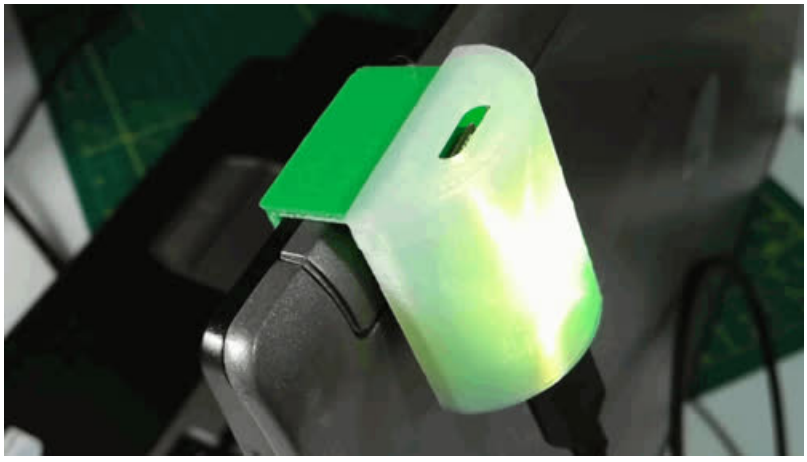
adafruit learning system

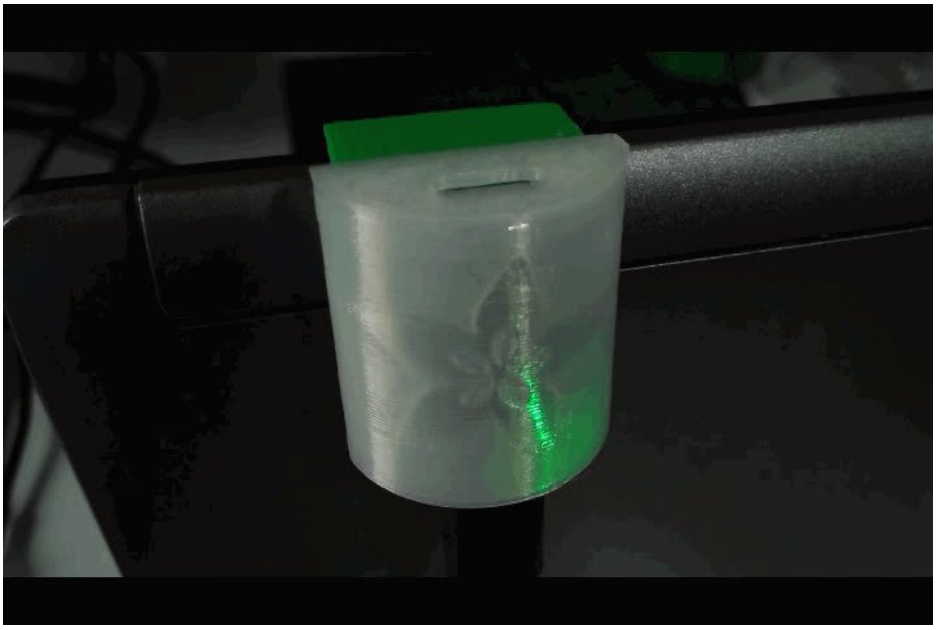
CircuitPython Powered AT Hand-Raiser

Created by Bill Binko



Last updated on 2019-06-25 08:32:23 PM UTC

Overview



Sometimes folks need help doing the simple things that most of us take for granted - that's where Assistive Technology comes into play. For example, ATMakers recently had a college student reach out and ask for help getting her professor's attention in class.

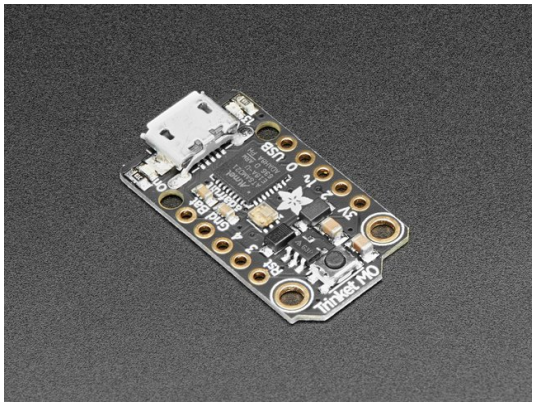
The problem was she has Spinal Muscular Atrophy (SMA) and couldn't raise her hand. So, we created this small Trinket-powered device that attaches to her laptop or tablet and lights up when she wants to participate!

Along the way, we found a fantastic way to control CircuitPython code using nothing more than the USB cord and Windows, Mac or Linux scripts loaded on the Trinket itself.

Parts List

This project requires just two parts and some 3D printing filament. You'll need a Trinket M0 microcontroller and a good USB Cable. Make sure you've got a data cable, not a charge-only cable, as this project will send data over the USB interface.

For the filament, you'll need a translucent style that allows light to pass through the faceplate. The Natural Translucent PLA listed below is great. For the other pieces, you can use the same filament or something opaque like the "ATMakers Green" filament we chose.



