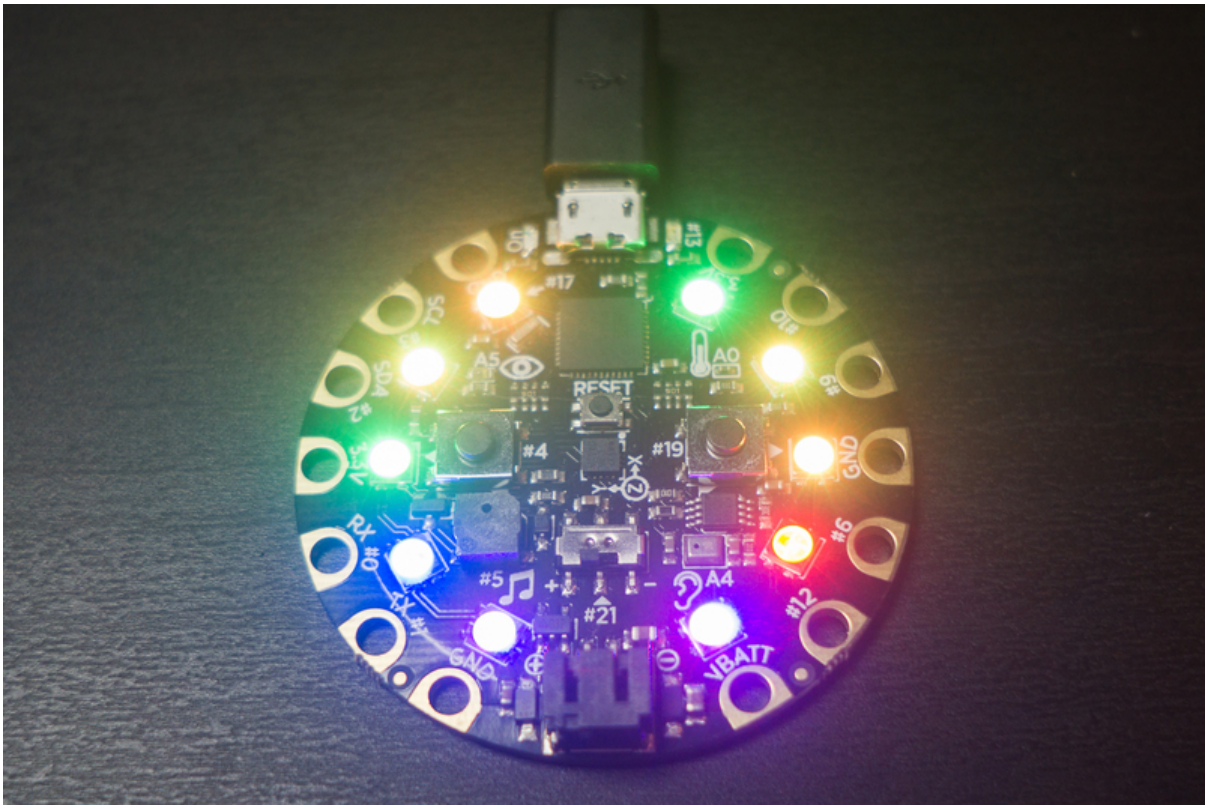
Circuit Playground Firmata

Created by Tony DiCola



https://learn.adafruit.com/circuit-playground-firmata

Last updated on 2024-06-03 01:54:34 PM EDT

Table of Contents

Overview

Circuit Playground (http://adafru.it/3000) is Adafruit's all-in-one Arduino-compatible physical computing board. This tiny board packs a lot of cool hardware like an Arduino-compatible microcontroller, NeoPixels (addressable RGB LEDs), an accelerometer to detect forces, tilting, and taps, sensors for temperature, light, and sound, buttons & capacitive touch inputs, and more. All of these components come preassembled and soldered to the board--you're ready to start learning and playing with Circuit Playground immediately!

However one challenging aspect for newcomers to programming and hardware is learning how to use programming languages like C++ & C. Higher level programming languages like Python are typically easier for beginners to learn, but can you control Circuit Playground with Python or other high level languages? It turns out that yes you can use high level programming languages with Circuit Playground using a tool called Firmata!

Firmata (https://adafru.it/ncF) is a protocol for talking to and controlling Arduino-compatible boards. By programming Circuit Playground with a special Firmata sketch you can control it from Python and other code running on your computer. This greatly reduces the barrier to entry when learning programming and hardware. Just plug Circuit Playground into your computer and start writing Python and other code to control its hardware!

This guide will show you how to program Circuit Playground with a special Firmata sketch that allows it to be controlled from your computer. You'll also see how to run Python example code that talks to the Circuit Playground Firmata board and controls all of its hardware, like lighting up the NeoPixels, reading the accelerometer, and much more.

Before you get started make sure to follow the Circuit Playground guide (https://adafru.it/ncG) to familiarize yourself with the board and setup the Arduino IDE to program it. Then continue on to learn about loading the Circuit Playground Firmata sketch.

Firmata Sketch

To load the Circuit Playground Firmata sketch you'll first need to make sure the Arduino IDE is setup to program Circuit Playground. Follow the Circuit Playground guide (https://adafru.it/ncG) and make sure you can upload basic sketches like blink to the board.

Next you'll need to install a library which the sketch uses. Use the library manager (https://adafru.it/fCN) in the latest versions of Arduino to search for and install this library:

- Adafruit Circuit Playground (https://adafru.it/naF)

Finally download the latest version of the Circuit Playground Firmata code from its home on GitHub (https://adafru.it/ncH) by clicking the button below:

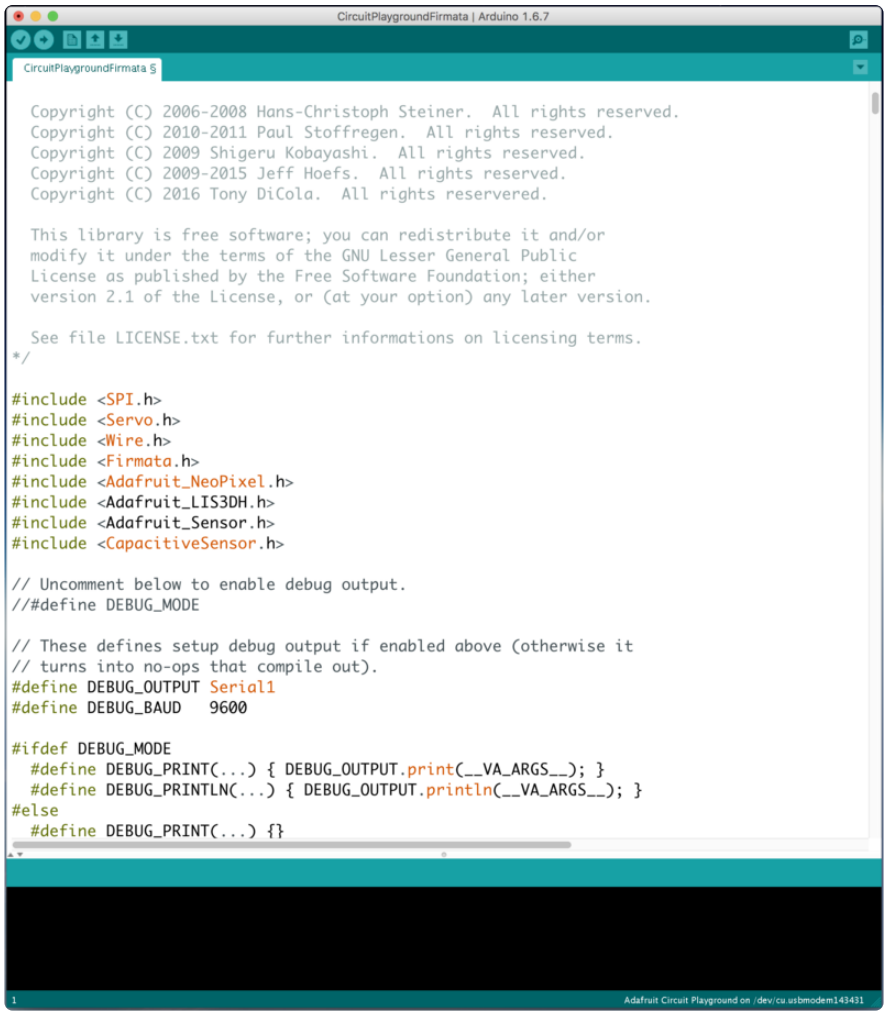
Download Circuit Playground Firmata Code

https://adafru.it/ncI

Unzip the archive and you'll see two folders inside:

- **CircuitPlaygroundFirmata** - This is the Arduino sketch you'll load on the Circuit Playground board to make it a Firmata device.
- **Python Examples** - These are examples of Python code that can control a Circuit Playground board running the Firmata sketch.

Open the CircuitPlaygroundFirmata sketch in the Arduino IDE and you should see something like the following:



There isn't anything you normally need to change in the sketch, however if you'd ever like to enable debug output on the serial pins (pin 0 and 1 on Circuit Playground) you can uncomment the debug mode line near the top:

```
// Uncomment below to enable debug output.
//#define DEBUG_MODE
```

With debug mode enabled the serial pins 0 and 1 will output debug text during certain Firmata actions. You can hook up a USB to serial cable to these pins to read the output (use 9600 baud).

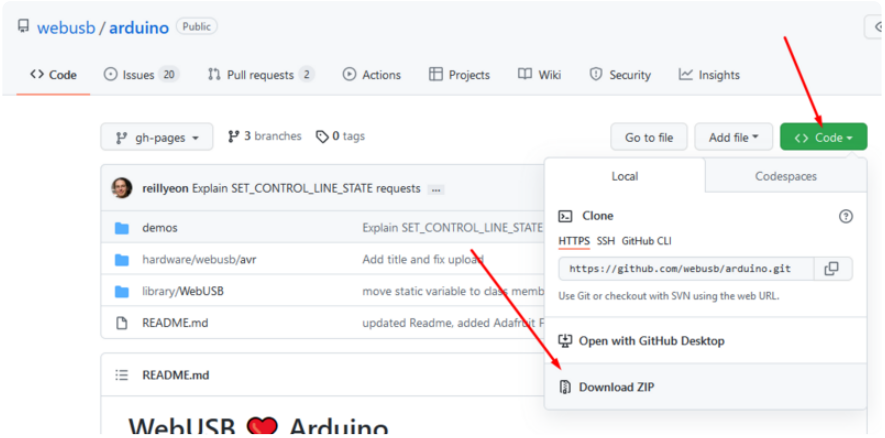
Install WebUSB Library

You'll need to install a library first, before uploading.

