Chromatron Documentation

Release 0.0

Jeremy Billheimer

Mar 30, 2018

Contents:

CHAPTER 1

Overview

Chromatron is an open source Wifi pixel controller. It is a complete hardware and software toolkit designed specifically for making projects with LED pixel strips.

Chromatron began life as a Kickstarter project, which is still an excellent place to see what this project is all about. You can also find more information here.

CHAPTER 2

Getting Started

2.1 Resources for beginners

If this is your first time doing an LED project, you can do no better than Adafruit's excellent getting started guides:

Adafruit NeoPixel Überguide

Adafruit DotStar LEDs

While you're there, please consider picking up your first set of LEDs from them. You can absolutely find LEDs cheaper elsewhere on the Internet, but Adafruit helps enable projects like Chromatron to exist by the sheer volume of information, guides, and enthusiasm they create. The Maker community thrives on the good will of its members.

2.2 Where to get help

Stuck? Something broken? Just want to show off your project?

This is the place for all of that.

Want to file a bug report (or request a feature)? You can do that on the Github page. Please only file an issue for actual bugs (or features) - if you just need help, the Google Group is a better place for that.

2.3 Things you will need

- Chromatron
- LEDs
- A suitable power supply
- A micro USB cable
- A precision sized flathead screwdriver (1/16 inch or 1.6 mm)

- A multimeter (not technically required, but since you're working with electricity, you should really have one)

2.4 Install tools

First, you're going to need Python 2.7, which you can get here. You won't need to use Python directly, but the command line tools need it to run.

Next, install Chromatron's tools:

```
$ pip install chromatron
```

You can check that it installed by checking the version:

```
$ chromatron --version
```

2.5 Basic setup

First, we're going to get Chromatron on your Wifi network.

Plug in your Chromatron via USB. The on-board LED will go through a few states as the operating system boots up. See *Status LED* to learn what the LED colors mean.

Let's see if we can talk to the Chromatron:

```
$ chromatron --host usb discover
```

This should come up with some information about your Chromatron.

Next, let's set up the Wifi:

```
$ chromatron --host usb wifi setup
```

Note: Chromatron only supports 2.4GHz networks.

Next, let's give it a name:

```
$ chromatron --host usb meta setup
```

Note: The meta setup will also ask for a location and a set of tags. These are optional, but useful for assigning Chromatrons to groups. Read more about meta tags here: *Key-value System, Meta Tags, and Queries*.

Find your Chromatron on the network:

```
$ chromatron discover
```

If you have multiple Chromatrons, you can also query for specific ones:

```
$ chromatron -q meow discover
```

This will find any Chromatrons named, tagged, or in location 'meow'.

Note: Meta tags must be 32 characters or less, encoded in ASCII (no Unicode support for now), and must not contain spaces.

2.6 LED setup

Now for the fun part: connecting your LED strip. Hardware connections are shown here: *Hardware Guide*.

Warning: Unplug the USB cable before connecting an LED strip!

Caution: Powering via USB: Chromatron can power LEDs via the USB port. However, USB ports can only power a small number of LEDs (8 to 10 is fairly safe - unless you are using high power LEDs like the Pixie). If you connect a full 300 LED strip and plug in the USB, you might damage your USB port or Chromatron, and that would be sad.

Got your LEDs connected? Double check your wiring before you power up! Everything ok? Let's move on:

```
$ chromatron -q meow pixels setup
```

This will walk you through the pixel setup, and also get the RGB order configured.

2.7 Dimmer Controls

You can control the dimmers via the command line:

```
$ chromatron -q meow dimmer master 1.0
```

Will set to full brightness.

See *Dimmers and Faders* for more detailed information about how the dimmer controls work.

2.8 FX Script

Example scripts are in the FX folder in the Github repository. You can download it here if you are not using git: https://github.com/sapphireos/chromatron/archive/master.zip

The LED setup loads the rainbow demo. You can load different scripts:

```
$ chromatron -q meow vm load script_name.fx
```

Note: The chaser.fx example will not work properly on firmware v1.0, due to a slight bug in the virtual machine (it runs the loop twice per iteration, instead of only once). This will be fixed in the next release. The update procedure is documented here: *Updating firmware*

If you're ready for more, FX script has its own guide: *FX Script Guide*

CHAPTER 3

FX Script Guide

FX Script is a scripting language for creating graphics on LED pixels (such as WS2801, APA102, Neopixels, etc). It is a simple, procedural language that uses Python syntax and runs on a custom designed virtual machine.

The language is designed for simplicity. This makes it easy to learn and easy to run on resource constrained hardware (low power microcontrollers with less than 10 KB of RAM). The virtual machine has a built in graphics system and optimizes common operations for pixel graphics to allow it to run as fast as possible. The compiler can operate in a live mode to allow instant code updates to any devices on the network.

The memory constraints and speed optimizations mean the language must be limited in many respects. Object oriented programming, advanced data structures, exception handling, strings, floating point math, and even recursion are not supported. The goal is to enable rapid development of pixel graphics for small to moderately sized projects (generally 300 pixels or less). If you need full 3D graphics across 20,000 pixels, FX Script is probably not the tool for you. However, if you want to do live code updates on more modest projects, FX Script can dramatically improve productivity.

3.1 Installation

The FX Script compiler is included with the chromatron Python package, which can be installed via pip:

```
$ pip install chromatron
```

3.2 Usage

The compiler is invoked via the command line:

```
$ chromatron compile script_name.fx
```

By convention, source files use a .fx extension and the compiled binary uses .fxb.