Adafruit Trinket M0 - for use with CircuitPython & Arduino IDE

$8.95
IN STOCK

ADD TO CART

USB cable - USB A to Micro-B

$2.95
IN STOCK

ADD TO CART

PLA Filament for 3D Printers - 1.75mm Natural Translucent - 1KG

OUT OF STOCK

OUT OF STOCK

PLA Filament for 3D Printers - 3mm Diameter - Green - 1KG

OUT OF STOCK

OUT OF STOCK

Printing the Enclosure

The CircuitPython code controls the on-board RGB LED, but without an enclosure that diffuses that light, it doesn't really grab your attention. So, we created a three-piece snap-fit case that holds the Trinket and allows it to hang over the edge of our user's laptop or tablet screen. The case has a translucent front cover that spreads out the light and makes a nice large lighted volume that's easy to see from across the room.

This enclosure is a "snap fit" design, based on the approach that Noe & Pedro Ruiz shared intheir great video last year (https://adafru.it/CS4). This design leverages those ideas and creates a design that is easy to print, requires no supports, and can be assembled in seconds, all with no tools.

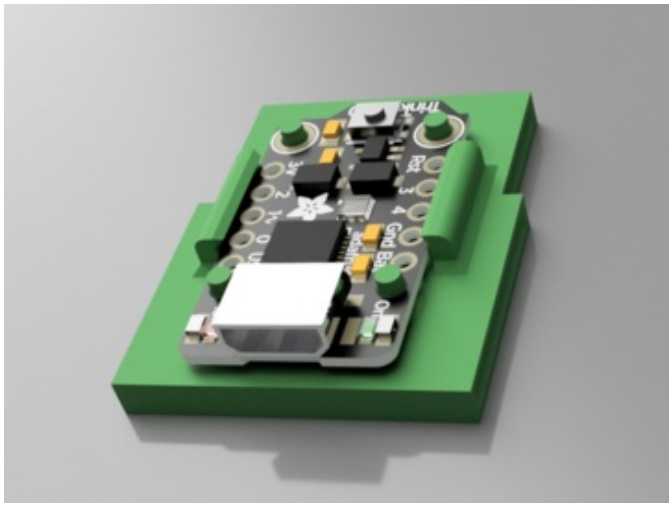
You will need to download three STL files: a backplate for the Trinket, a hanging hook, and a faceplate that must be printed in translucent filament. There are several versions of the hook and faceplate, you only need one:

https://adafru.it/CS5

https://adafru.it/CS5

https://adafru.it/CS6

https://adafru.it/CS6

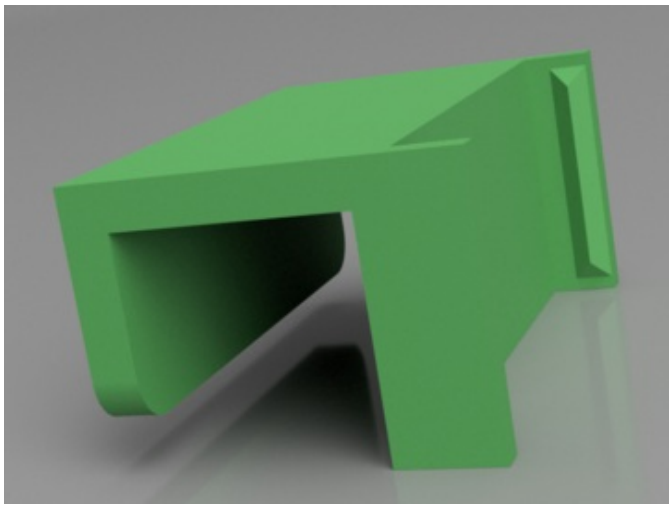


Back Plate

Filename: Snap_Back.stl

This part holds the Trinket in place. It's specifically designed to hold the board in place with no screws or tools.

There is only one backplate design, and you can print it in any color you'd like. We've used ATMakers Green.

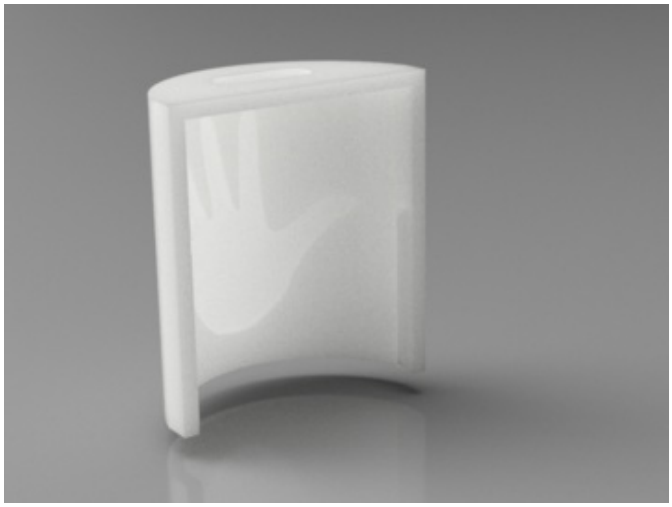


Hanging Hook

Filename: Snap_Hanger.stl

The hook snaps onto the backplate and holds the enclosure onto the screen of a laptop or tablet. This part is configurable in Fusion 360 in the User Parameters screen. You can set the width of the screen you're mounting the device on and how far you'd like the tab to hang down.

The default is 10mm wide, but if you're not up to editing the file, we've included two alternative widths: 15mm, and 20mm. You only need to print one of the files that start with **Snap_Hanger**.



