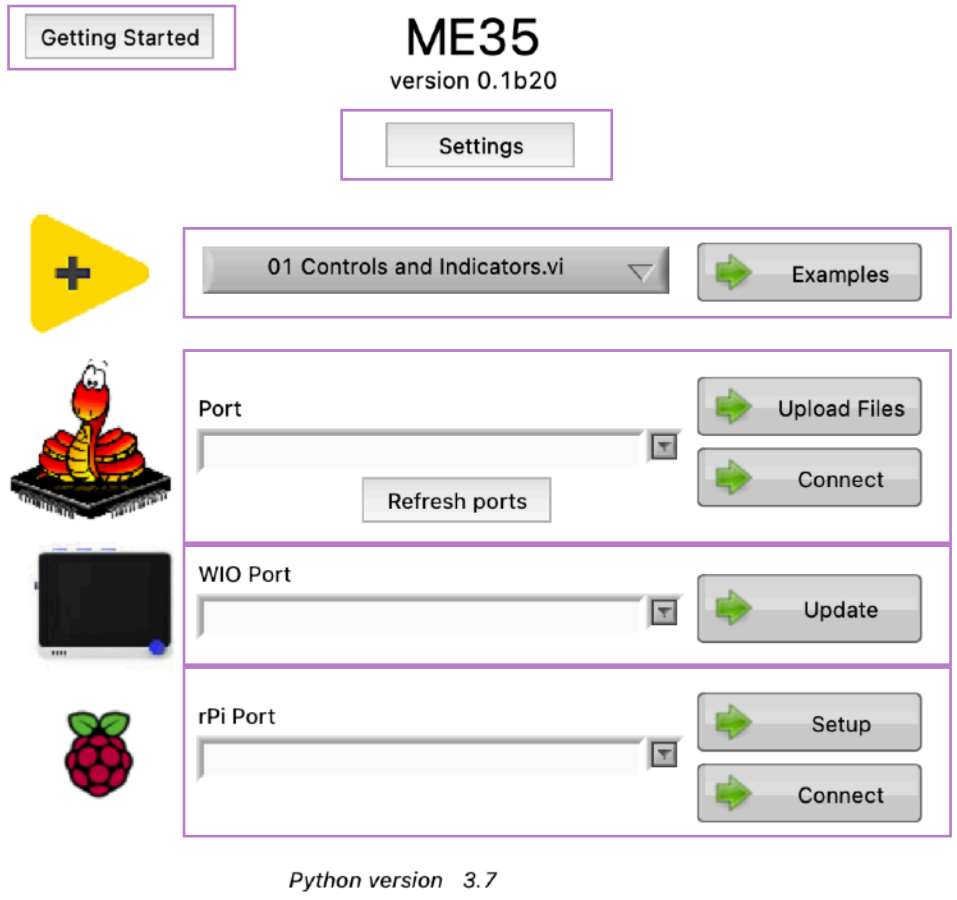
7/14/21

This software that lets you talk over serial to a lot of different microprocessors that I wrote for my class. Below is the overview in this document. Click on any line to get more info. The two links in the footer will always bring you back to either the entry splash screen or the main coding screen overviews.

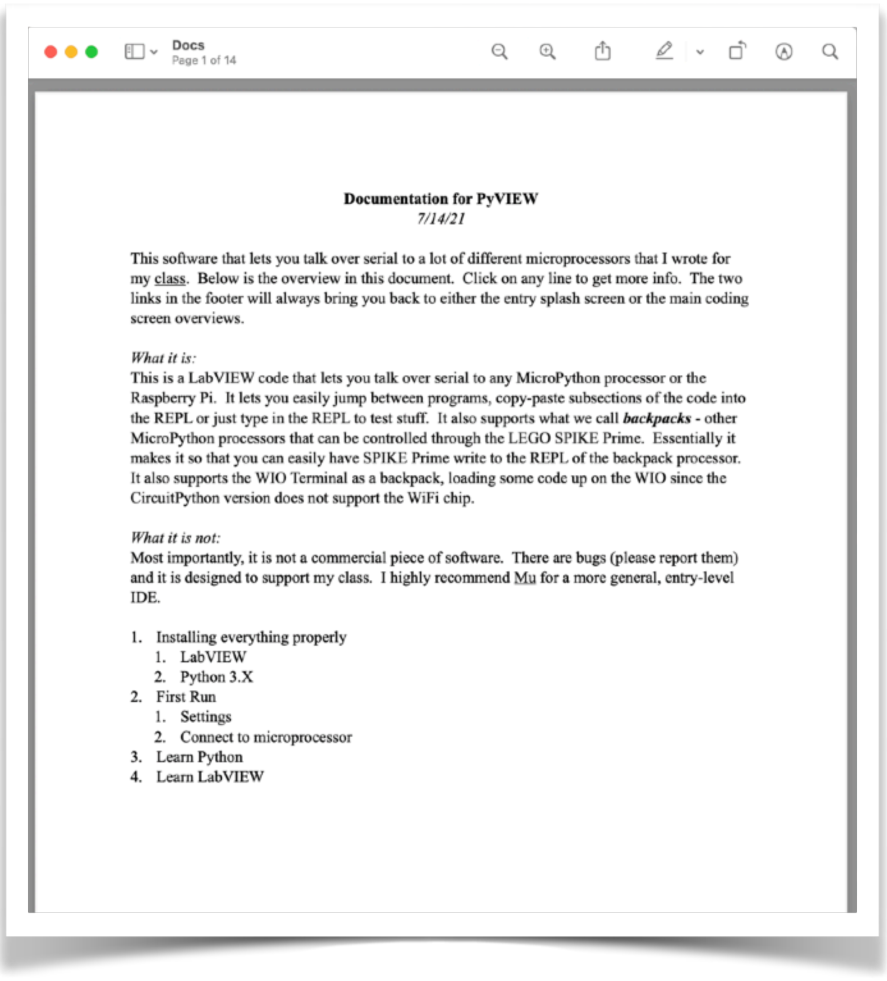
What it is:
This is a LabVIEW code that lets you talk over serial to any MicroPython processor or the Raspberry Pi. It lets you easily jump between programs, copy-paste subsections of the code into the REPL or just type in the REPL to test stuff. It also supports what we call ***backpacks*** - other MicroPython processors that can be controlled through the LEGO SPIKE Prime. Essentially it makes it so that you can easily have SPIKE Prime write to the REPL of the backpack processor. It also supports the WIO Terminal as a backpack, loading some code up on the WIO since the CircuitPython version does not support the WiFi chip.