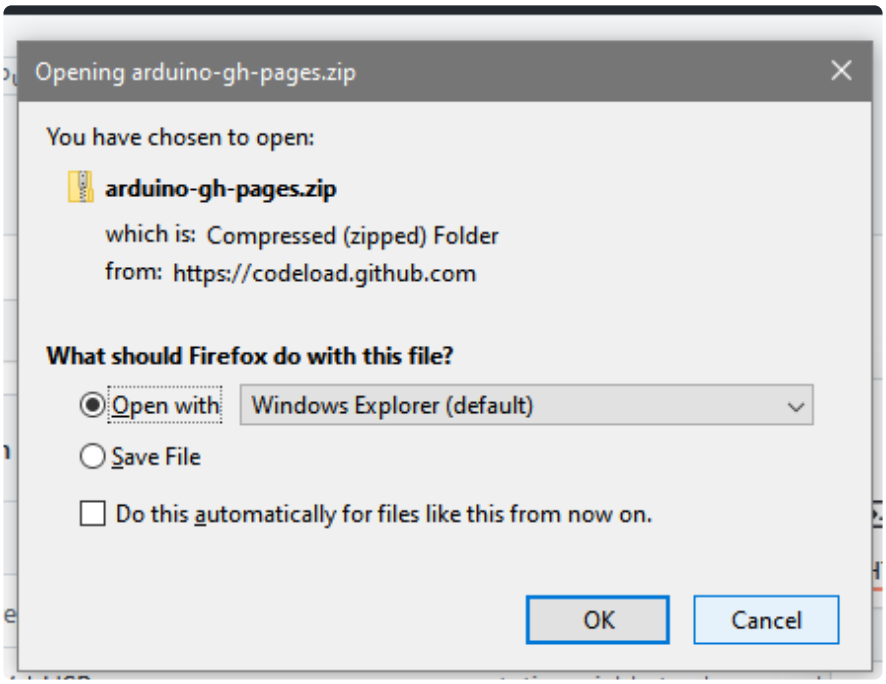
Visit the WebUSB repository at

https://github.com/webusb/arduino **(https://adafru.it/XOE)**

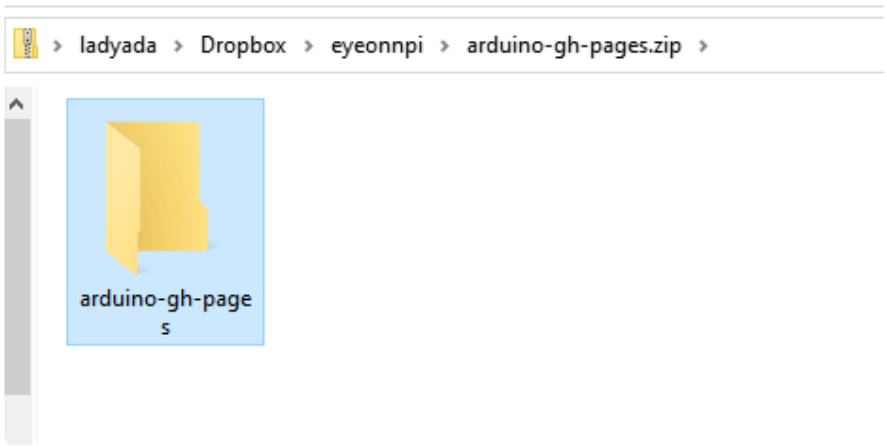
Once there, **click the Code button in the top-right** to get the code download. Then click **Download Zip**



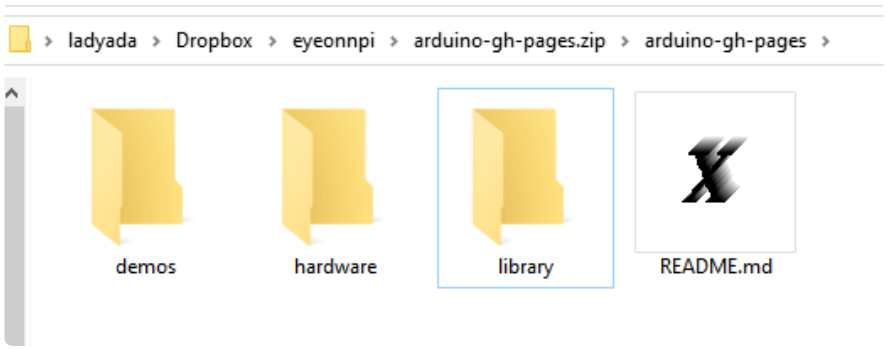
You will get a zip file to download. Save it to your desktop or somewhere else convenient!



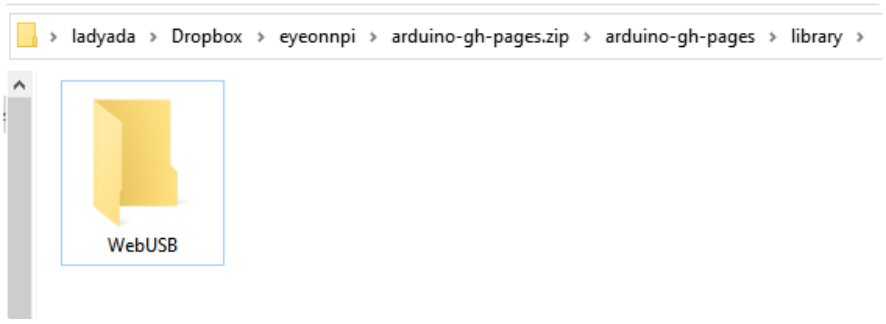
Open the zip to find a folder called gh-pages