Faceplate

Filename: Faceplate-Hand.stl

The faceplate snaps over both of the other two parts and must be printed in filament that allows light through such as a natural translucent PLA. (https://adafru.it/uuF)

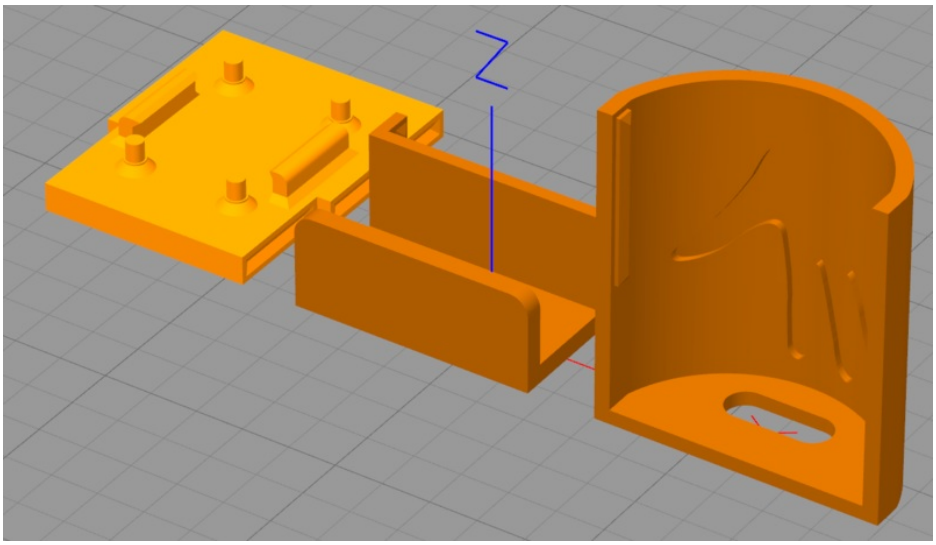
You only need to print one faceplate - there are four available:

- Handprint
- Adafruit Logo
- ATMakers.org Logo
- and Blank

Printing Tips

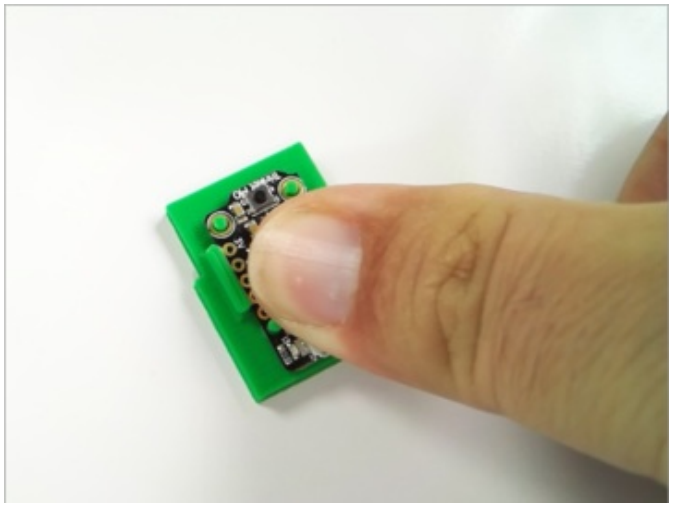
- **Supports:** All three parts print easily with out any supports, however your software may try to add them on the faceplate. You should disable supports so that they don't make marks on the translucent surface when they break apart.
- **Filament:** For best results, print the parts in similar materials. That is, if you're using a translucent PLA for the faceplate, try to print the other parts in a similar PLA rather than mixing PLA and ABS, etc. We have had excellent results with many styles of PLA and PETG with this design.
- **Resolution:** For the snap-fit to work best, a layer height of .200mm or finer is recommended. These parts do not take long to print and the result is worth the wait.

- **Orientation:** For easiest printing, orient the parts as seen below



Assembly

Assembling the HandRaiser light is quite simple and requires no tools. Just pop the Trinket into the backplate and snap on the hanger and the cover.



Firmly press the Trinket M0 onto the backplate.

The board should be oriented so that the USB port is at the bottom of the backplate. The top of the backplate is thinner so that the hanger can snap on - that is where the reset button should go.

> Note: The fit can be snug, and you may have to press one side and then the other, but when complete the board should be secure.

Snap on the Hanger

While there are several Hanger sizes, they all attach the same way.

Position the hanger as shown and firmly press the snap-fit grooves together.

Choose and Attach Your Cover

We've provide four different covers you can choose: an ATMakers logo, an Adafruit Logo, a hand, and a blank cover.

Once you choose and print the cover, it's a breeze to snap it over the Trinket and onto the Backplate.

That's all there is to it! Now it's time to program the Trinket and let our users get their teacher's attention.

Programming the Trinket

To get you started quickly, we've created a single file that contains everything you'll need to program your Trinket M0 and turning it into a HandRaiser. Once that's working you can adjust and customize your device to suit your needs.