What it is not:
Most importantly, it is not a commercial piece of software. There are bugs (please report them) and it is designed to support my class. I highly recommend Mu for a more general, entry-level IDE.

Splash Screen
The main screen allows you to set stuff up or connect to a processor - click on any purple box for more information (and then return here by clicking on the splash link in the footer).



Getting Started
This button opens up this PDF of documentation (found in the Docs folder).



Documentation for PyVIEW
7/14/21

This software that lets you talk over serial to a lot of different microprocessors that I wrote for my class. Below is the overview in this document. Click on any line to get more info. The two links in the footer will always bring you back to either the entry splash screen or the main coding screen overviews.

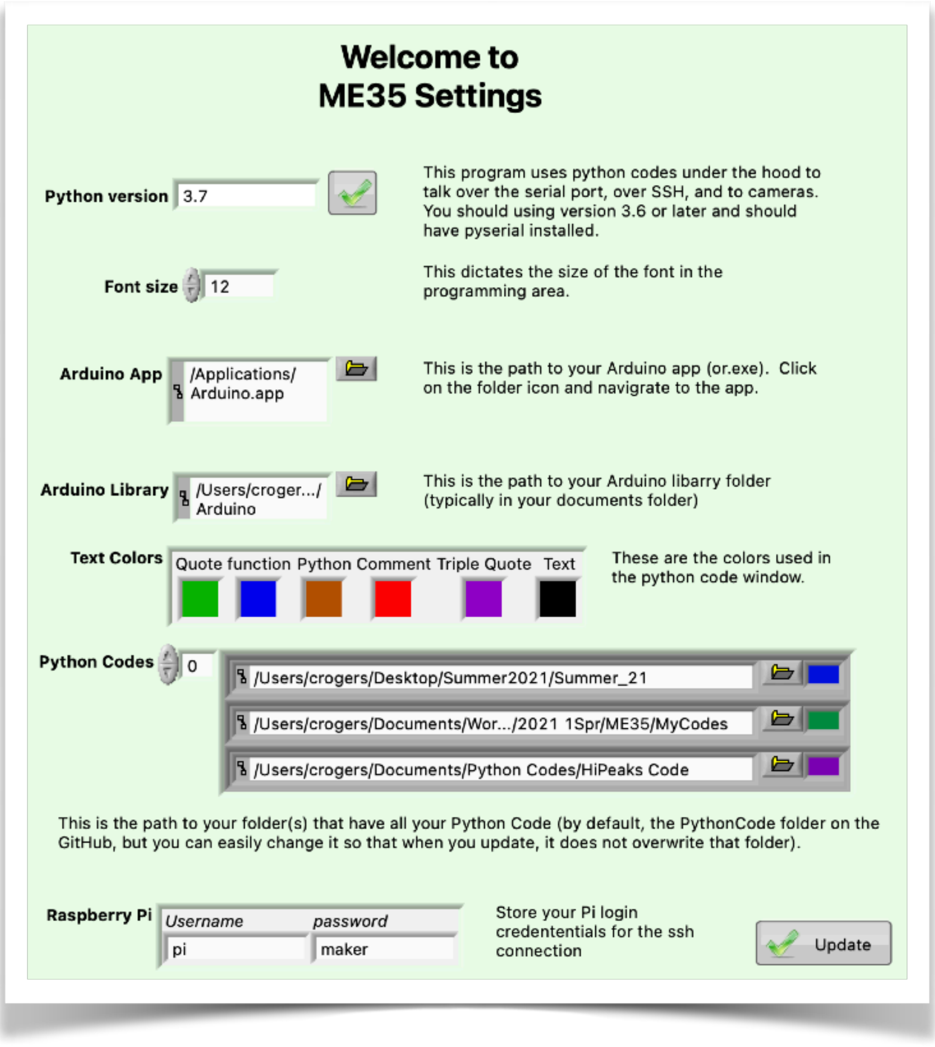
What it is:
This is a LabVIEW code that lets you talk over serial to any MicroPython processor or the Raspberry Pi. It lets you easily jump between programs, copy-paste subsections of the code into the REPL or just type in the REPL to test stuff. It also supports what we call ***backpacks*** - other MicroPython processors that can be controlled through the LEGO SPIKE Prime. Essentially it makes it so that you can easily have SPIKE Prime write to the REPL of the backpack processor. It also supports the WIO Terminal as a backpack, loading some code up on the WIO since the CircuitPython version does not support the WiFi chip.

What it is not:
Most importantly, it is not a commercial piece of software. There are bugs (please report them) and it is designed to support my class. I highly recommend Mu for a more general, entry-level IDE.