Then open that to find a folder called **Library**

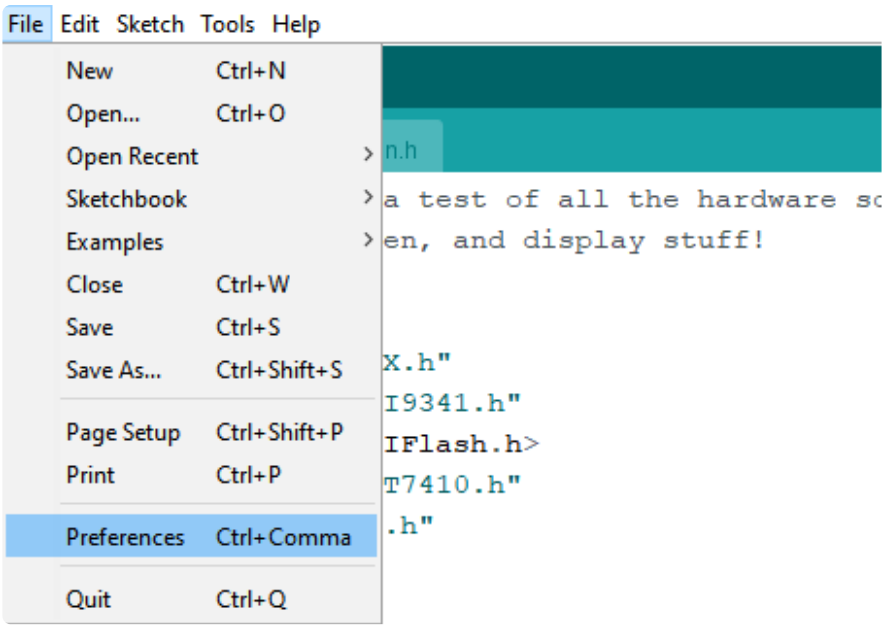


Then inside that folder is a folder called **WebUSB**

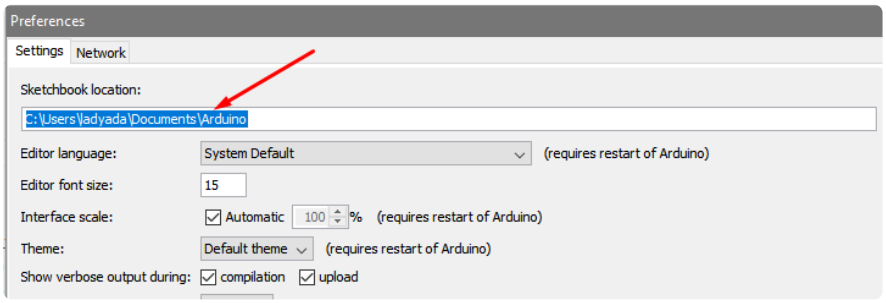


Drag the WebUSB folder out to your Desktop

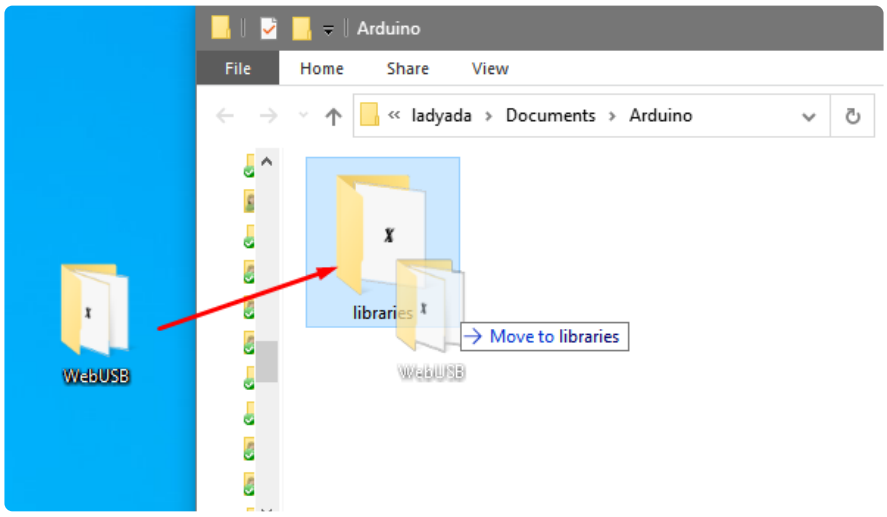
Open up your Sketches folder. Find it by going to the Arduino Preferences menu



It's called **Sketchbook location**, find and open that location on your computer!



Drag the WebUSB folder you extracted earlier into the **libraries** folder within your sketchbook folder.



Now make sure Circuit Playground is selected in the **Tools -> Board** menu, and the serial port for Circuit Playground is selected in the **Tools -> Port** menu. Then press the upload button in the Arduino IDE to compile and upload the Firmata sketch.

Note that if the upload fails for some reason try double pressing the reset button on Circuit Playground. This will force it into bootloader mode and you should see the red LED start pulsing on and off. As soon as the red LED pulses press upload and try uploading the sketch again.

Once the sketch has successfully uploaded you're all set and are ready to start using Firmata clients to control Circuit Playground. Continue on to the next page to learn about Python example code to control Circuit Playground.

Example Python Code

Once your Circuit Playground is running the Firmata sketch you can use many different Firmata clients to control the board. For example standard Firmata clients can control the digital pins on the board and even read some of the analog sensors like the thermistor and light sensor. However to make full use of the Circuit Playground Firmata sketch you'll want to use a client that's specially modified to use custom Firmata commands that interact with Circuit Playground's on-board hardware like its accelerometer, NeoPixels, and more.

Included in the repository with the Circuit Playground Firmata sketch (https://adafru.it/ncH) is a set of Python code to talk to Circuit Playground Firmata. This code implements all of the custom Firmata extensions to control every component on the board. You can write simple Python programs to light up the NeoPixels, read the accelerometer & tap detection, capacitive inputs, and much more.

If you're using a different Firmata client (https://adafru.it/ncF) you might need to look into how it supports custom Firmata SysEx extensions. See the Firmata Extension Reference page (https://adafru.it/ncJ) for details on the custom Firmata extensions for Circuit Playground.

Install Dependencies

To get started you'll first need to install a few dependencies to use the Python Firmata example code. The Python code is based on the excellent PyMata library (https://adafru.it/ncK) which implements most of the code to talk to Circuit Playground Firmata.

First make sure you've installed the latest version of Python (https://adafru.it/fa7), either 2.7.x or 3.4+.

Next you will want to install the pip Python package manager (https://adafru.it/ncL). Some recent versions of Python include pip, however if you don't have pip installed (i.e. running the **pip** command at the command prompt fails with an unknown command error) follow the installation instructions (https://adafru.it/ncM) to download and run get-pip.py.

After pip is installed you're ready to install the PyMata library (https://adafru.it/ncK). Open a command prompt and run the following command to install the library:

```
pip install pymata
```

Note on Linux and Mac OSX you might need to prefix the command with sudo so it runs as root and installs pymata globally:

```
sudo pip install pymata
```

Make sure you see the PyMata installation succeeds, for example you might see text like:

```
Collecting pymata
  Downloading PyMata-2.12.tar.gz
Collecting pyserial&gt;=2.7 (from pymata)
  Downloading pyserial-3.0.1.tar.gz (134kB)
    100% |████████████████████████████████| 143kB 3.9MB/s
Installing collected packages: pyserial, pymata
  Running setup.py install for pyserial ... done
  Running setup.py install for pymata ... done
Successfully installed pymata-2.12 pyserial-3.0.1
```

If you see the installation fail with an error go back and check you have both Python and pip installed and try again.

Once PyMata is installed you're ready to run the Python Circuit Playground Firmata code.

Python Circuit Playground Firmata Code

To run the Python Circuit Playground Firmata code first make sure you have a Circuit Playground board that is running the Circuit Playground Firmata sketch. Check out the previous page (https://adafru.it/ncN) if you haven't setup a Circuit Playground with the Firmata sketch yet.

Next make sure you've downloaded the Circuit Playground Firmata repository (https://adafru.it/ncH) code. You can use the button below to download this code:

Download Circuit Playground Firmata Code

https://adafru.it/ncl

Unzip the archive and notice the **Python Examples** subfolder. Inside this folder is all the Python code to control Circuit Playground Firmata.

Before you run the example code first make sure the Circuit Playground board is plugged into the computer using its micro-USB port. You'll also want to find the name of the serial port for Circuit Playground. Check the **Tools -> Port** menu in the Arduino IDE and remember the name of the port that you see for Circuit Playground, like **COM1**, **/dev/tty.usbmodem143431**, **/dev/ttyUSB0**, etc.