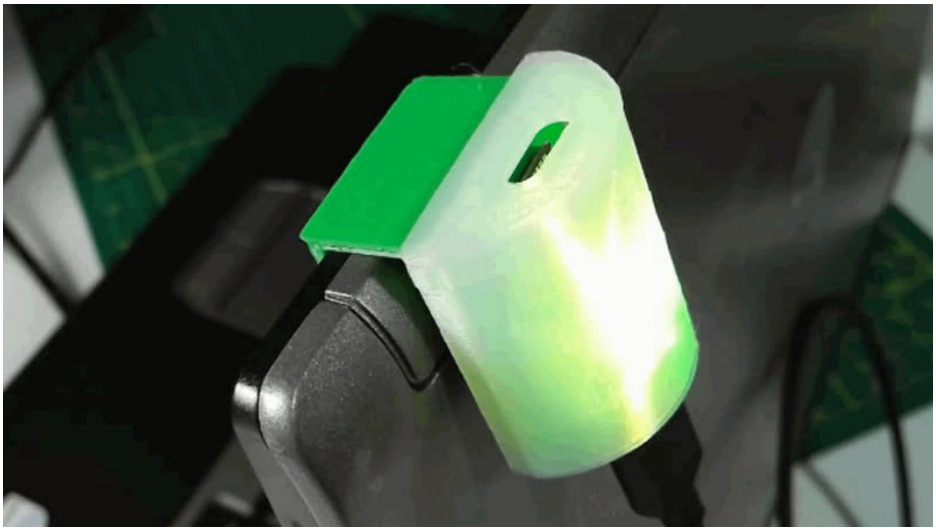
https://adafru.it/CXr

https://adafru.it/CXr

The **HandiRaiser.UF2** file is a complete image that will program the Trinket and load the support files all in one shot. To load the UF2 file:

1. Plug in the Trinket using the (known-good) Power+Data USB cord
2. Place the Trinket in Bootloader Mode by double-tapping the Reset Button (https://adafru.it/CXs). (The RGB LED on the Trinket will blink green)
3. Locate the new drive (named **TRINKETBOOT**) that is created in Bootloader Mode (usually named D:\, E:\ or another letter on Windows). You will know you have the right drive if you see a file named **CURRENT.UF2**
4. Copy the **HandRaiser.UF2** file to that drive. The Trinket will restart and you should see a new drive named **CIRCUITPY.**

Once the process is complete, your HandRaiser should reset and should go into a slowly changing "Wheel" color pattern that goes through all the colors of the spectrum.



Next, let's set this up so that our users can choose the color and pattern they want directly from any computer with a USB port.

HandRaiser Commands

The HandRaiser works by listening to incoming requests over it's USB Serial connection. That allows any PC, Mac, etc. with a USB Serial (UART) connection to control the device.

For Michelle, that controlling device was her Tobii Speech Generating Device (which is based on a Windows Tablet), but it could just as easily be a Raspberry Pi, Mac, Android or anything with a USB port.

We will share details about how to connect via Windows and Linux later in the guide, but first, let's show what commands are available on the HandRaiser and what they do.

Shortcuts - Just 4 Colors

This project started with a very simple request: the ability to set her color to red, yellow, green or black. So, those names are special in this project: they simply set the color and make sure the'll stay on that color by switching out of the "wheel" mode. Other modes like "blink" and "ramp" continue to work, the color hue just changes.

Any Color - #RRGGBB

If you'd like to set any other color, you can send a Hex code, just like in HTML. The Hex code starts with a hashtag, then two letters for Red, Green, and Blue in that order. Just like the standard colors, modes stay the same except for "wheel" which switches to solid.

There are many Hex Code calculators and tools out there, including this one from W3Schools. (https://adafru.it/DwD)

Changing Brightness - %0 - %100

To change the brightness of the LED without changing the mode or base color, just send a percent symbol followed by a number between 0 and 100.

Changing Mode - @modename

Messages that start with an @ symbol trigger a change of mode. Currently there are five modes that change how the HandRaiser's colors change:

- **solid -** Show the target color with no variation or changes.
- **blink -** Alternate between the target color and black every half second (or so)
- **ramp -** Slowly fade between the target color and black every two seconds (or so)
- **beat -** Change the brightness of the target color to match that of a standard heartbeat rythm (experimental)
- **wheel -** Rotate the color of the LED around a color wheel.

Examples

Here are some examples:

```
green
@blink
%50
```

Set the color to red, blinking, and 1/2 full brightness

```
#990099
%90
@ramp
```

Set the color to dark purple, 90% brightness and a slow ramp from black to purple and back.

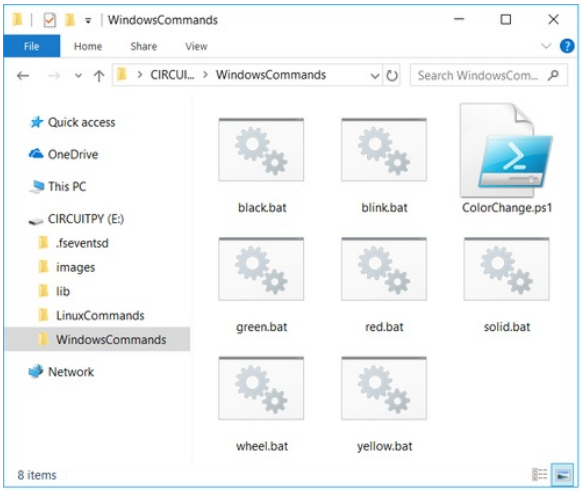
```
#FF0000
%100
@beat
```

Make the HandRaiser follow a heartbeat pattern in red with full brightness.

Windows Shortcuts

The HandRaiser includes a set of Windows Shortcuts that can be used to change the LED just by double clicking them. You'll find them in the **WindowsCommands** folder on the **CIRCUITPY** drive that's created when you plug in the HandRaiser.

These scripts use a language that many of us aren't very familiar with, but that is installed on every modern Windows machine: PowerShell!



The folder has a PowerShell script named "ColorChange.ps1" that connects to the Serial Port on the HandRaiser and sends a message. It also has .BAT files that simply call that file with different messages.