1. Installing everything properly
   1. LabVIEW
   2. Python 3.X
2. First Run
   1. Settings
   2. Connect to microprocessor
3. Learn Python
4. Learn LabVIEW

Settings

The settings area lets you control defaults for PyVIEW. First, make sure to enter the correct ***python version*** used by your computer and confirm that LabVIEW can find it with the check box. If it does not fund it, try reinstalling with just the python installer - and make sure to use the correct bit depth (Mac is always 64 bit, many PC versions of LabVIEW are 32 bit - you can check by looking at the splash screen as LabVIEW starts up. ***Font size*** lets you vary the size of the code font and the console font in the IDE. PCs seem to prefer larger fonts. The ***Arduino paths*** are only if you are using the WIO. ***Text colors*** let you choose your favorite color pattern for the coding area. Next, the ***Python codes*** array is a listing of all the folders that the IDE will display in the IDE. Note that you can change the color of the text in the listing depending on which folder it comes from. Finally, if you are using ssh to connect in to your ***Pi***, save the username and password here.



Welcome to ME35 Settings

Python version: 3.7 ✓

This program uses python codes under the hood to talk over the serial port, over SSH, and to cameras. You should using version 3.6 or later and should have pyserial installed.

Font size: 12

This dictates the size of the font in the programming area.

Arduino App: /Applications/Arduino.app

This is the path to your Arduino app (or.exe). Click on the folder icon and navigrate to the app.

Arduino Library: /Users/croger.../Arduino

This is the path to your Arduino libarry folder (typically in your documents folder)

Text Colors

Quote	function	Python	Comment	Triple Quote	Text
🟩	🟦	🟫	🟥	🟪	⬛

These are the colors used in the python code window.

Python Codes: 0

/Users/crogers/Desktop/Summer2021/Summer_21	🟦
/Users/crogers/Documents/Wor.../2021 1Spr/ME35/MyCodes	🟩
/Users/crogers/Documents/Python Codes/HiPeaks Code	🟪

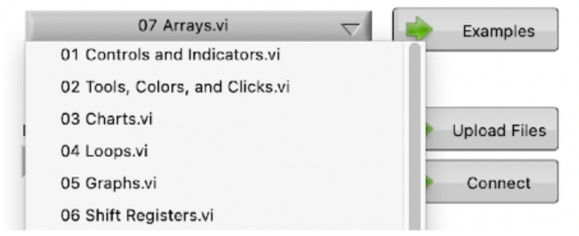
This is the path to your folder(s) that have all your Python Code (by default, the PythonCode folder on the GitHub, but you can easily change it so that when you update, it does not overwrite that folder).

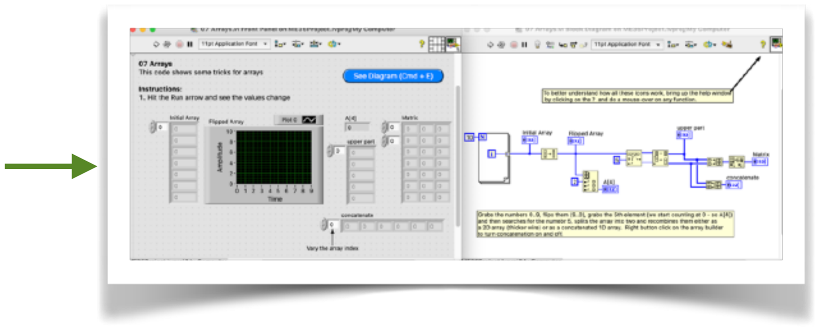
Raspberry Pi: *Username* pi, *password* maker

Store your Pi login credententials for the ssh connection

Closing the window will revert to the last saved values, update will save the current settings.

LabVIEW Examples
PyVIEW is written in LabVIEW with python handling all the serial and ssh communication underneath. In this way, you can edit / modify the code however you want if you know LabVIEW. Learn LabVIEW by playing with examples - this dropdown menu has a bunch of simple labVIEW code examples - simply select one and click on Examples to check it out.

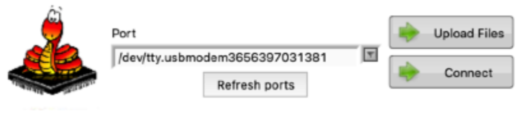


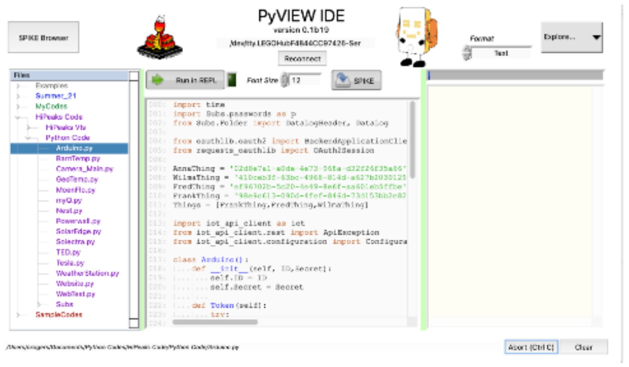


The list of example codes are:

- 01 Controls and Indicators.vi
- 02 Tools, Colors, and Clicks.vi
- 03 Charts.vi
- 04 Loops.vi
- 05 Graphs.vi
- 06 Shift Registers.vi
- 07 Arrays.vi
- 08 Strings.vi
- 09 Clusters.vi
- 10 Rings, Strings, and Tables.vi
- 11 Execution Order.vi
- 12 Events.vi
- 13 Case Structures.vi
- 14 Saving and Reading Files.vi
- 15 Messing with Time.vi
- 16 JSON Packets.vi
- 21 Talking Serial.vi
- 22 RESTful API.vi
- 23 Python Node.vi
- 24 OS Commands.vi
- 51 Simple Queues.vi
- 52 Properties.vi
- 53 Invoke.vi
- 54 Occurrences and more.vi
- 55 Channels.vi
- 56 Panels in panels.vi
- 57 Scripting.vi
- 71 MP_Reading a voltage.vi
- 72 MP_Setting a servo.vi
- 81 Arduino_Reading a voltage.vi
- 82 Arduino_Setting a servo.vi
- 91 WIO_Reading a voltage.vi
- 92 WIO_Setting a Servo.vi