Compile and load a script to a device:

```
$ chromatron vm load script_name.fx
```

Live mode: This will set up the command to recompile and reload the script file every time it is changed.

```
$ chromatron vm load script_name.fx --live
```

See *Command Line Interface Reference* for more information on how to use the command line interface.

3.3 Example

We'll start with an annotated example of a rolling rainbow pattern. Afterwards, we'll explain what everything does. This is the FX Script equivalent of "hello world".

```
# rainbow.fx

# this script generates a rolling rainbow pattern

# declare a global variable for current hue
current_hue = Number()

# init - runs once when script is loaded
def init():
    # set pixels to full colors (maximum saturation)
    pixels.sat = 1.0

    # set to maximum brightness
    pixels.val = 1.0


# runs periodically, frame rate is configurable
def loop():
    # increment the base hue so the rainbow pattern
    # shifts across the pixels as we go from one frame
    # to the next.
    current_hue += 0.005

    # declare a local variable
    a = Number()
    a = current_hue

    # loop over all pixels in array
    for i in pixels.count:
        pixels[i].hue = a

        # shift color for next pixel.
        # this will distribute the rainbow pattern
        # across the entire array.
        a += 1.0 / pixels.count
```

3.4 Basics

FX Script uses Python as the front end language, so the best way to learn FX is to learn some basic Python first, and then come back here.

There are a number of differences from Python:

3.4.1 Program Structure

Each program requires an init() function. init() executes once when the program is loaded.

Most programs will also need a loop() function. loop() is called repeatedly by the virtual machine. The loop frame rate is configurable. See *Frame Rate*.

```
def init():
    # setup code goes here

    pass

def loop():
    # called repeatedly to generate each frame of your graphics

    pass
```

3.4.2 Variables

There is only one basic data type, Number, which is a signed 32 bit integer. Number ranges from -2,147,483,648 to 2,147,483,647.

Variables must be declared:

```
my_variable = Number()
```

All variables will initialize to 0.

3.4.3 For Loops

For loops do not uses the range/xrange functions:

```
# correct
for count in my_variable:
    # iterate my_variable number of times.
    # count will start at 0 and increment with each iteration.
    pass

# wrong
for count in xrange(my_variable):
    pass
```

3.4.4 Arrays

Instead of Python's lists, FX uses fixed length arrays.

```
my_array = Array(4) # declare an array of 4 Numbers

# you can do this:
for i in my_array.count:
    temp = Number()
```

```
    temp = my_array[i]

# you cannot do things like this:
for i in my_array:
    temp = Number()
    temp = i
```

3.4.5 Floats

Floating point numbers are a shortcut to represent integer values in the graphics system. They do *not* behave like normal floating point numbers in Python. Remember - the underlying virtual machine only understands integers.

Most internal graphics parameters are represented as 16 bit integers (0 to 65535). However, it is often simpler to represent these values as a floating point number between 0.0 and 1.0. Thus, in FX Script the number 0.5 represents the integer 32768. You can use these special floats in expressions, such as 0.1 + 0.1, but be aware that something like 0.5 * 0.5 may not do what you expect. Instead of yielding 16384 (0.25), you will actually get 32768 * 32768 = 1,073,741,824. It is generally best to avoid complex math with the floating point representation.

3.5 Graphics System

3.5.1 HSV Colorspace

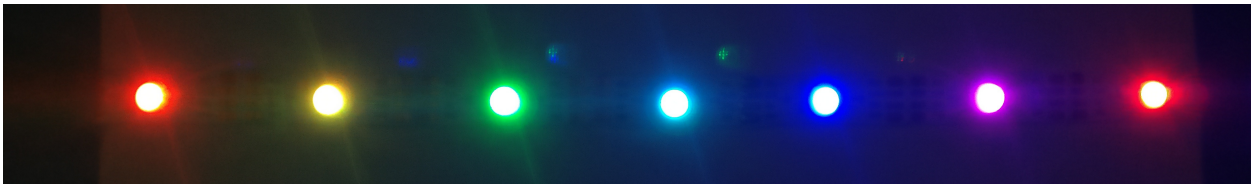
All graphics in FX Script use the HSV color space. Although the LEDs themselves (along with our eyes) use RGB, HSV is much simpler to design for specific colors. If you're not familiar with HSV, the Wikipedia article is a good place to start. However, we'll explain the basics right here.

HSV is shorthand for hue, saturation, value. You could translate it as color, whiteness, brightness.

Hue

Hue is color. It works as a circle, with the colors of the rainbow spaced out along the edge. 0.0 is the top of the circle, and represents the color red. 1/3 (0.333) is green, 2/3 (0.667) is blue. Every other color is a blend between these values:

hue	color	16 bit integer value
0.000	red	0
0.167	yellow	10944
0.333	green	21823
0.500	teal	32767
0.667	blue	43711
0.833	purple	54590
1.000	red	0



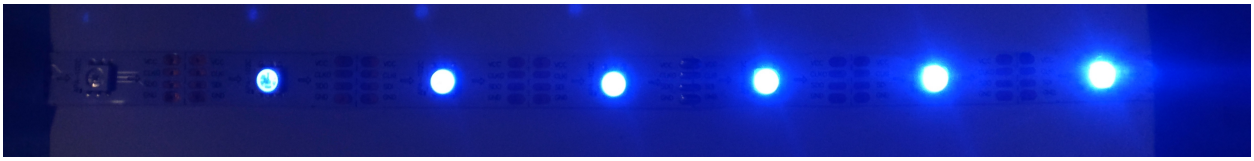
Note that 1.0 is the same as 0.0, hue wraps around as a circle. Thus, the value 1.5 would be the same as 0.5, etc. As you'll see in the rainbow demo, this wraparound behavior is incredibly useful in doing continuous, smooth color shifts.