For example, the "red.bat" file has the following text:

```
powershell -executionpolicy bypass -File "ColorChange.ps1" "red"
```

When executed, it calls ColorChange.ps1 and request that it send the code "red" to the Trinket. (the "-executionpolicy bypass" is necessary to run unsigned scripts from the command line)

Let's take a look at the ColorChange.ps1 file and how it works.

```
function sendColor($colorName)
{
    $portInfo = ( Get-WmiObject Win32_SerialPort | Where { $_.PNPDeviceID -like '*VID_239A*' } | select -
    $port = new-Object System.IO.Ports.SerialPort $portInfo.DeviceID,9600,None,8,one
    $port.open()
    $text = $colorName + "`r"
    $port.Write($text)
    start-sleep -m 50
    $port.ReadExisting()
    $port.Close()
}
if ($args.Length -eq 0)
{
    echo "Usage: ColorChange <color>"
}
else
{
    sendColor($args[0])
}
```

This PowerShell script has a single function sendColor() that takes a string and sends it cleanly to the Trinket over the Serial connection. The critical section begins on line 3:

```
    $portInfo = ( Get-WmiObject Win32_SerialPort | Where { $_.PNPDeviceID -like '*VID_239A*' } | select -
    $port = new-Object System.IO.Ports.SerialPort $portInfo.DeviceID,9600,None,8,one
    $port.open()
```

Here, PowerShell requests information about the Serial Ports and filters that down to only return those with a Plug and Play DeviceID that includes "VID_239A" which is Adafruit's identifier. It then selects only the last device (the most recently added device).

This means that if you're connecting multiple Adafruit devices (perhaps a Trinket, Circuit Playground and a Feather) it will always connect to the most recently added device with a Serial Port.

The next line creates a new SerialPort object connecting at 9600 baud and the standard "N-8-1" parity settings.

Then the port is opened so that we can read and write to it.

```
    $text = $colorName + "`r"
    $port.Write($text)
```

Next we add a carriage return character to the end of any text passed in. This is because the CircuitPython code running on the Trinket uses that to recognize the end of a a command.

And we use the **Write()** method to send the command and carriage return to the serial port.

```
    start-sleep -m 50
    $port.ReadExisting()
    $port.Close()
```

These lines are necessary to get the Trinket to stop listening for the command to perform the action. A short delay (50ms) is added and then the port is read before closing it.

This is a limitation of the **input()** method in CircuitPython and you may have to increase or decrease your delay to match your device. However, we've found the 50ms delay is adequate for the Trinket M0 we're using.

Linux/Mac Scripts

For those running Mac OS/X, Linux or using a Raspberry Pi, shortcuts like those for Windows are less useful.

Using serial interface programs like Minicom or Screen are great and work wonderfully, however, a simple script that sends commands to the HandRaiser are certainly helpful.

Like on Windows, we'll use a scripting language that is on every major Linux release and in Mac OS/X: the Bash shell.

in the Scripts folder, you'll find a script named **colorChange.sh**. Let's take a look at it.

```
#!/bin/bash
# The line above has to be bash (not sh) because we're using the printf builtin

#before we start, make sure we know whether we're under Linux or OS/X (Darwin)
uname -s | grep -i Linux > /dev/null
ISLINUX=$?

#$ISLINUX will be 0 (true) if we're on linux and 1 (false) if we're on mac

#First, get the name of the most recent Adafruit serial device connected
#Conveniently linux helps us by grouping them by manufacturer id
#Mac doesn't so we just take the most recently added serial device overall

if [ "$ISLINUX" -eq 0 ]; then

    DEVNAME=`ls -1t /dev/serial/by-id/ | grep Adafruit | head -1`

    if [[ -z "$DEVNAME" ]]; then
        echo "No Adafruit Device Found"
        exit
    fi

    FULLDEVPATH=/dev/serial/by-id/$DEVNAME
else
    DEVNAME=`ls -1t /dev/cu.usb* | head -1`
    if [[ -z "$DEVNAME" ]]; then
        echo "No Serial Device Found"
        exit
    fi
    FULLDEVPATH=$DEVNAME
fi

#Comment this to remove debug text
echo "Found recent Adafruit device at $FULLDEVPATH"

text=$@

if [[ -z "$text" ]]; then
    echo "Usage: colorChange.sh <Text To Send>"
    exit
fi

#Comment this to remove debug text
echo "Sending text '$text' to serial device"

printf "%b\r" $text > $FULLDEVPATH
#for some reason input() needs to have the serial port READ to return on linux
if [ "$ISLINUX" -eq 0 ]; then
 head -1 $FULLDEVPATH > /dev/null
fi
```

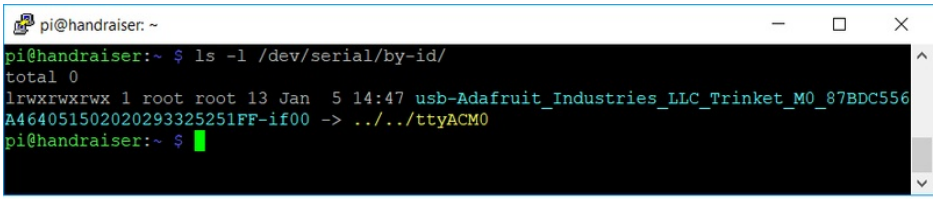
First, we need to find the most recently added Adafruit device with a serial port (just like we did on Windows). How we do this differs depending on whether we're on Linux or Mac OS/X. We determine that by running the command:

```
uname -s | grep -i Linux
```

and saving the return code in the $ISLINUX variable.