Connecting to a MicroPython processor
Select your processor from the dropdown list - if it is not there, then hit refresh ports. Make sure everything is on and connected. The upload button allows you to select any file you have on your host computer and upload it to the microprocessor. The Connect button opens up the PyVIEW interface. The software tries to guess which processor is yours (anything with usbmodem in it for the Mac and the last COM port for the PC)



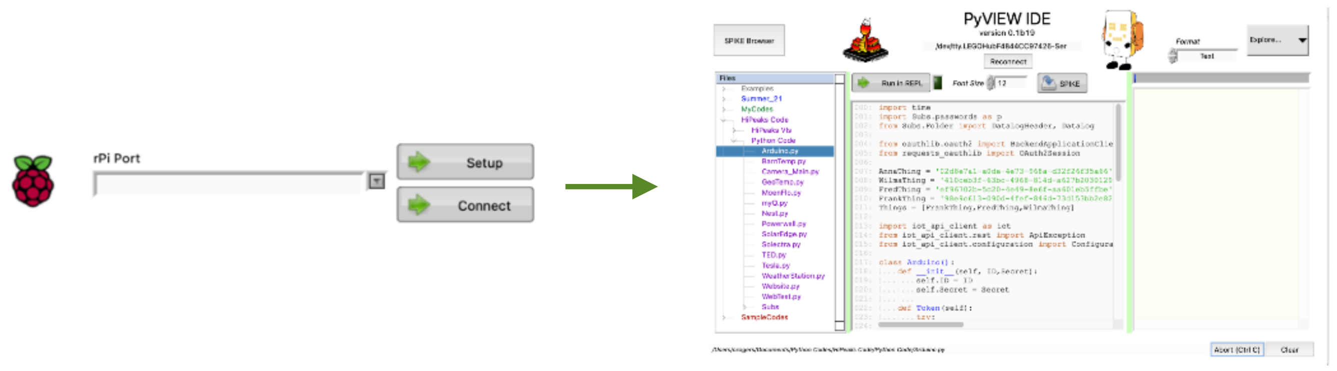


```
import time
import Subs.passwords as p
from Subs.Folder import DatalogHeader, Datalog

from oauthlib.oauth2 import BackendApplicationClie
from requests_oauthlib import OAuth2Session

AnnaThing = '12d8e7a1-a0da-4e73-568a-d32f26f35a86'
WilmaThing = '410ceb3f-63bc-4968-814d-a627b203012
FredThing = 'ef96102b-5c20-4e49-8e6f-aa601eb5ffbe'
FrankThing = '98e9c613-090d-4fef-844d-73d157bb2e82
Things = [FrankThing,FredThing,WilmaThing]

import iot_api_client as iot
from iot_api_client.rest import ApiException
from iot_api_client.configuration import Configura

class Arduino():
    def __init__(self, ID,Secret):
        self.ID = ID
        self.Secret = Secret

    def Token(self):
        try:
```

Once you get to the PyVIEW IDE page, you can click here for more instructions.

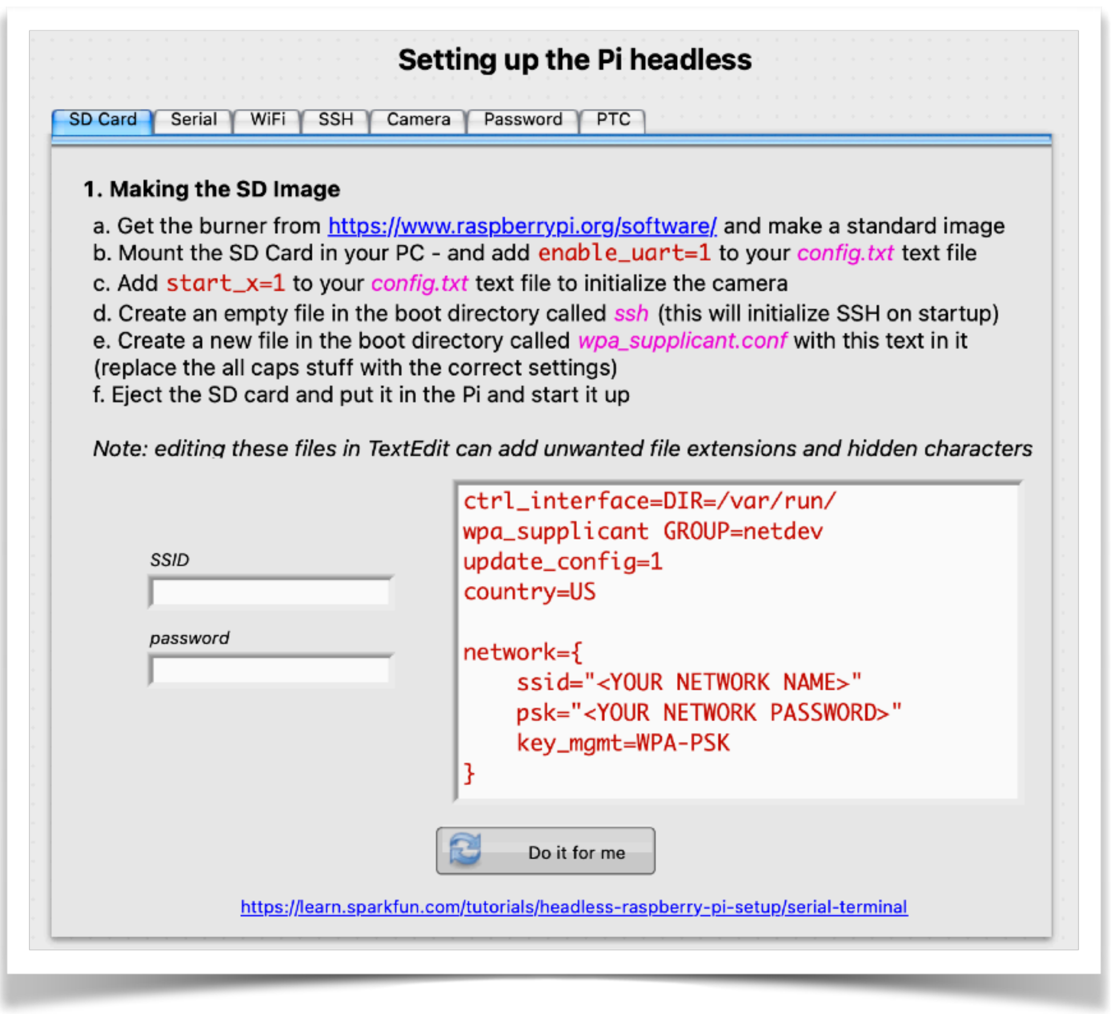
WIO Terminal
Most people can ignore this. If you want to use the WIO Terminal, select it on the serial port and then you can upload the Arduino code that lets you run the WIO Terminal through a uart connection. This uses the paths defined in the Settings page to find the Arduino files and upload the program onto the WIO Terminal.

