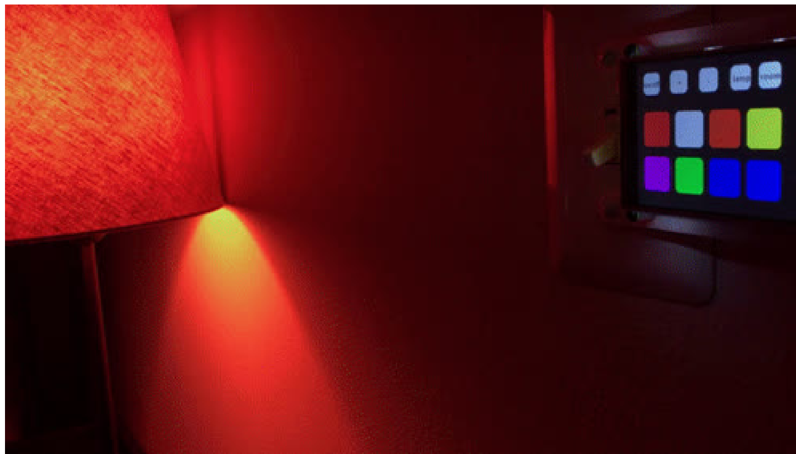
PyPortal LIFX Lighting Controller

Created by Brent Rubell



Last updated on 2020-09-11 04:40:46 PM EDT

Overview

Want to add unique lighting to your room?

A simple solution would be adding a couple of **WiFi-enabled LED smart-bulbs**. These bulbs are energy efficient and come in a variety of shapes, sizes, and colors to fit your space. Smaller bulbs can be used for accent-lighting or lamps, while larger bulbs are more powerful and can light up an entire room!

These types of bulbs are *incredibly* popular - and are sold in most major electronics brick-and-mortar retailers.

In this guide, **you'll be turning your PyPortal into a smart lighting controller capable of selecting and setting multiple light sources**.

Unlike a more complicated smart-lighting system like the popular Phillips Hue (https://adafru.it/Evp), LIFX lights are incredibly simple to setup. each light has its own wifi radio (*possibly something ESP32 (https://adafru.it/Evq) or ESP8266 (https://adafru.it/Evr)-based?* We can't tell from the teardown photo (https://adafru.it/Evs)) and connects to your WiFi network.

Most notable about the LIFX lights is they provide a well-documented HTTP API (https://adafru.it/Evt), similar to the Adafruit IO HTTP API (https://adafru.it/uff). This means you can easily interface with it using your PyPortal.

If you're looking for a way to **interactively control a room's lighting using your PyPortal**, follow along!

```
while True:
    touch = ts.touch_point
    if touch:
        for i, button in enumerate(buttons):
            if button.contains(touch):
                button.selected = True
                if button.name == 'lamp':
                    current_light = lifx_lights[0]
                    print('Switching to ', current_light)
                elif button.name == 'room':
                    current_light = lifx_lights[1]
                    print('Switching to ', current_light)
                elif button.name == 'onoff':
                    print('Toggling {0}...'.format(current_light))
                    lifx.toggle_light(current_light)
                elif button.name == 'up':
                    light_brightness += 0.25
                    print('Setting {0} brightness to {1}'.format(current_light, light_brightnes
                    lifx.set_brightness(current_light, light_brightness)
                elif button.name == 'down':
                    light_brightness -= 0.25
                    print('Setting {0} brightness to {1}'.format(current_light, light_brightnes
                    lifx.set_brightness(current_light, light_brightness)
                else:
                    print('Setting {0} color to {1}'.format(current_light, button.name))
                    lifx.set_color(current_light, 'on', button.name, light_brightness)
                button.selected = False
            else:
                button.selected = False
```

CircuitPython Code

CircuitPython is perfect for building Internet-of-Things projects. This project uses the ESP32SPI CircuitPython (https://adafru.it/DWV)library, which can use the ESP32 as a WiFi-coprocessor to send web requests to the LIFX HTTP Remote Control API.

We've also built a LIFX helper module for CircuitPython (https://adafru.it/Evu) to make interfacing with these lights incredibly easy.

You can rapidly update your code without having to compile and store WiFi and API secret keys on the device. This means that there's no editing code and re-uploading whenever you move the PyPortal to another network - just update a file and you're set.

PyPortal

Say hello to PyPortal! The easiest way to build your IoT projects with a touchscreen display! Make sure to walk through the PyPortal introduction guide and walkthrough the pages. It'll get you setup with CircuitPython and a handful of demo code to play with! (https://adafru.it/Ecp)

Prerequisite Guides

If you're new to CircuitPython, take a moment to walk through the following guides to get you started and up-to-speed:

- Welcome to CircuitPython (https://adafru.it/cpy-welcome)

Parts

You only need a PyPortal for this guide - no other sensors or external circuitry required!

Adafruit PyPortal - CircuitPython Powered Internet Display

$54.95
IN STOCK

Add To Cart

1 x USB Cable
USB cable - USB A to Micro-B - 3 foot long

Add To Cart

You can get both of these and a PyPortal stand in AdaBox 011:

AdaBox011 - PyPortal

OUT OF STOCK

Out Of Stock

PyPortal CircuitPython Setup

To use all the amazing features of your PyPortal with CircuitPython, you must first install a number of libraries. This page covers that process.