Detecting Boards on Linux

Thankfully, Linux makes this easy by already grouping all of the serial port in a folder named **/dev/serial/by-id/.** Here's a listing:



As you can see with the HandRaiser attached, there is an entry for a serial device that has an id that includes **Adafruit_Industries_LLC_Trinket_M0**. We'll use this to find the correct device.

```
DEVNAME=`ls -1t /dev/serial/by-id/ | grep Adafruit | head -1`
```

This command calls the ls command with the -1t option asking for a simple name output in order of modification, sends the output to grep to look for the word "Adafruit" and then takes just the first line by using "head -1". results are stored in the **DEVNAME** variable using backticks (`)

If we don't find one, we exit with an error message.

We build the full pathname to the serial device using this command:

```
FULLDEVPATH=/dev/serial/by-id/$DEVNAME
```

Detecting Boards under OS/X

If we're on Mac OS/X, we don't have the list by device id, so we just use the list of all serial devices which can be found under **/dev/cu.usb*** and follow a similar path to determine the **FULLPATH**

```
DEVNAME=`ls -1t /dev/cu.usb* | head -1`
```

In this case, the **FULLDEVPATH** is actually returned directly:

```
FULLDEVPATH=$DEVNAME
```

Sending the Text using Printf

Next, we read the text from the command line using this line:

```
text=$@
```

That stores the command line arguments (which bash puts in $@) into a variable named **text**. If we don't get any we return with an error.

Finally we need to send the command with a carriage return and read from the serial port to make the Trinket process the command. All of that can be done using these two lines:

```
printf "%q\r" "$text" > $FULLDEVPATH
head -1 $FULLDEVPATH > /dev/null
```

The **printf** call is a bash built-in function that lets you specify formatting characters like the carriage return (\r) easily and to pass through special characters like "%" using the %q option. We send the output directly to the serial port using the redirection operator (>)

The **head** command reads a single line from the serial port and sends it to**/dev/null** (basically this just ignores the result).

Using the colorChange.sh Script

To use the **colorChange.sh** script, copy it to your Linux machine and make sure you have permissions to access the serial port. Kattni's guide on using the Serial port can really help here.

Once installed you can use the script like this:

```
./colorChange.sh "red"
```

Set the color to **red**

```
./colorChange.sh "%50"
```

Set the brightness to 50% (note the quotes are required here!)

```
./colorChange.sh "@wheel"
```

Set the device to rotate through the colors

```
./colorChange.sh "#AA00AA"
```

Set the color to **dark purple**.

CircuitPython Code

The HandRaiser uses CircuitPython to listen for commands on its USB Serial connection and respond by changing its LED's color accordingly.

Let's walk through the code showing how this works:

```
import board
import adafruit_dotstar
from time import sleep

import supervisor
```

First we need to import some modules that we'll use. Most of these are common: **board** includes definitions that are specific to the Trinket M0 like pin names, **adafruit_dotstar** lets use control the single color LED and the **sleep()** function lets us add delays to our loop.

A less common addition is the **supervisor** module. It provides information and control of the Python environment itself. In this case, we want to ask CircuitPython if there is any text available to read from the USB Serial connection (without blocking). Thankfully, there's an attribute (**runtime.serial_bytes_available)** on the **supervisor** module r**untime** object that lets us check that.

```
def hex2rgb(hex_code):
    red = int("0x"+hex_code[0:2], 16)
    green = int("0x"+hex_code[2:4], 16)
    blue = int("0x"+hex_code[4:6], 16)
    rgb = (red, green, blue)
    return rgb
```

This helper function, **hex2rgb()** is handy to convert Web-style hex color codes to RGB tuples. This lets the controlling computer send commands like "#550000" for dark red or "#FFFF00" for bright yellow.

```
beatArray = [0.090909091,0.097902098,0.104895105,0.118881119,0.132867133,0.146853147,
    0.153846154........
```

The **beatArray variable holds an array of sampled values that represent a heartbeat. It supports the "@beat" mode.**

```
# When we start up, make the LED black
black = (0, 0, 0)
# the color that's passed in over the text input
targetColor = black

#pos is used for all modes that cycle or progress
#it loops from 0-255 and starts over
pos = 0

#curColor is the color that will be displayed at the end of the main loop
#it is mapped using pos according to the mode
curColor = black
```

Here we set the initial values for the current and target color, position of the color wheel and make a constant color **black** that we'll use for turning off the LED.

```
# the mode can be one of
# solid - just keep the current color
# blink - alternate between black and curColor
# ramp  - transition continuously between black and curColor
# beat - pulse to a recorded heartbeat intensity
# wheel - change hue around the colorwheel (curColor is ignored)

mode='wheel'
```

The **mode** variable keeps track of which type of animation we're running. We'll use it to change the LED color on each pass through the main loop.

```
# standard function to rotate around the colorwheel
def wheel(cpos):
    # Input a value 0 to 255 to get a color value.
    # The colours are a transition r - g - b - back to r.
    if cpos < 85:
        return (int(cpos * 3), int(255 - (cpos * 3)), 0)
    elif cpos < 170:
        cpos -= 85
        return (int(255 - (cpos * 3)), 0, int(cpos * 3))
    else:
        cpos -= 170
        return (0, int(cpos * 3), int(255 - cpos * 3))

# We start by turning off pixels
pixels.fill(black)
pixels.show()
```

This code is common to many "blinky" projects from Adafruit - it cycles through the color wheel making a lovely rainbow pattern.

```
# Main Loop
while True:
    # Check to see if there's input available (requires CP 4.0 Alpha)
    if supervisor.runtime.serial_bytes_available:
        # read in text (@mode, #RRGGBB, %brightness, standard color)
        # input() will block until a newline is sent
        inText = input().strip()
        # Sometimes Windows sends an extra (or missing) newline - ignore them
        if inText == "":
            continue
```

Once the setup is complete, we have our main loop - this will continue forever, checking for new messages from the controller and changing the LED.