Raspberry Pi
If you are connecting to a Raspberry Pi, then you can connect over serial or over ssh. For serial, simply select your communications port. For ssh, type in the IP address of the board.



```
import time
import Subs.passwords as p
from Subs.Folder import DatalogHeader, Datalog

from oauthlib.oauth2 import BackendApplicationClie
from requests_oauthlib import OAuth2Session

AnnaThing = '[illegible]'
WilmaThing = '[illegible]'
FredThing = '[illegible]'
FrankThing = '[illegible]'
Things = [FrankThing,FredThing,WilmaThing]

import iot_api_client as iot
from iot_api_client.rest import ApiException
from iot_api_client.configuration import Configura

class Arduino():
    def __init__(self, ID,Secret):
        self.ID = ID
        self.Secret = Secret

    def Token(self):
        try:
```

The setup button takes to you a front panel that reminds me what to do to set up a headless Pi (taken from the Sparkfun site listed at the bottom)



Setting up the Pi headless

SD Card | Serial | WiFi | SSH | Camera | Password | PTC

1. Making the SD Image

a. Get the burner from https://www.raspberrypi.org/software/ and make a standard image
b. Mount the SD Card in your PC - and add enable_uart=1 to your *config.txt* text file
c. Add start_x=1 to your *config.txt* text file to initialize the camera
d. Create an empty file in the boot directory called *ssh* (this will initialize SSH on startup)
e. Create a new file in the boot directory called *wpa_supplicant.conf* with this text in it (replace the all caps stuff with the correct settings)
f. Eject the SD card and put it in the Pi and start it up

Note: editing these files in TextEdit can add unwanted file extensions and hidden characters

SSID

password

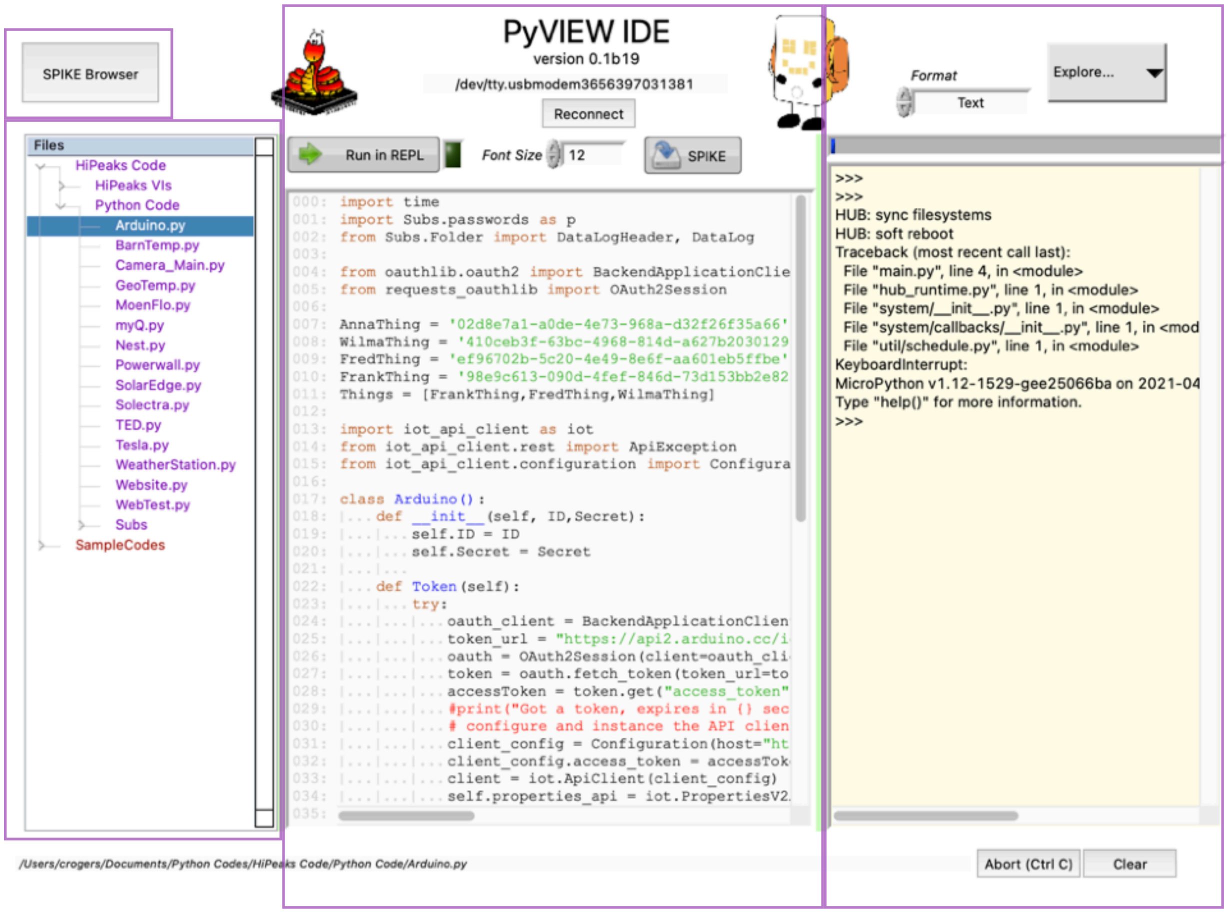
```
ctrl_interface=DIR=/var/run/
wpa_supplicant GROUP=netdev
update_config=1
country=US

network={
    ssid="<YOUR NETWORK NAME>"
    psk="<YOUR NETWORK PASSWORD>"
    key_mgmt=WPA-PSK
}
```