Sat

Saturation (sat for shorthand) is the whiteness value. 1.0 is full saturation, meaning you get all of the colors of the rainbow. 0.0 is no saturation, which is full white. The pastel colors are created by selecting a hue, and then adjusting the saturation.

Val

Value (val for shorthand) is brightness. 0.0 is off, 1.0 is maximum.



3.5.2 Pixel Array

This is where the fun begins. The pixel array is how we get HSV values into the pixel strip itself.

There is a master pixel array called pixels. You can set HSV values on individual pixels like so:

```
pixels[0].hue = 0.5 # set pixel 0 to teal
pixels[0].val = 1.0 # set pixel 0 to max brightness

pixels[1].val = 0.0 # set pixel 1 to off
```

If you wanted to turn on all of the pixels in the array, you could do something like this:

```
for i in pixels.count:
    pixels[i].val = 1.0 # set pixel to max brightness
```

However, this is a pretty common operation, so FX Script has a better way of doing the same thing:

```
pixels.val = 1.0 # set all pixels to max brightness
```

Since pixels was not indexed, FX knows that you want to write to the entire array. While it's nice that this saves some typing and makes your code look a bit cleaner, the other benefit is that the virtual machine can skip doing a for loop in the VM's bytecode, and instead run an optimized instruction with the loop implemented in C. This means that the second method is much, much faster than the first one.

This technique is not limited to just assignment. You can adjust parameters mathematically as well:

```
pixels.val += 0.1 # increment all pixels' brightness by 0.1
```

This works with +, -, *, /, and % operators.

What happens when the pixel val exceeds 1.0? FX understands what hue, sat, and val all mean - it knows they are special.

In the case of sat and val, FX will automatically limit to a range between 0.0 and 1.0. This means if you attempt to increment val beyond 1.0, it will just stop at 1.0. The same occurs if you try to decrease below 0.0. It knows you can't have a negative brightness, and it also knows you probably don't want to jump from 0.0 to 1.0 when you're using this syntax.

For hue, FX will allow the parameter to wrap around. If hue is 0.99 and you increment by 0.1, hue will end up at 0.09 (0.99 + 0.1 = 1.09, so we wrap to 0.09). This is how doing a rainbow color effect works, you can just increment hue continuously and it will wrap around. No need to bounds check (unless you want to intentionally limit the color range!).

There are some limitations of course. For instance, you cannot do an array read:

```
if pixels.val == 0.5: # this is not valid
    pass

temp = Array()
temp = pixels.val # this is also not valid
```

The Array() declaration is also wrong because it does not specify the length. The point is, this type of operation is not supported.

This means that you cannot do something like this:

```
pixels.val = pixels.hue # sorry, FX doesn't work this way.
```

If for some reason you really needed to do something like that, you can always write your own loop.

API

class fx_api.**PixelArray**

An array of *Pixel*

count

Get the number of pixels in the array.

hs_fade

Set the hue/sat fader of all pixels in the array.

hue

Set the hue of all pixels in the array.

is_fading

Check if any pixels in the array are fading.

Returns

0 - no pixels are fading

1 - one or more pixels are fading

sat

Set the saturation of all pixels in the array.

size_x

Get the number of pixels in the X dimension of the array.

size_y
Get the number of pixels in the Y dimension of the array.

v_fade
Set the value fader of all pixels in the array.

val
Set the value of all pixels in the array.

class fx_api.**Pixel**
A single pixel

hs_fade
Get/set the hue/sat fader of pixel.

hue
Get/set the hue of pixel.

is_fading
Check if pixel is fading.

Returns

0 - pixel is not fading

1 - pixel is fading

sat
Get/set the saturation of pixel.

v_fade
Get/set the value fader of pixel.

val
Get/set the value of pixel.

3.5.3 Dimmers and Faders

The graphics system has a number of dimmers and faders built in.

Dimmers

There are two global dimmers, called master and sub. Each affects the brightness of all pixels. Additionally, each pixel's val will further modulate the brightness for that pixel. The dimmer values multiply to achieve the final dimming level for each pixel:

pixel_brightness = master * sub * pixel val

For instance, if master is 0.5, sub is also 0.5, and the pixel val is 1.0, the actual dimmed output will be 0.25. If the master, sub, and the pixel val are all 0.5, that pixel will be at 0.125.

The master and sub dimmers are currently only accessible over the network via the CLI and Python APIs. There is no interface from within FX Script to override them.

Check the current dimmer levels:

```
$ chromatron dimmer show
```

Set master dimmer to 0.5:

```
$ chromatron dimmer master 0.5
```

Set sub dimmer to 0.3:

```
$ chromatron dimmer sub 0.3
```

See *Command Line Interface Reference* for more information on how to use the command line interface.

Faders

Each pixel has two time based faders. One fader is shared between hue and sat, the other is used for val. This allows FX to produce smooth timed fades automatically.

The fader system runs every 20 ms, regardless of the frame rate of the FX VM.

```
# assumes pixels start fully off

pixels.v_fade = 1000 # set val fader to 1000 ms fade
pixels.val = 1.0 # set all pixels to max brightness
```

This code will instruct each pixel to fade up to max brightness over the course of one second. Note that each time you set val it will recompute the fader (thus resetting the one second timer from wherever val is at the time). If the frame rate is faster than once per second, you can ensure a one second fade by doing something like this:

```
# assumes pixels start fully off

# only start the value fade if the pixels are not currently fading.
if pixels.is_fading == 0:
    pixels.v_fade = 1000 # set val fader to 1000 ms fade
    pixels.val = 1.0 # set all pixels to max brightness
```

The hue/sat fader is hs_fade, and operates the same way.