First, it checks with the CircuitPython Supervisor to see if there is any text to read from the USB port. If so, it calls **input()** to read the message (and then strips off any whitespace like newlines or spaces).

Note that the input() function will block until it gets a carriage return (\r) and something reads the Serial port from the other side. That's why the serial_bytes_available attribute is important!

```
# Process the input text - start with the presets (no #,@,etc)
        # We use startswith to not have to worry about CR vs CR+LF differences
        if inText.lower().startswith("red"):
            # set the target color to red
            targetColor = (255, 0, 0)
            # and set the mode to solid if we're in a mode that ignores targetColor
            if mode == "wheel":
                mode="solid"
# Repeat for green, yellow and black...
```

Now that we're processing text, look for standard colors: **red, green, yellow,** and **black**. We set an appropriate color and make the **mode** "solid" if it was set to "wheel".

```
# Here we're going to change the mode - which starts w/@
        elif inText.lower().startswith("@"):
            mode= inText[1:]
```

If the incoming message starts with an "@" symbol, change the **mode** to the incoming text. Valid values of **mode** are "solid", "blink", "ramp", "beat", and "wheel".

```
# Here we can set the brightness with a "%" symbol
        elif inText.startswith("%"):
            pctText = inText[1:]
            pct = float(pctText)/100.0
            pixels.brightness=pct
```

If the controller sends a message starting with "%" take the number after it as a brightness level (0-100).

```
# If we get a hex code set it and go to solid
        elif inText.startswith("#"):
            hexcode = inText[1:]
            targetColor = hex2rgb(hexcode)
            if (mode == "wheel"):
                mode="solid"
```

Here we use the **hex2rgb()** function to convert any incoming #RRGGBB values into a valid color tuple before setting the color.

```
# if we get a command we don't understand, set it to gray
        #we should probably just ignore it but this helps debug
        else:
            targetColor =(50, 50, 50)
            if (mode == "wheel"):
                mode="solid"
```

If we get a command but don't understand it, just set the color to medium gray so we have an indicator something's

gone wrong.

```
else:
        #If no text availble, update the color according to the mode
        if mode == 'blink':
            if curColor == black:
                curColor = targetColor
            else:
                curColor = black
            sleep(.4)
        #        print('.', end='')
            pixels.fill(curColor)
            pixels.show()
        elif mode == 'wheel':
            sleep(.05)
            pos = (pos + 1) % 255
            pixels.fill(wheel(pos))
            pixels.show()
        elif mode == 'solid':
            pixels.fill(targetColor)
            pixels.show()
        elif mode == 'beat':
            pos = (pos + 5 ) % 106
            scaleAvg = (beatArray[(pos-2)%106] + beatArray[(pos-1)%106] + beatArray[pos] + beatArray[(pos
            beatColor = tuple(int(scaleAvg*x) for x in targetColor)
            pixels.fill(beatColor)
            sleep(.025)
            pixels.show()
        elif mode == 'ramp':
            pos = ((pos + 5 ) % 255)
            scaleFactor = (2*abs(pos-127))/255
            beatColor = tuple(int(scaleFactor * x) for x in targetColor)
            pixels.fill(beatColor)
            sleep(.075)
            pixels.show()
```

At this point, we know that we **do not** have an incoming message. So, we just want to update the color of the LED based on the current mode, **sleep()** if appropriate and continue the loop to check for the message again.

Each of the **If/Elif** sections covers a different mode and simply calculates the next color, sets it and sleeps. There are lots of ways to modify these color modes or to add your own!