Do it for me

https://learn.sparkfun.com/tutorials/headless-raspberry-pi-setup/serial-terminal

PyVIEW

This is where most of the action happens. Note that you can change the window size and move the green dividers around to your favorite configuration.



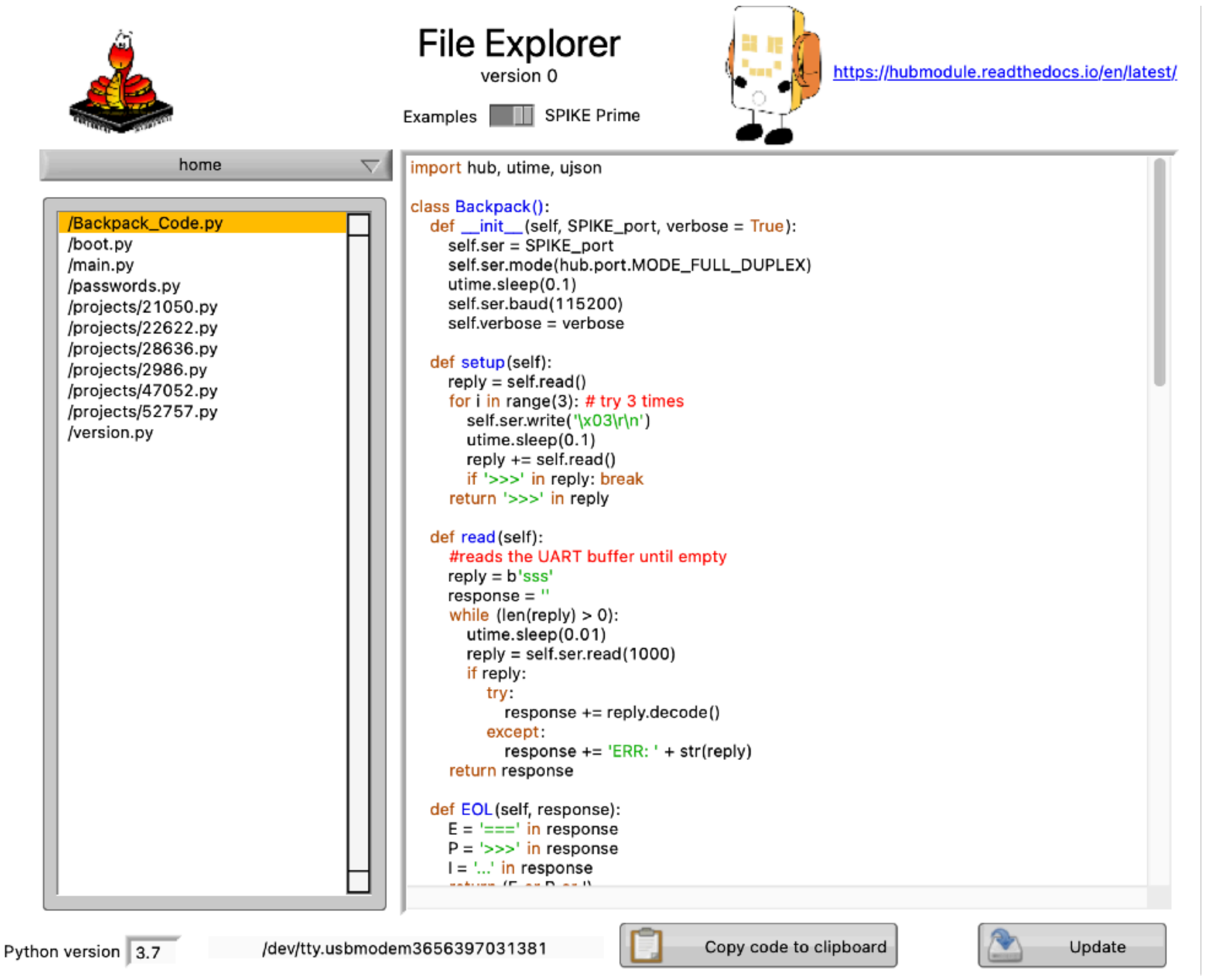
```
import time
import Subs.passwords as p
from Subs.Folder import DataLogHeader, DataLog

from oauthlib.oauth2 import BackendApplicationClie
from requests_oauthlib import OAuth2Session

AnnaThing = '02d8e7a1-a0de-4e73-968a-d32f26f35a66'
WilmaThing = '410ceb3f-63bc-4968-814d-a627b2030129
FredThing = 'ef96702b-5c20-4e49-8e6f-aa601eb5ffbe'
FrankThing = '98e9c613-090d-4fef-846d-73d153bb2e82
Things = [FrankThing,FredThing,WilmaThing]

import iot_api_client as iot
from iot_api_client.rest import ApiException
from iot_api_client.configuration import Configura

class Arduino():
|...def __init__(self, ID,Secret):
|...|...self.ID = ID
|...|...self.Secret = Secret
|...|...
|...def Token(self):
|...|...try:
|...|...|...oauth_client = BackendApplicationClien
|...|...|...token_url = "https://api2.arduino.cc/i
|...|...|...oauth = OAuth2Session(client=oauth_cli
|...|...|...token = oauth.fetch_token(token_url=to
|...|...|...accessToken = token.get("access_token"
|...|...|...#print("Got a token, expires in {} sec
|...|...|...# configure and instance the API clien
|...|...|...client_config = Configuration(host="ht
|...|...|...client_config.access_token = accessTok
|...|...|...client = iot.ApiClient(client_config)
|...|...|...self.properties_api = iot.PropertiesV2.
```

```
>>>
>>>
HUB: sync filesystems
HUB: soft reboot
Traceback (most recent call last):
  File "main.py", line 4, in <module>
  File "hub_runtime.py", line 1, in <module>
  File "system/__init__.py", line 1, in <module>
  File "system/callbacks/__init__.py", line 1, in <mod
  File "util/schedule.py", line 1, in <module>
KeyboardInterrupt:
MicroPython v1.12-1529-gee25066ba on 2021-04
Type "help()" for more information.
>>>
```