3.5.4 Frame Rate

The frame rate (rate at which the loop() function is called) is adjustable via the Python API and CLI. There is currently no interface within FX to adjust it.

Check the current frame rate with:

```
$ chromatron keys get gfx_frame_rate
```

Set the frame rate with:

```
$ chromatron keys set gfx_frame_rate 100
```

This will set the frame rate to 100 ms, or 10 times per second. The range is adjustable from 10 ms to 65535 ms.

See *Command Line Interface Reference* for more information on how to use the command line interface.

3.6 Performance Notes

The FX VM is designed to be as fast as possible, but the fact is that a VM incurs some overhead that would normally be done by hardware in machine code. The processor has a tremendous amount of work to do: process the FX script

itself, run faders over 300 LEDs with 3 information channels each 50 times per second (this is 45,000 channel updates per second), do 15,000 HSV to RGB conversions per second, and also manage the Wifi connection.

You can check the performance of your script with the following command:

```
$ chromatron keys get vm_loop_time
```

This will return the number of microseconds your loop function takes to execute.

You can also check the timing of the fader system:

```
$ chromatron keys get vm_fade_time
```

For reference, with 300 pixels and running the default rainbow script, the faders will run in 800 to 900 microseconds and the script loop itself around 1,000 microseconds.

If your script is running slow, there are a few things you can do to help.

Try lowering the frame rate. If you are using a high frame rate to achieve smooth fades, consider that the fader system is already trying to do this for you .

Make sure you use array operations where possible, they are much faster than looping in the script.

Be aware that the compiler only does *very* basic optimizations. For instance, it will optimize expressions that only involve constants:

```
a = Number()
a = 1 + 2 + 3
```

The compiler will evaluate this expression and just assign 6 to variable a.

However, it currently does not do more sophisticated optimizations:

```
a = Number()

for i in pixels.count:
    pixels[i].hue = a

    a += 1.0 / pixels.count
```

In this case, the 1.0 / pixels.count is computed *every* iteration. A faster way is this:

```
a = Number()
b = Number()
b = 1.0 / pixels.count

for i in pixels.count:
    pixels[i].hue = a

    a += b
```

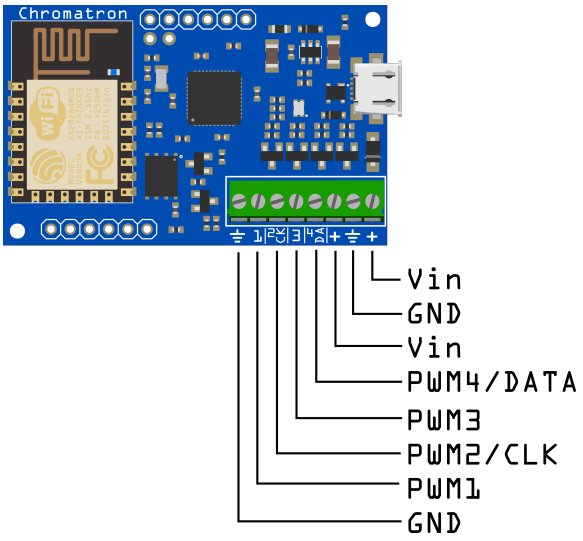
Just that change in the rainbow script saves almost 400 microseconds.

Finally, note that memory is also constrained. The FX compiler will impose a limit of 128 variables (which includes all of the elements in arrays). The .fxb file format is a somewhat self-contained VM image, so you can use the file size as a reference for how much memory (code and data) your script uses. The rainbow demo is less than 200 bytes. The VM itself has 4096 bytes allocated, however, a script that actually uses that much memory is unlikely to perform well at high pixel counts.

CHAPTER 4

Hardware Guide

4.1 Pinouts



4.1.1 Power

Vin - 4 to 16V power supply connection

Vin must match the supply voltage required for the LEDs. Thus, if you are using 5V LEDs, Vin needs to be 5V. The same goes with 12V LEDs.

> **Warning:** Double check your supply voltage before powering on for the first time! If you're powering Chromatron on 12V and plug in 5V LEDs, you're going to damage the LED strip (Chromatron itself will be fine).

Note: Note that in the case of the WS2811 at 12V, the data and clock drivers with automatically limit their output to less than 6V (the WS2811 power is 12V, but signalling is nominally 5V).

GND - Ground connection

There are two sets of power pins. Both Vins and both GNDs are shorted together in the PCB with thick traces. You can supply power to the board with one set, and use the other as power connections for the LED strip.

4.1.2 IO

PWM1 - PWM output for analog pixel mode

PWM2/CLK - PWM output or pixel clock, depending on pixel strip mode.