Open a command prompt and navigate to the **Python Examples** folder. Start by running a simple example to test the buttons on the Circuit Playground board:

```
python buttons.py /dev/tty.usbmodem143431
```

Replace /dev/tty.usbmodem143431 with the name of the serial port for Circuit Playground on your computer.

You should see text like the following that shows PyMata connecting to the board and then the example code waiting for buttons to be pressed:

```
Python Version 2.7.11 (default, Jan 22 2016, 08:29:18)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-700.1.81)]

PyMata version 2.12  Copyright(C) 2013-16 Alan Yorinks    All rights reserved.

Opening Arduino Serial port /dev/tty.usbmodem143431

Please wait while Arduino is being detected. This can take up to 30 seconds ...
Board initialized in 0 seconds
Total Number of Pins Detected = 30
Total Number of Analog Pins Detected = 12
Press the left button, right button, or slide switch (Ctrl-C to quit)...
Switch is on the left!
```

Try pressing and releasing the left button, right button, and moving the slide switch between its left and right positions. You should see text printed to the console:

```
Left button pressed!
Left button released!
Right button pressed!
Right button released!
Switch is on the right!
Switch is on the left!
```

If you see an error check you have Python and the PyMata library successfully installed. Also confirm Circuit Playground is connected to the serial port you specified when running the example. Finally be sure the Circuit Playground board is running the Circuit Playground Firmata sketch--it doesn't hurt to reload the sketch if you're unsure.

As the buttons.py code runs you can see it listens for button presses on the Circuit Playground board and prints text when they are pressed. This is a great example of using Circuit Playground to interact with Python code running on your computer. When you're finished press **Ctrl-C** in the terminal to stop the program.

Try running other examples in the Python Examples directory. Each example is run in the same way as buttons.py, you must specify a command line parameter with the name of the serial port for Circuit Playground.

Note: If you see an error like **IndexError: list out of range** this means PyMata is a little confused about all the data being sent to it as it connects to the Circuit Playground board. Usually this happens when a previous example was run and it turned on streaming of values to the host computer. You can fix this by unplugging and plugging back in the Circuit Playground board. This will reset it and make sure it isn't streaming any data that confuses PyMata.

Below is a description of each example:

- **accelerometer.py** - This example will take a single reading from the accelerometer's X, Y, Z axis every 2 seconds and print it out. This is a simple example of how to read the accelerometer with a function call, but you might want to look at the **accelerometer_streaming.py** example for a faster and more efficient way to use it. The units for accelerometer data are meters per second squared.
- **accelerometer_streaming.py** - This example uses fast streaming to quickly read and print the accelerometer's X, Y, Z axis acceleration. The code will show how streaming can be turned on and off (for 5 seconds in the example) and then on again. When accelerometer data is streamed you'll see much faster update rates, up to about 5 times a second, compared to the simpler **accelerometer.py** example. This is great for quickly detecting when the acceleration and orientation of the Circuit Playground board changes. The units for accelerometer data are meters per second squared.
- **buttons.py** - This example will listen for button presses on the Circuit Playground (left button, right button, slide switch) and print them out.
- **cap_touch.py** - This example will listen for capacitive touches on pin 10 of Circuit Playground. Every two seconds the pin will be checked and if a large

enough capacitance is found, like if a human is touching it, then a message will be printed. This is a simple example of reading a capacitive touch input. You can see a faster and more efficient way with the **cap_streaming.py** example.

- **cap_streaming.py** - This example will listen for capacitive touches on both pins 3 and 10 of Circuit Playground. The code shows how these changes are quickly streamed back to the host computer for fast responses. In addition the code shows how streaming can be turned on and off and back on again. Streaming the capacitive touch input is faster than the simple **cap_touch.py** example and will help programs quickly detect capacitive touches.
- **circuitplayground.py** - This actually is **not** an example to run, instead it's a helper class to simplify talking to the Circuit Playground Firmata board. You can copy this file and place it in the same directory as Python scripts you write yourself to control Circuit Playground!
- **light_sensor.py** - This example will print out the light sensor value as it changes. Note that you'll only see values printed when there is a change in light level--if the light level isn't changing then it won't print out. The value has no units but is proportional to how bright the light is hitting the sensor. Darkness will have a low value and bright light will have a very high value in the thousands. Experiment to see what values appear with different levels of light!
- **pixels.py** - This example will animate lighting the NeoPixels on Circuit Playground. As the example runs you'll see a rainbow of colors move along the LEDs.
- **sound.py** - This example will read the microphone and print out its readings as it changes. Note that these readings are raw samples from the microphone and don't necessarily relate to volume level or sound pressure--you might need to average out or apply other smoothing to find the volume of sound.
- **tap.py** - This example will read the tap detection on the LIS3DH accelerometer every two seconds and print out if the board has been tapped or double-tapped. Note that this example is meant to show a simple way to check the tap detection state and isn't very accurate or fast at detecting taps. Instead look at the **tap_streaming.py** example for a fast and efficient way to detect taps.
- **tap_streaming.py** - This example will stream tap detection to the computer. The code will turn on, off, and then on again streaming of tap detection. Try tapping the LIS3DH accelerometer (tiny square in the exact middle of the Circuit Playground board) and you should see single and/or double tap events printed. Note that tap detection is a little 'noisy' and you might see both a single and double tap or even multiple tap events so be aware in your own code that taps might occur multiple times in a short period.
- **temperature.py** - This example will print the temperature from the thermistor every second. In addition it will print the 'raw' analog to digital converter value for the thermistor in case you want to do your own processing on it.

- **tones.py** - This example will play a music scale using the buzzer on Circuit Playground. The code demonstrates simple tone playback at different frequencies and durations.

If you'd like to write your own Python code to control Circuit Playground you'll want to use the included **circuitplayground.py** file. This file implements a class that simplifies talking to the Circuit Playground board and all of the examples above use this class to interact with Circuit Playground.

To use the class in your own code make sure you **copy circuitplayground.py into the same directory as your code**. Then just import circuitplayground.py by adding to the top of your Python code:

```
# Import CircuitPlayground class from the circuitplayground.py in the same
directory.
from circuitplayground import CircuitPlayground
```

Check out the code for the examples above to see how they use the **circuitplayground.py** class to talk to the board. There are simple functions to read sensors like the thermistor, light sensor, microphone, etc. In addition there are functions to read the accelerometer, tap detection, capacitive touch inputs, and more. Some of the components are actually read using PyMata's standard analog and digital input functions, while others use special Firmata extensions to control parts of the board like the NeoPixels. The great thing is that your code doens't need to worry about how Circuit Playground Firmata works or is implemented--just call the functions in **circuitplayground.py** and you'll be controlling the board in no time!