/Users/crogers/Documents/Python Codes/HiPeaks Code/Python Code/Arduino.py

Browse Button

This area allows you to look at files on the processor (or on your computer). Sometimes you might need to reload this (close and open again) if it did not read the files properly. If you change any of the text in the file, you can update the file on the processor by hitting the update button.



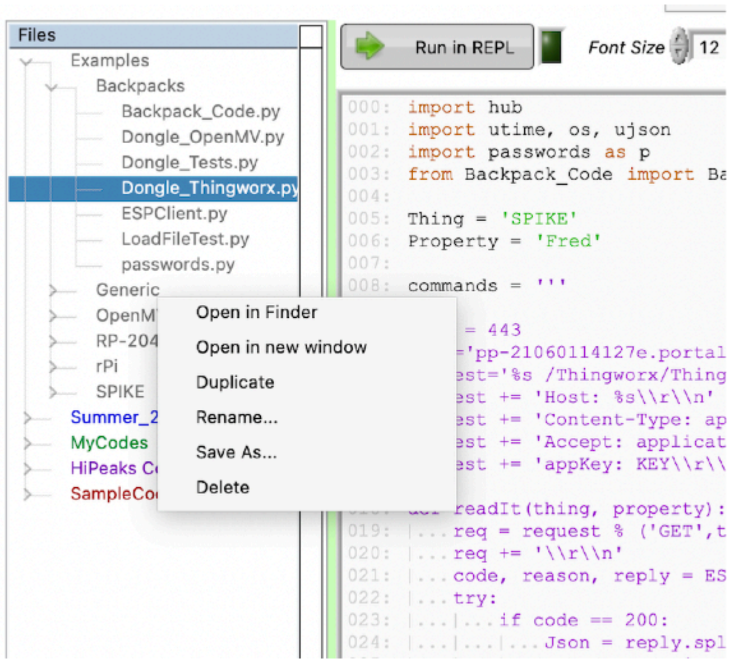
```
import hub, utime, ujson

class Backpack():
    def __init__(self, SPIKE_port, verbose = True):
        self.ser = SPIKE_port
        self.ser.mode(hub.port.MODE_FULL_DUPLEX)
        utime.sleep(0.1)
        self.ser.baud(115200)
        self.verbose = verbose

    def setup(self):
        reply = self.read()
        for i in range(3): # try 3 times
            self.ser.write('\x03\r\n')
            utime.sleep(0.1)
            reply += self.read()
            if '>>>' in reply: break
        return '>>>' in reply

    def read(self):
        #reads the UART buffer until empty
        reply = b'sss'
        response = ''
        while (len(reply) > 0):
            utime.sleep(0.01)
            reply = self.ser.read(1000)
            if reply:
                try:
                    response += reply.decode()
                except:
                    response += 'ERR: ' + str(reply)
        return response

    def EOL(self, response):
        E = '===' in response
        P = '>>>' in response
        I = '...' in response
```

Python version 3.7

/dev/tty.usbmodem3656397031381

File Tree
The File Tree lists all the *.py files in the folders you specified in the Settings area. The text color makes it easy to differentiate files from multiple folders. If you Right Btn Click on the tree (or go to the File menu) you hav a few options, including going to the location of the File on your computer (Explorer or Finder), saving it, renaming it, duplicating it or deleting it. All of these apply to the file that is currently selected, independent of where your mouse is. ***Open in New Window*** lets you view your python code in multiple windows. Although if you end up having multiple windows of the same code, you will run into issues because each window will save a new version of the file whenever you change it - but will NOT load a new version any time the code is changed in another window.

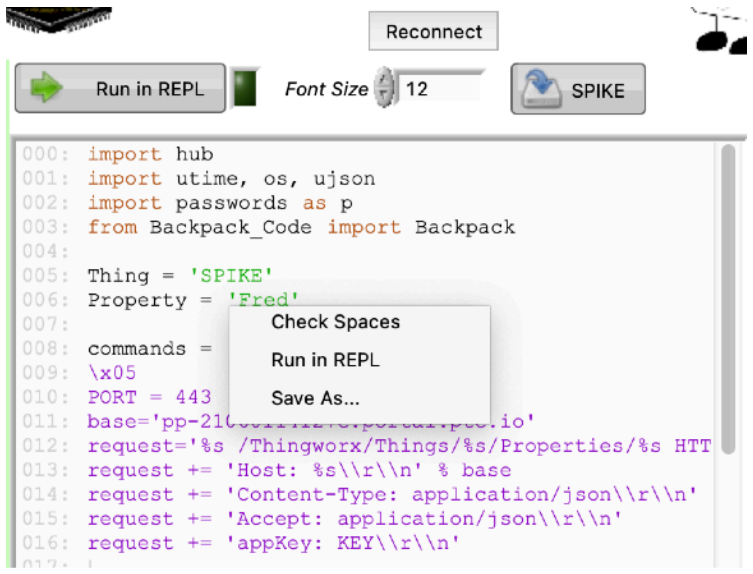
Finally - you can refresh the tree at any time by selecting the Tools menu (Reload File Tree).



```
import hub
import utime, os, ujson
import passwords as p
from Backpack_Code import Ba

Thing = 'SPIKE'
Property = 'Fred'

commands = '''

 = 443
='pp-21060114127e.portal
est='%s /Thingworx/Thing
est += 'Host: %s\\r\\n'
est += 'Content-Type: ap
est += 'Accept: applicat
est += 'appKey: KEY\\r\\

readIt(thing, property):
    req = request % ('GET',t
    req += '\\r\\n'
    code, reason, reply = ES
    try:
        if code == 200:
            Json = reply.spl
```