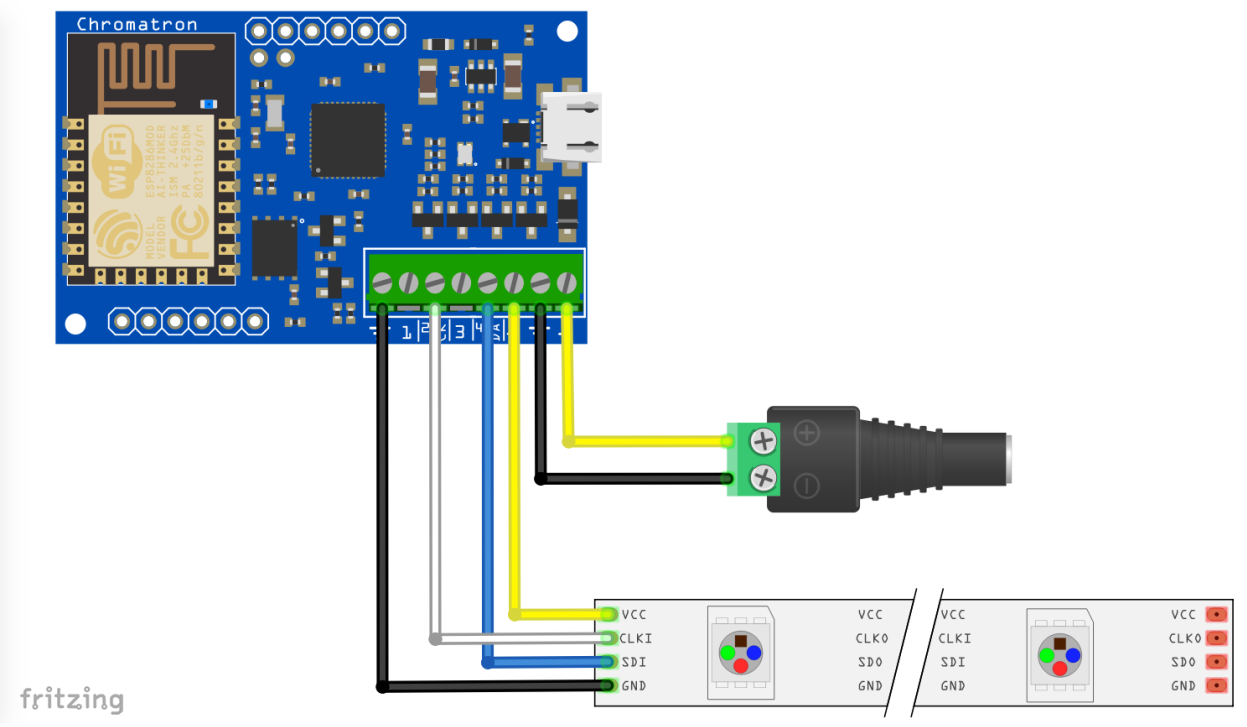
PWM3 - PWM output for analog pixel mode

PWM4/DATA - Pixel data output. This can also be a spare PWM channel, however, this is not yet supported in the current firmware version.

4.2 LED Connections

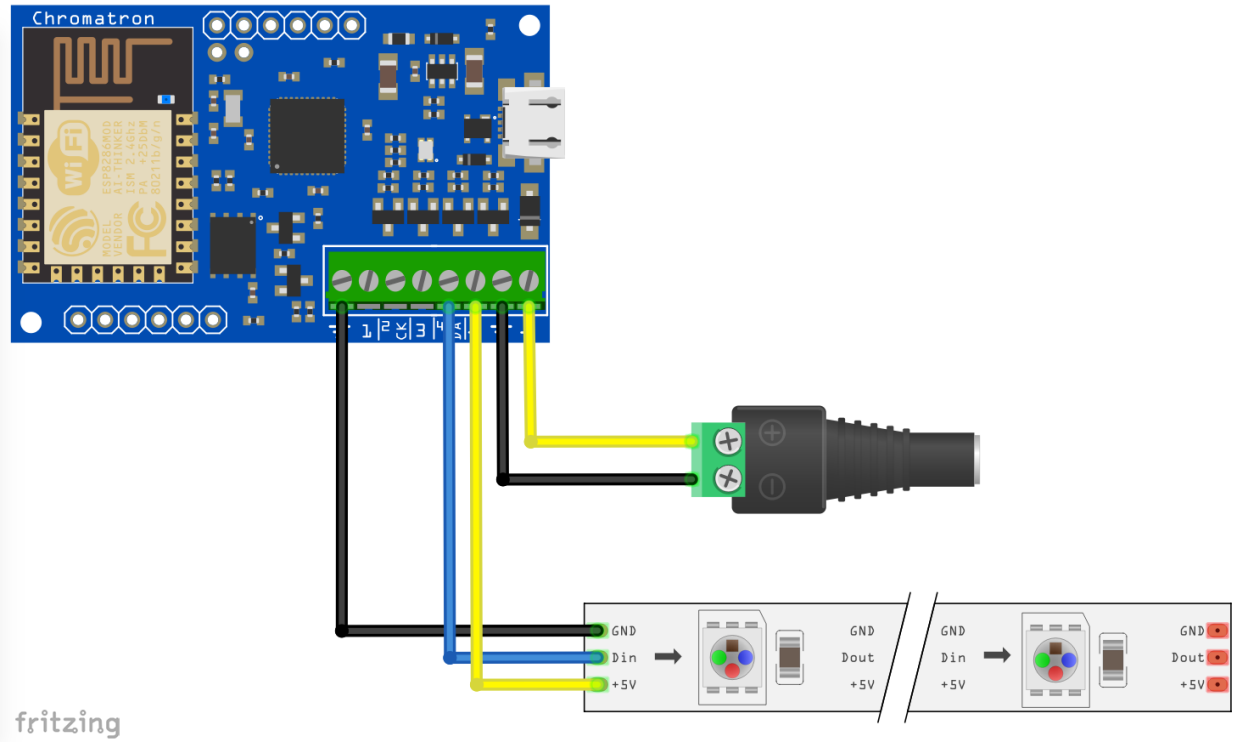
4.2.1 4 Wire (APA102, WS2801)

4 wire LED strips use one set of power connections, data, and clock.