Complete Code

```
# ATMakers HandUp
# Listens to the USB Serial port and responds to incoming strings
# Sets appropriate colors on the DotStar LED

# This program uses the board package to access the Trinket's pin names
# and uses adafruit_dotstar to talk to the LED
# other boards would use the neopixel library instead

from time import sleep
import board
import adafruit_dotstar
```

```
import supervisor

# create an object for the dotstar pixel on the Trinket M0
# It's an array because it's a sequence of one pixel
pixels = adafruit_dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1, brightness=.95)

# this function takes a standard "hex code" for a color and returns
# a tuple of (red, green, blue)
def hex2rgb(hex_code):
    red = int("0x"+hex_code[0:2], 16)
    green = int("0x"+hex_code[2:4], 16)
    blue = int("0x"+hex_code[4:6], 16)
    rgb = (red, green, blue)
    # print(rgb)
    return rgb

# This array contains digitized data for a heartbeat wave scaled to between 0 and 1.0
# It is used to create the "beat" mode. Ensure each line is <78 characters for Travis-CI
beatArray = [0.090909091,0.097902098,0.104895105,0.118881119,0.132867133,0.146853147,
             0.153846154,0.160839161,0.181818182,0.181818182,0.195804196,0.181818182,0.188811189,
             0.188811189,0.181818182,0.174825175,0.174825175,0.160839161,0.167832168,0.160839161,
             0.167832168,0.167832168,0.167832168,0.160839161,0.146853147,0.146853147,0.153846154,
             0.160839161,0.146853147,0.153846154,0.13986014,0.153846154,0.132867133,0.146853147,
             0.13986014,0.13986014,0.146853147,0.146853147,0.146853147,0.146853147,0.160839161,
             0.146853147,0.160839161,0.167832168,0.181818182,0.202797203,0.216783217,0.20979021,
             0.202797203,0.195804196,0.195804196,0.216783217,0.160839161,0.13986014,0.13986014,
             0.13986014,0.118881119,0.118881119,0.111888112,0.132867133,0.111888112,0.132867133,
             0.104895105,0.083916084,0.020979021,0,0.230769231,0.636363636,1,0.846153846,
             0.27972028,0.048951049,0.055944056,0.083916084,0.090909091,0.083916084,0.083916084,
             0.076923077,0.076923077,0.076923077,0.090909091,0.06993007,0.083916084,0.076923077,
             0.076923077,0.06993007,0.076923077,0.083916084,0.083916084,0.083916084,0.076923077,
             0.090909091,0.076923077,0.083916084,0.06993007,0.076923077,0.062937063,0.06993007,
             0.062937063,0.055944056,0.055944056,0.048951049,0.041958042,0.034965035,0.041958042,
             0.027972028]

# When we start up, make the LED black
black = (0, 0, 0)
# the color that's passed in over the text input
targetColor = black

# pos is used for all modes that cycle or progress
# it loops from 0-255 and starts over
pos = 0

# curColor is the color that will be displayed at the end of the main loop
# it is mapped using pos according to the mode
curColor = black

# the mode can be one of
# solid - just keep the current color
# blink - alternate between black and curColor
# ramp  - transition continuously between black and curColor
# beat - pulse to a recorded heartbeat intensity
# wheel - change hue around the colorwheel (curColor is ignored)

mode='wheel'

# standard function to rotate around the colorwheel
def wheel(cpos):
    # Input a value 0 to 255 to get a color value.
```

```
    # The colours are a transition r - g - b - back to r.
    if cpos < 85:
        return (int(cpos * 3), int(255 - (cpos * 3)), 0)
    elif cpos < 170:
        cpos -= 85
        return (int(255 - (cpos * 3)), 0, int(cpos * 3))
    else:
        cpos -= 170
        return (0, int(cpos * 3), int(255 - cpos * 3))

# We start by turning off pixels
pixels.fill(black)
pixels.show()

# Main Loop
while True:
    # Check to see if there's input available (requires CP 4.0 Alpha)
    if supervisor.runtime.serial_bytes_available:
        # read in text (@mode, #RRGGBB, %brightness, standard color)
        # input() will block until a newline is sent
        inText = input().strip()
        # Sometimes Windows sends an extra (or missing) newline - ignore them
        if inText == "":
            continue
        # Process the input text - start with the presets (no #,@,etc)
        # We use startswith to not have to worry about CR vs CR+LF differences
        if inText.lower().startswith("red"):
            # set the target color to red
            targetColor = (255, 0, 0)
            # and set the mode to solid if we're in a mode that ignores targetColor
            if mode == "wheel":
                mode="solid"
        # similar for green, yellow, and black
        elif inText.lower().startswith("green"):
            targetColor = (0, 255, 0)
            if mode == "wheel":
                mode="solid"
        elif inText.lower().startswith("yellow"):
            targetColor = (200, 200, 0)
            if mode == "wheel":
                mode="solid"
        elif inText.lower().startswith("black"):
            targetColor = (0, 0, 0)
            if mode == "wheel":
                mode="solid"
        # Here we're going to change the mode - which starts w/@
        elif inText.lower().startswith("@"):
            mode= inText[1:]
        # Here we can set the brightness with a "%" symbol
        elif inText.startswith("%"):
            pctText = inText[1:]
            pct = float(pctText)/100.0
            pixels.brightness=pct
        # If we get a hex code set it and go to solid
        elif inText.startswith("#"):
            hexcode = inText[1:]
            targetColor = hex2rgb(hexcode)
            if mode == "wheel":
                mode="solid"
```

```
        # if we get a command we don't understand, set it to gray
        # we should probably just ignore it but this helps debug
        else:
            targetColor =(50, 50, 50)
            if mode == "wheel":
                mode="solid"
    else:
        # If no text available, update the color according to the mode
        if mode == 'blink':
            if curColor == black:
                curColor = targetColor
            else:
                curColor = black
            sleep(.4)
            # print('.', end='')
            pixels.fill(curColor)
            pixels.show()
        elif mode == 'wheel':
            sleep(.05)
            pos = (pos + 1) % 255
            pixels.fill(wheel(pos))
            pixels.show()
        elif mode == 'solid':
            pixels.fill(targetColor)
            pixels.show()
        elif mode == 'beat':
            pos = (pos + 5 ) % 106
            scaleAvg = (beatArray[(pos-2)%106] + beatArray[(pos-1)%106] + beatArray[pos] +
                        beatArray[(pos+1)%106] + beatArray[(pos+2)%106])/5
            beatColor = tuple(int(scaleAvg*x) for x in targetColor)
            pixels.fill(beatColor)
            sleep(.025)
            pixels.show()
        elif mode == 'ramp':
            pos = ((pos + 5 ) % 255)
            scaleFactor = (2*abs(pos-127))/255
            beatColor = tuple(int(scaleFactor * x) for x in targetColor)
            pixels.fill(beatColor)
            sleep(.075)
            pixels.show()
```