Python Code and Controls

The code window will always contain the code selected in the Tree. A right Btn Click (or Tools menu) lets you check all your spaces (and check for hidden characters) or save it as a new name (similar to "Duplicate" on the tree). ***Run in REPL*** (or the Run in REPL button) will essentially type each line, line by line, into the REPL and you can see it get executed - this is very useful for debugging in that you can also query the values of different variables in the REPL as well. You can also select multiple lines from your code and paste them into the REPL. The software will remove spaces and clean things up for you as you paste it. The green button to the right of the Run in REPL button will leverage the Ctrl E microPython environment - basically a faster download. You can also adjust the ***Font Size*** to make it easier to read, ***Reconnect*** with your hardware, and download files directly to the microprocessor with the ***SPIKE*** button. Finally, the ***reconnect*** button tries to reconnect to your processor after it inadvertently decided to bo back into sleep mode and you have woken it up again.



```
import hub
import utime, os, ujson
import passwords as p
from Backpack_Code import Backpack

Thing = 'SPIKE'
Property = 'Fred'

commands =
\x05
PORT = 443
base='pp-21            .io'
request='%s /Thingworx/Things/%s/Properties/%s HTT
request += 'Host: %s\\r\\n' % base
request += 'Content-Type: application/json\\r\\n'
request += 'Accept: application/json\\r\\n'
request += 'appKey: KEY\\r\\n'
```

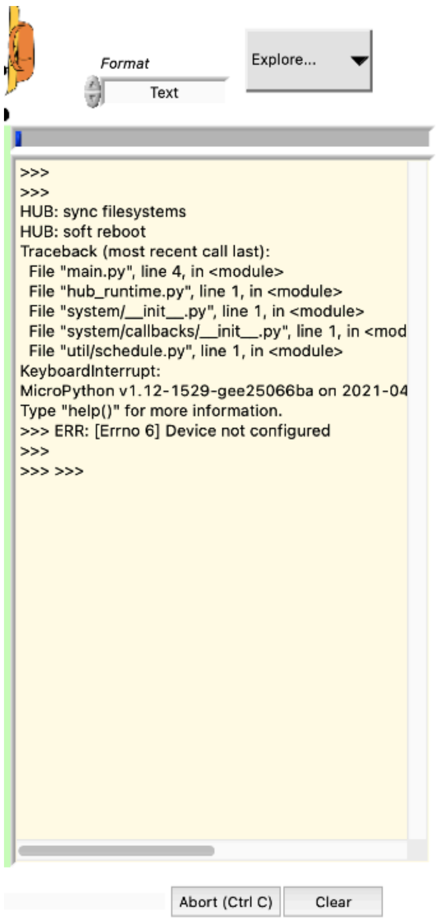
Console

The console area lets you directly interact with the REPL on the processor. There are a number of ways you can view the REPL - as text (as in the image to the right), or a plot etc. The different selections in the ***Format*** text ring let you see the data as:

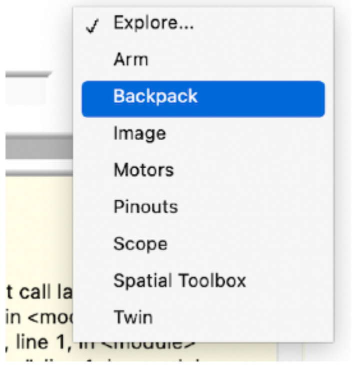
1. Text - you can see and type in this area - errors are highlighted in red
2. Chart - this takes any line in the text buffer that is just one number and puts it on the chart.
3. Waveforms - this takes any line in the text buffer that is an array and plots the array
4. JPEG - this is not working yet - but will eventually show a JPEG instead of adding the text to the text buffer
5. Explore - this will give you a split view - handy for when you are running backpacks

The console area also has an ***Abort*** button that sends a Ctrl-C (\x03) and a ***Clear*** button which clears the text buffer. The blue line on the top tells you how full the buffer is. When the buffer is too full, things slow down. You can also ***Save*** the console log or clear it with the right button menu or the Tools menu.

The ***Explore…*** menu lets you choose from a number of separate windows. These are LabVIEW front panels that read the latest line from the console and interact with it in some way. It is easy to add more as well. Most of these were customized for class but the ***Pinouts*** one is pretty useful when wiring things up.



```
>>>
>>>
HUB: sync filesystems
HUB: soft reboot
Traceback (most recent call last):
  File "main.py", line 4, in <module>
  File "hub_runtime.py", line 1, in <module>
  File "system/__init__.py", line 1, in <module>
  File "system/callbacks/__init__.py", line 1, in <mod
  File "util/schedule.py", line 1, in <module>
KeyboardInterrupt:
MicroPython v1.12-1529-gee25066ba on 2021-04
Type "help()" for more information.
>>> ERR: [Errno 6] Device not configured
>>>
>>> >>>
```


Explore...
Arm
Backpack
Image
Motors
Pinouts
Scope
Spatial Toolbox
Twin