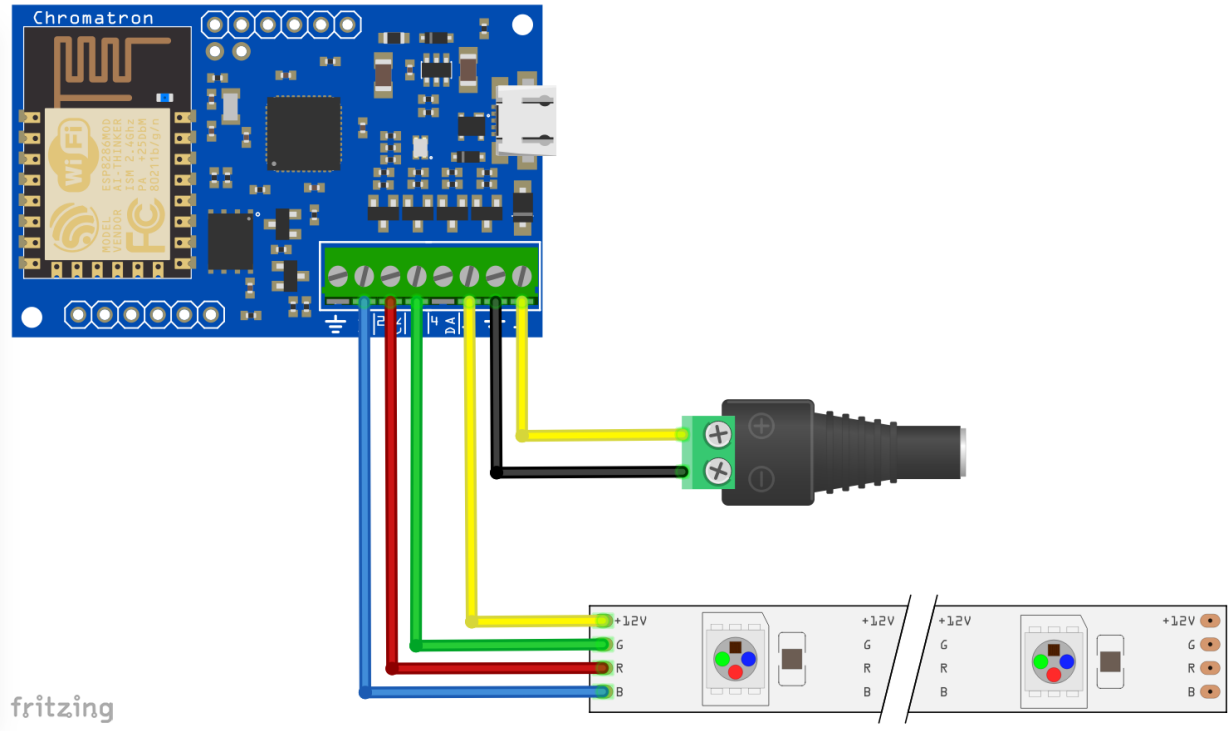
4.2.2 3 Wire (WS2812, WS2812b, WS2811, Pixie)

3 wire LED strips use one set of power connections and data. Clock is not used.



4.2.3 PWM

PWM strips use PWM1, PWM2, and PWM3. Connect on of the Vin connections to the strip's power input. You do not need to worry about the RGB order, it is configurable in software.



4.3 Specifications

Vin - 4V to 16V

PWM channels - Max. 3 amps each

Pixel channels - 320 addressable channels

CHAPTER 5

Status LED

Chromatron has an RGB status LED mounted near the middle of the board. The colors and patterns all mean different things:

5.1 Power up

5.1.1 Solid yellow

This means the bootloader is running. It will flash briefly on a normal start up, and will light for a few seconds after a firmware update.

5.1.2 Solid green

The operating system is booting up.

5.2 Normal operation

5.2.1 Flashing green

The operating system is running, and Wifi is not connected.

5.2.2 Flashing blue

Wifi is connected.

5.2.3 Flashing purple

Wifi is running as an access point.

Note: On firmware v1.0, this mode will flash blue/white.

5.3 Errors

5.3.1 Flashing red

The operating system has entered safe mode. Please post on the support group if this happens, as this usually indicates the operating system has crashed and we may have a bug.

CHAPTER 6

Updating firmware

Download newest firmware:

```
$ chromatron firmware update
```

List firmware releases on your system:

```
$ chromatron firmware releases
```

Upgrade firmware to latest version:

```
$ chromatron firmware upgrade
```

Change firmware to a specific version:

```
$ chromatron firmware upgrade -r release_version
```

CHAPTER 7

Key-value System, Meta Tags, and Queries

Chromatron's operating system is built around an embedded key-value database. A basic key-value database is simply a mapping between a string and some value, and some interface to read and write those values.

The key-value database is central to how Chromatron works. Almost all of its configuration data (things like which Wifi network to connect to, dimmer settings, etc) and much of its run time information (such as Wifi signal strength, input voltage, etc) is stored there. Most of the time when you're changing a setting on Chromatron through the Python API or command line tools, you're accessing the database.

The database has eight items which are special, called "meta tags". Each of these tags is a string, up to 32 characters long. By convention, the first tag is the device name and the second tag is the device location. The other six have no special meaning. These meta tags are used to find and identify devices in a way which is more descriptive than just using a single name, IP address, or some other device ID number.