That's all there is to using Circuit Playground Firmata with Python! Let your imagination run wild as you write Python code to control Circuit Playground!

Firmata Extension Reference

This page describes the format for custom extensions to the standard Firmata sketch that the Circuit Playground Firmata sketch implements. These extensions allow Firmata clients to interact with the devices on the board more easily, like to read accelerometer data or light up the NeoPixels.

If you're developing a library or code to talk to the Circuit Playground Firmata sketch this page will be very useful to review. However if you're just planning to use Circuit Playground Firmata with Python or other existing Firmata clients then you don't need to worry about the information here--you're all set to start exploring Firmata and Circuit Playground!

Also it is highly recommended that you skim through and familiarize yourself with the Firmata protocol (https://adafru.it/ncO) before diving too deeply into these custom extensions.

Circuit Playground Firmata SysEx Commands

To support controlling the components of Circuit Playground with Firmata a set of custom Firmata/MIDI SysEx commands are implemented. These commands follow the SysEx message format in the Firmata protocol (https://adafru.it/ncP) and use a variable number of arguments depending on the command. None of the standard Firmata commands have been modified with Circuit Playground Firmata, only new SysEx commands were added.

For all Circuit Playground SysEx commands they start with a unique byte value that identifies the command as specific to Circuit Playground. The value of this Circuit Playground SysEx command byte is **0x40.** SysEx messages that start with this byte will be interpreted as Circuit Playground commands.

Following the Circuit Playground SysEx identifier is sub-command byte (or in some cases two bytes) that identifies the type of Circuit Playground command. Each Circuit Playground command has a unique sub-command value that's described below.

All bytes that follow the sub-command byte are parameters to the Circuit Playground command. See the description of a command below to see the exact number and format of parameters to expect with the command. Remember a byte in the Firmata protocol is only 7-bytes long--the high order bit is always 0 to indicate the byte being data instead of a command. In order to send an 8-bit or larger value it must be broken down into multiple 7-bit bytes and reassembled. Carefully read the description of commands to understand how and when they break large values into smaller 7-bit bytes.

Below is a description of each custom Circuit Playground SysEx command:

NeoPixel Commands

Set NeoPixel Color

Sub-command byte: 0x10

Parameters: 5 bytes total

- **NeoPixel ID (1 byte):** This is the number of the NeoPixel on the board (0-9) to change.
- **Color (4 bytes):** These 4 bytes pack in 24 bits of color data for the NeoPixel in red, green, blue component order. Each component is a full 8-bit byte with a value from 0-255 (lowest intensity to highest intensity). For example the bits of the color parameters look like: 0RRRRRRR 0RGGGGGG 0GGBBBBB 0BBB0000 Notice the first bit of each byte is 0 (Firmata protocol requirement) and color component bits are tightly packed into the bits that follow.

Description: This command will change the color of the specified NeoPixel. Note that the pixel won't actually change, instead just the color in the pixel data buffer will be updated. You must call the pixel show command to push the pixel data buffer to all the pixels and update them.

Example: To change the color of pixel 2 to red = 255, green = 127, blue = 63 the following SysEx command bytes are sent:

0x40 0x10 0x02 0x7F 0x5F 0x67 0x70

Show NeoPixels

Sub-command byte: 0x11

Parameters: None

Description: This command pushes the current NeoPixel color buffer out to the physical pixels. After calling this command the pixels will update their colors.

Example: To call the show NeoPixels command send the following SysEx command bytes:

0x40 0x11

Clear All NeoPixels

Sub-command byte: 0x12

Parameters: None

Description: This command will clear all NeoPixels in the pixel data buffer to 0/off. You must call the show NeoPixels command to update the pixels after calling this command!

Example: To call the clear all NeoPixels command send the following SysEx command bytes:

0x40 0x12

Set NeoPixel Brightness

Sub-command byte: 0x13

Parameters: 1 byte

- **Brightness (1 byte):** This is the brightness value to set for the NeoPixels (see the NeoPixel library setBrightness function for more information). The value should be a byte from 0 to 100 inclusive.

Description: This command will adjust the brightness of all the NeoPixels. The value should be 0 to 100 where 0 is completely dark and 100 is full brightness. Note that animating brightness is not recommended as going down to 0 will 'lose' information and not bable to go back up to 100. Instead just use this function to set the brightness once at the start. Note that by default the pixels are set to 20% brightness.

Example: To change the brightness to 50% send the following SysEx command bytes:

0x40 0x13 0x32

Speaker/Buzzer Commands

Play Tone

Sub-command byte: 0x20

Parameters: 4 bytes total

- **Frequency (2 bytes):** These two 7-bit bytes define an unsigned 14-bit frequency for the tone in hertz (0-16,383). The byte order is little endian with the first byte defining the lower 7 bits and the second byte defining the upper 7 bits.

- **Duration (2 bytes):** This is the duration to play the tone in milliseconds. Like frequency this is an unsigned 14-bit value in little endian byte order. Note that a value of 0 (zero) will start playback of a tone with no duration--the stop tone function must be called to stop playback.

Description: This command will play a tone of the specified frequency on the speaker/buzzer. The tone is generated using a square wave so it will have a slightly harsh sound but is great for simple sounds and music. You can specify the duration the tone should be played, or you can specify to start playing the tone forever until a stop tone command is sent.

Example: To play a 440 hz tone for 2 seconds (2000 milliseconds) send the following SysEx command bytes:

0x40 0x20 0x38 0x03 0x50 0x0F

To start playback of a 338 hz tone send (no duration, i.e. play forever) the following SysEx command bytes:

0x40 0x20 0x52 0x02 0x00 0x00

Stop Tone

Sub-command byte: 0x21

Parameters: None

Description: This command will stop the playback of any tone on the speaker/buzzer.

Example: To stop tone playback send the following SysEx command bytes:

0x40 0x21

Accelerometer Commands

Single Accelerometer Reading

Sub-command byte: 0x30

Parameters: None

Description: This command will request a single accelerometer reading. The result of this command will be an accelerometer read response command sent back from the Circuit Playground board to the host computer. See the description of the accelerometer read response below for what data to expect.