A query is a listing of up to 8 strings that are compared against the meta tags. They work as a keyword search, so the order of strings in the meta tags or the query doesn't matter. For instance, if you query for two tags, 'King_Arthur' and 'Camelot', you'll retrieve any devices that have both of those strings in their meta tags. The positional order does not matter. This makes it easy to compose multiple overlapping groups of devices.

CHAPTER 8

Command Line Interface Reference

8.1 chromatron

Chromatron Command Line Interface

All of these commands, except for a few setup commands, can run on groups of devices. If the host or query options are passed, the command will discover matching devices and run on those results. If options are not provided, the command will run on the results from the last invocation of the discover command.

Examples:

chromatron.py discover –query Camelot

> Finds and stores all devices with a query tag matching Camelot.

chromatron.py firmware version

> List firmware version on all devices matching the previous discover, in this case, devices located in Camelot.

chromatron.py –query King_Arthur firmware version

> List firmware version on all devices with a tagged with King_Arthur. This will not overwrite the previous discovery results.

chromatron.py firmware version

> Since no options are given, this will use the previous discovery, again matching devices located in Camelot. This does not re-run the discovery, so if a device is added or removed from Camelot, the discover command would need to be run again.

```
chromatron [OPTIONS] COMMAND [ARGS]...
```

Options

-h, --host <host>
Name or IP address of target. Can also specify USB for local connection.

-q, --query <query>
Query for matching tags.

--version
Show the version and exit.

8.1.1 automaton

Automaton controls

```
chromatron automaton [OPTIONS] COMMAND [ARGS]...
```

load

Compile and load script to automaton

```
chromatron automaton load [OPTIONS] FILENAME
```

Options

--live
Live mode

Arguments

FILENAME
Required argument

8.1.2 compile

Compile an FX script

```
chromatron compile [OPTIONS] FILENAME
```

Options

--debug
Print debug information during script compilation

Arguments

FILENAME
Required argument

8.1.3 dimmer

Dimmer commands

```
chromatron dimmer [OPTIONS] COMMAND [ARGS]...
```

master

Set master dimmer.

VALUE is from 0.0 to 1.0.

```
chromatron dimmer master [OPTIONS] VALUE
```

Arguments

VALUE
Required argument

show

Show current dimmer settings

```
chromatron dimmer show [OPTIONS]
```

sub

Set submaster dimmer.

VALUE is from 0.0 to 1.0.

```
chromatron dimmer sub [OPTIONS] VALUE
```

Arguments

VALUE
Required argument

8.1.4 discover

Discover Chromatron devices on the network

Discovers devices matching host or tag options. The results of this command will be stored to a file, and will be used as the default set of devices for all other commands.

You can specify multiple tags to narrow the search. Only devices matching all tags will be found.

If host or tags are not specified, discover will find all devices.

Note that host and tag options should be passed before discover:

This will work:

chromatron.py –tag Camelot discover

This will not:

chromatron.py discover –tag Camelot

```
chromatron discover [OPTIONS]
```

8.1.5 firmware

Firmware commands

```
chromatron firmware [OPTIONS] COMMAND [ARGS]...
```

backup

Backup settings on selected devices

```
chromatron firmware backup [OPTIONS]
```

manifest

Show manifest for current firmware package

```
chromatron firmware manifest [OPTIONS]
```

Options

-r, --release <release>
Name of release to display

releases

List available releases

```
chromatron firmware releases [OPTIONS]
```

restore

Restore settings to selected devices

```
chromatron firmware restore [OPTIONS]
```

update

Update releases

```
chromatron firmware update [OPTIONS]
```

upgrade

Upgrade firmware on selected devices

```
chromatron firmware upgrade [OPTIONS]
```

Options

-r, --release <release>
Name of release to load. Default is latest.

--force
Force firmware upgrade even if versions match.

--change_firmware <change_firmware>
Change firmware on device.

--yes
Answer yes to all firmware change confirmation prompts.

version

Show firmware version on selected devices

```
chromatron firmware version [OPTIONS]
```

8.1.6 fs

File system commands

```
chromatron fs [OPTIONS] COMMAND [ARGS]...
```

cat

View a file

```
chromatron fs cat [OPTIONS] FILENAME
```

Arguments

FILENAME
Required argument

list

List files

```
chromatron fs list [OPTIONS]
```

put

Put a file

```
chromatron fs put [OPTIONS] FILENAME
```

Arguments

FILENAME
Required argument

rm

Remove a file

```
chromatron fs rm [OPTIONS] FILENAME
```

Arguments

FILENAME
Required argument

8.1.7 identify

Identify devices

```
chromatron identify [OPTIONS]
```

8.1.8 keys

Key-value system commands.

These commands provide direct access to the key-value database.

Use caution when setting system keys. Actions cannot be undone, and a careless setting may require a complete reset and reconfiguration of the device.

```
chromatron keys [OPTIONS] COMMAND [ARGS]...
```

get

Get key on devices

```
chromatron keys get [OPTIONS] KEY
```

Arguments

KEY
Required argument

list

List all keys on devices

```
chromatron keys list [OPTIONS]
```

set

Set key on devices

```
chromatron keys set [OPTIONS] KEY VALUE
```

Arguments

KEY
Required argument

VALUE
Required argument

8.1.9 meta

Meta data setup commands

```
chromatron meta [OPTIONS] COMMAND [ARGS]...
```

add_tag

Add a meta data tag to devices

```
chromatron meta add_tag [OPTIONS] TAG
```

Arguments

TAG
Required argument

list

List meta data tags on devices

```
chromatron meta list [OPTIONS]
```

reset

Reset meta data to default

```
chromatron meta reset [OPTIONS]
```

Options

--force
Force meta data reset without prompting for confirmation.

rm_tag

Remove a meta data tag from devices

```
chromatron meta rm_tag [OPTIONS] TAG
```

Arguments

TAG
Required argument

set_location

Set location on devices

```
chromatron meta set_location [OPTIONS] TAG
```

Arguments

TAG
Required argument

set_name

Set name on devices

```
chromatron meta set_name [OPTIONS] TAG
```

Arguments

TAG
Required argument

setup

Set meta data on devices.

Meta data is:

name location

and up to 6 additional tags

Each item is an ASCII string, up to 32 characters.

```
chromatron meta setup [OPTIONS]
```

8.1.10 ping

Ping devices

```
chromatron ping [OPTIONS]
```

8.1.11 pixels

Pixel commands

```
chromatron pixels [OPTIONS] COMMAND [ARGS]...
```

get_clock

Get pixel clock.

Returns current pixel clock in Hz.

```
chromatron pixels get_clock [OPTIONS]
```

hue

Set hue on all pixels

```
chromatron pixels hue [OPTIONS] VALUE
```

Arguments

VALUE
Required argument

reset

Reset pixel configuration to default

```
chromatron pixels reset [OPTIONS]
```

sat

Set sat on all pixels

```
chromatron pixels sat [OPTIONS] VALUE
```

Arguments

VALUE
Required argument

set_clock

Set pixel clock.

VALUE is in Hz.

Valid range is 250 KHz to 3.2 MHz.

The default pixel clock is set to 1 MHz. It is not recommended to change this.

The signal quality will begin to degrade above 2 MHz, and operation above this range is not guaranteed.

The WS2811/WS2812 mode uses a fixed clock and will override any setting attempted through this command.

This setting is ignored in PWM mode.

```
chromatron pixels set_clock [OPTIONS] VALUE
```

Arguments

VALUE
Required argument

setup

Configure pixel driver settings.

This command will set up pixel type, number of pixels, and RGB order.

```
chromatron pixels setup [OPTIONS]
```

show_settings

Show pixel settings.

Shows pixel mode, clock, and pixel count.

Note that some of the pixel modes map to multiple pixel types. This command will return the name of the pixel mode used by Chromatron internally, and may not match the name used to configure the device.

Example: ws2811, ws2812, and ws2812b internally all use the ws2811 mode.

```
chromatron pixels show_settings [OPTIONS]
```

val

Set val on all pixels

```
chromatron pixels val [OPTIONS] VALUE
```

Arguments

VALUE
Required argument

8.1.12 reboot

Reboot devices

```
chromatron reboot [OPTIONS]
```

8.1.13 show

Show current devices

```
chromatron show [OPTIONS]
```

8.1.14 vm

Virtual machine controls

```
chromatron vm [OPTIONS] COMMAND [ARGS]...
```

clean

Erase all VM script files (.fxb)

```
chromatron vm clean [OPTIONS]
```

load

Compile and load script to virtual machine

```
chromatron vm load [OPTIONS] FILENAME
```

Options

--live
Live mode

Arguments

FILENAME
Required argument

reload

Recompile and reload the FX script on device

```
chromatron vm reload [OPTIONS]
```

reset

Reset virtual machine

```
chromatron vm reset [OPTIONS]
```

start

Start virtual machine

```
chromatron vm start [OPTIONS]
```

stop

Stop virtual machine

```
chromatron vm stop [OPTIONS]
```

8.1.15 wifi

Wifi setup commands

```
chromatron wifi [OPTIONS] COMMAND [ARGS]...
```

monitor

Monitor Wifi signal strength

```
chromatron wifi monitor [OPTIONS]
```

reset

Reset wifi configuration to default.

This command only works over USB.

```
chromatron wifi reset [OPTIONS]
```

setup

Configure wifi settings over USB.

Connects to a device over USB and prompts for wifi parameters. If multiple devices are connected to USB, only one of them will be selected.

This command ignores discovery parameters and only works over USB. This is to prevent accidentally breaking the wifi configuration of an entire group of devices. If you really need to change wifi parameters of a device over wifi, see the keys subcommand.

You can pass the SSID and password on the command line as a shortcut.

```
chromatron wifi setup [OPTIONS]
```

Options

--wifi_ssid <wifi_ssid>
Wifi router

--wifi_password <wifi_password>
Wifi password

CHAPTER 9

Documentation To-Do

Chromatron is an extremely powerful system, and the documentation at present only covers the basics.

Here's some things that still need documentation. Just because the docs aren't finished doesn't mean you can't use advanced features! If you want to learn more about how something works but don't want to wait for me to finish writing, post a question in the user group.

- database linkage to FX script: you can link variables in FX script to the database, and then access them over the network.
- pixel arrays: you can create virtual pixel arrays in an FX script and access them independently. This is useful if you want to control different sections of a single strip differently (for instance, if you made a light up cube, you could have a separate array defined for each side).
- Accessing the auxiliary IO connections to connect sensors, etc: this will require a firmware update (which is not yet released), but will allow your scripts to interact with the outside world.
- Using the Python API directly: you can integrate Chromatron's drivers into your own projects, stream raw graphics directly to the pixel array, etc.
- Building your own custom firmware: Not for the faint of heart, as Chromatron has about 20,000 lines of operating system that makes it work. However, it is an extremely powerful system and can be used as a general purpose microcontroller platform.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

Python Module Index

f

Index

Symbols

C

F

H

I

K

P

S

T

V