Example: To request a single accelerometer reading send the following SysEx bytes:

0x40 0x30

Accelerometer Read Response

Sub-command byte: 0x36 0x00 (Note this sub-command has **two bytes** instead of one byte as a side-effect of using internal Firmata sketch SysEx functions)

Parameters: 24 bytes total

- **X Axis Acceleration (8 bytes)**: These bytes define a 32-bit floating point value in little endian byte order (least significant bytes first). Each pair of 7-bit bytes defines an 8-bit byte in the float so 8 bytes total will give 4 bytes of floating point data (32 bits total). For these pairs of 7-bit bytes they are in little endian order with the 7 lowest bits first and the 8th/high order bit in the second position. This parameter is the X axis acceleration in meters per second squared.
- **Y Axis Acceleration (8 bytes)**: These bytes define a 32-bit floating point value in little endian byte order (least significant bytes first). Each pair of 7-bit bytes defines an 8-bit byte in the float so 8 bytes total will give 4 bytes of floating point data (32 bits total). For these pairs of 7-bit bytes they are in little endian order with the 7 lowest bits first and the 8th/high order bit in the second position. This parameter is the Y axis acceleration in meters per second squared.
- **Z Axis Acceleration (8 bytes)**: These bytes define a 32-bit floating point value in little endian byte order (least significant bytes first). Each pair of 7-bit bytes defines an 8-bit byte in the float so 8 bytes total will give 4 bytes of floating point data (32 bits total). For these pairs of 7-bit bytes they are in little endian order with the 7 lowest bits first and the 8th/high order bit in the second position. This parameter is the Z axis acceleration in meters per second squared.

Description: This command is sent back from the Circuit Playground board to the host computer when an accelerometer reading is available. The current X, Y, Z acceleration is returned in meters per second squared units.

Example: An accelerometer reading of X = 9.8, Y = -1.0, Z = 0.0 will have the following SysEx command bytes:

0x40 0x36 0x00 0x41 0x00 0x1C 0x00 0x4C 0x01 0x4D 0x01 0x3F 0x01 0x00 0x01 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

Enable Accelerometer Streaming

Sub-command byte: 0x3A

Parameters: None

Description: This command will turn on continuous streaming of accelerometer readings from Circuit Playground to the host computer. Reading will be taken every 20 milliseconds and sent to the host using the accelerometer read response command above. This is useful for quickly reading the accelerometer without all the overhead of constantly sending single accelerometer read requests. To disable the streaming see the disable accelerometer streaming function below.

Example: To enable steaming of accelerometer data send the following SysEx bytes:

0x40 0x3A

Disable Accelerometer Streaming

Sub-command byte: 0x3B

Parameters: None

Description: This command will turn off continuous streaming of accelerometer readings from Circuit Playground to the host computer.

Example: To disable steaming of accelerometer data send the following SysEx bytes:

0x40 0x3B

Set Accelerometer Range

Sub-command byte: 0x3C

Parameters: 1 byte

- **Range (1 byte):** This value defines the range of the accelerometer and should be one of the following values:
  - **0** = +/- 2G
  - **1** = +/- 4G
  - **2** = +/- 8G
  - **3** = +/- 16G

Description: This command will change the range of the accelerometer. Increasing the range will allow you to read higher forces, but will reduce the accuracy of the readings. The default range is +/- 2G.

Example: To set the range of the accelerometer to +/- 16G send the following SysEx bytes:

0x40 0x3C 0x03

Tap Detection Commands

Single Tap Detection Reading

Sub-command byte: 0x31

Parameters: None

Description: This command will request a single tap detection reading. The result of this command will be a tap detection response command sent back from the Circuit Playground board to the host computer. See the description of the tap detection response below for what data to expect.

Example: To request a single tap detection reading send the following SysEx bytes:

0x40 0x31

Tap Detection Response

Sub-command byte: 0x37 0x00 (Note this sub-command has **two bytes** instead of one byte as a side-effect of using internal Firmata sketch SysEx functions)

Parameters: 2 bytes

- **Tap Detection Register Value (2 bytes):** These two bytes define an unsigned 8-bit value that is the current tap detection register value. The bytes are in little endian order so the low 7-bits are in the first byte and the 8th/high order bit is in the second byte. See the LIS3DH accelerometer datasheet (https://adafru.it/ncQ) for the exact meaning, however a double tap will have bit 6 set and a single tap will have bit 5 set.

Description: This command is sent back from the Circuit Playground board to the host computer when a tap detection reading is available. The current tap detection register value is returned.

Example: A double tap detection event would respond with the following SysEx bytes:

0x40 0x37 0x00 0x20 0x00

Enable Tap Detection Streaming

Sub-command byte: 0x38

Parameters: None

Description: This command will turn on continuous streaming of tap detection readings from Circuit Playground to the host computer. Reading will be taken every 20 milliseconds and sent to the host using the tap detection response command above. This is useful for quickly reading the tap detection state without all the overhead of constantly sending single tap detection read requests. To disable the streaming see the disable tap detection streaming function below.

Example: To enable steaming of tap detection data send the following SysEx bytes:

0x40 0x38

Disable Tap Detection Streaming

Sub-command byte: 0x39

Parameters: None

Description: This command will turn off continuous streaming of tap detection readings from Circuit Playground to the host computer.

Example: To disable steaming of tap detection data send the following SysEx bytes:

0x40 0x39

Set Tap Detection Configuration

Sub-command byte: 0x3D

Parameters: 4 bytes

- **Type (2 bytes):** These two bytes define an unsigned 8-bit value which identifies the type of tap detection to perform. The lower 7 bits of the value are in the first byte and the 8th/high order bit are in the second byte. The following values are allowed:
  - **0** = no tap detection
  - **1** = single tap detection only
  - **2** = single & double tap detection (default)
- **Tap Threshold (2 bytes):** These two bytes define an unsigned 8-bit value (0-255) which is the threshold for tap detection. The higher the value the less sensitive the tap detection. Generally the threshold should be set depending on the accelerometer range and good values to try are:
  - 5-10 for +/- 16G range
  - 10-20 for +/- 8G range
  - 20-40 for +/- 4G range
  - 40-80 for +/- 2G range (default is 80)

Description: This command will change the type and sensitivity of tap detection.

Example: To configure tap detection for only single clicks with a threshold of 40 send the following SysEx bytes:

0x40 0x3D 0x01 0x00 0x28 0x00

Capacitive Touch Commands

Single Capacitive Touch Reading

Sub-command byte: 0x40

Parameters: 1 byte

- **Input number (1 byte):** This byte specified which capacitive touch input to read and should be one of: 0, 1, 2, 3, 6, 9, 10, 12

Description: This command will request a single capacitive touch reading. The result of this command will be a capacitive touch response command sent back from the Circuit Playground board to the host computer. See the description of the capacitive touch response below for what data to expect.

Example: To request a single capacitive touch reading for input 12 send the following SysEx bytes:

0x40 0x40 0x0C

Capacitive Touch Response

Sub-command byte: 0x43 0x00 (Note this sub-command has **two bytes** instead of one byte as a side-effect of using internal Firmata sketch SysEx functions)

Parameters: 10 bytes

- **Input number (2 bytes):** These bytes define an unsigned 8-bit value which is the input number that is associated with this response (0, 1, 2, 3, 6, 9, 12). The bytes are in little endian order with the 7 low bits first and the 8th/high order bit last (note this bit will never really be set since the inputs only go up to 12).
- **Raw capacitive touch value (8 bytes):** These eight bytes define an unsigned 32-bit value which is the raw capacitive touch reading for this input. The bytes are in little endian order with each pair of bytes representing a single byte in the reading (the 7 low bits are first, then the 8th/high order bit follows). The higher the value the larger the capacitance touching the input. You can apply your own threshold to consider an input 'pressed', for example values above 300 or so are a good reading to detect a press.

Description: This command is sent back from the Circuit Playground board to the host computer when a capacitive input reading is available.

Example: A response for input 12 with a capacitive touch reading of 316 is made of the following SysEx bytes:

0x40 0x43 0x00 0x0C 0x00 0x3C 0x00 0x01 0x00 0x00 0x00 0x00 0x00

Enable Capacitive Touch Streaming

Sub-command byte: 0x41

Parameters: 1 byte

- **Input number (1 byte):** This byte specified which capacitive touch input to enable streaming and should be one of: 0, 1, 2, 3, 6, 9, 10, 12

Description: This command will turn on continuous streaming of capacitive touch readings from Circuit Playground to the host computer. Reading will be taken every 20 milliseconds and sent to the host using the capacitive touch response command above. This is useful for quickly reading the capacitive touch state without all the overhead of constantly sending single capacitive touch read requests. To disable the streaming see the disable capacitive touch streaming function below.

Example: To enable steaming of capacitive touch for input 12 send the following SysEx bytes:

0x40 0x41 0x0C

Disable Capacitive Touch Streaming

Sub-command byte: 0x42

Parameters: 1 byte

- **Input number (1 byte):** This byte specified which capacitive touch input to disable streaming and should be one of: 0, 1, 2, 3, 6, 9, 10, 12

Description: This command will turn off continuous streaming of capacitive touch readings from Circuit Playground to the host computer.

Example: To disable steaming of capacitive touch for input 12 send the following SysEx bytes:

0x40 0x42 0x0C

Color Sensing Commands

These commands use the light sensor and NeoPixel #1 (adjacent to the light sensor) to perform basic color detection of an object in front of the sensor. See this video for more details and examples of the color sensing logic: https://www.youtube.com/watch?v=30NKAKwgkIM (https://adafru.it/oB5)

Single Color Sense Command

Sub-command byte: 0x50

Parameters: None

Description: This command will perform a single color sense using the light sensor and NeoPixel #1. The result will be returned in a new color sense command response (see below).

Example: To request a color sense send the following SysEx bytes:

0x40 0x50

Color Sense Response

Sub-command byte: 0x51 0x00 (Note this sub-command sends **two bytes** instead of one)

Parameters: 6 bytes

- **Red component value (2 bytes):** These two bytes define the unsigned 8-bit value for the red component of the detected color. The first byte is the 7 low order bits and the second byte is the 8th/high order bit.
- **Green component value (2 bytes):** These two bytes define the unsigned 8-bit value for the green component of the detected color. The first byte is the 7 low order bits and the second byte is the 8th/high order bit.

- **Blue component value (2 bytes):** These two bytes define the unsigned 8-bit value for the blue component of the detected color. The first byte is the 7 low order bits and the second byte is the 8th/high order bit.

Description: This response is sent from Circuit Playground to the host when a color sense result is available. The parameters include the red, green, blue component values of the detected color (0 is minimum intensity and 255 is maximum intensity).

Example: A color sense response with red=255, green=128, and blue=0 would send the following SysEx bytes:

0x40 0x51 0x00 0x7F 0x01 0x00 0x01 0x00 0x00

Other Components

For other components on the Circuit Playground board you can use existing Firmata functions to read them:

- **Light Sensor:** The light sensor is connected to **analog input 5** and can be read with Firmata's standard analog input functions. The value of the light sensor has no units and increases as more light is visible.
- **Temperature Sensor:** The temperature sensor is a thermistor connected to **analog input 0** and can be read with Firmata's standard analog input functions. You probably want to apply Steinhart-Hart thermistor equations to the output in order to convert it to a temperature. See how the Python Circuit Playground Firmata client does this conversion (https://adafru.it/Cgl) for more details.
- **Microphone:** The microphone is connected to **analog input 4** and can be read with Firmata's standard analog input functions. The value of these microphone readings are raw analog samples and must be averaged or smoothed out to understand the volume. You can also take a set of sample and apply a FFT for example to find the frequencies of sound being heard.
- **Left Pushbutton:** The left pushbutton is connected to **digital input 4** and can be read with Firmata's standard digital input functions. When the button is pressed it will be pulled to a high logic level, and when not pressed it will be pulled to a low/ground logic level.
- **Right Pushbutton:** The right pushbutton is connected to **digital input 19** and can be read with Firmata's standard digital input functions. When the button is pressed it will be pulled to a high logic level, and when not pressed it will be pulled to a low/ground logic level.
- **Slide Switch:** The slide switch is connected to **digital input 21** and can be read with Firmata's standard digital input functions. When the switch is pulled to the

left it will read a high logic level, and when pulled to the right it will read a low/ground logic level.