Adafruit CircuitPython Bundle

Download the Adafruit CircuitPython Library Bundle. You can find the latest release here:

https://adafru.it/ENC

https://adafru.it/ENC

Download the **adafruit-circuitpython-bundle-*.x-mpy-*.zip** bundle zip file where ***.x MATCHES THE VERSION OF CIRCUITPYTHON YOU INSTALLED**, and unzip a folder of the same name. Inside you'll find a **lib** folder. You have two options:

- You can add the **lib** folder to your **CIRCUITPY** drive. This will ensure you have *all the drivers*. But it will take a bunch of space on the 8 MB disk
- Add each library as you need it, this will reduce the space usage but you'll need to put in a little more effort.

At a minimum we recommend the following libraries, in fact we more than recommend. They're basically required. So grab them and install them into **CIRCUITPY/lib** now!

- **adafruit_esp32spi** - This is the library that gives you internet access via the ESP32 using (you guessed it!) SPI transport. You need this for anything Internet
- **adafruit_requests** - This library allows us to perform HTTP requests and get responses back from servers. GET/POST/PUT/PATCH - they're all in here!
- **adafruit_pyportal** - This is our friendly wrapper library that does a lot of our projects, displays graphics and text, fetches data from the internet. Nearly all of our projects depend on it!
- **adafruit_touchscreen** - a library for reading touches from the resistive touchscreen. Handles all the analog noodling, rotation and calibration for you.
- **adafruit_io** - this library helps connect the PyPortal to our free datalogging and viewing service
- **adafruit_imageload** - an image display helper, required for any graphics!
- **adafruit_display_text** - not surprisingly, it displays text on the screen
- **adafruit_bitmap_font** - we have fancy font support, and its easy to make new fonts. This library reads and parses font files.
- **adafruit_slideshow** - for making image slideshows - handy for quick display of graphics and sound
- **neopixel** - for controlling the onboard neopixel
- **adafruit_adt7410** - library to read the temperature from the on-board Analog Devices ADT7410 precision temperature sensor
- **adafruit_sdcard** - support for reading/writing data from the onboard SD card slot.
- **adafruit_bus_device** - low level support for I2C/SPI

Internet Connect!

Once you have CircuitPython setup and libraries installed we can get your board connected to the Internet. Note that access to enterprise level secured WiFi networks is not currently supported, only WiFi networks that require SSID and password.

To get connected, you will need to start by creating a *secrets file*.

What's a secrets file?

We expect people to share tons of projects as they build CircuitPython WiFi widgets. What we want to avoid is people accidentally sharing their passwords or secret tokens and API keys. So, we designed all our examples to use a `secrets.py` file, that is in your **CIRCUITPY** drive, to hold secret/private/custom data. That way you can share your main project without worrying about accidentally sharing private stuff.

Your **secrets.py** file should look like this:

```
# This file is where you keep secret settings, passwords, and tokens!
# If you put them in the code you risk committing that info or sharing it

secrets = {
    'ssid' : 'home ssid',
    'password' : 'my password',
    'timezone' : "America/New_York", # http://worldtimeapi.org/timezones
    'github_token' : 'fawfj23rakjnfawiefa',
    'hackaday_token' : 'h4xx0rs3kret',
    }
```

Inside is a python dictionary named secrets with a line for each entry. Each entry has an entry name (say `'ssid'`) and then a colon to separate it from the entry key `'home ssid'` and finally a comma ,

At a minimum you'll need the `ssid` and `password` for your local WiFi setup. As you make projects you may need more tokens and keys, just add them one line at a time. See for example other tokens such as one for accessing github or the hackaday API. Other non-secret data like your timezone can also go here, just cause it's called secrets doesn't mean you can't have general customization data in there!

For the correct time zone string, look at http://worldtimeapi.org/timezones (https://adafru.it/EcP) and remember that if your city is not listed, look for a city in the same time zone, for example Boston, New York, Philadelphia, Washington DC, and Miami are all on the same time as New York.

Of course, don't share your **secrets.py** - keep that out of GitHub, Discord or other project-sharing sites.

Connect to WiFi

OK now you have your secrets setup - you can connect to the Internet. Lets use the ESP32SPI and the Requests libraries - you'll need to visit the CircuitPython bundle and install (https://adafru.it/ENC):

- **adafruit_bus_device**
- **adafruit_esp32spi**
- **adafruit_requests**
- **neopixel**

Into your lib folder. Once that's done, load up the following example using Mu or your favorite editor:

This first connection example doesn't use a secrets file - you'll hand-enter your SSID/password to verify connectivity first! See the detailed instructions after the code below.

```
# SPDX-FileCopyrightText: 2019 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

import board
import busio
from digitalio import DigitalInOut
import adafruit_requests as requests
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
from adafruit_esp32spi import adafruit_esp32spi

# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

print("ESP32 SPI webclient test")

TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
JSON_URL = "http://api.coindesk.com/v1/bpi/currentprice/USD.json"


# If you are using a board with pre-defined ESP32 Pins:
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)

# If you have an AirLift Shield:
# esp32_cs = DigitalInOut(board.D10)
# esp32_ready = DigitalInOut(board.D7)
# esp32_reset = DigitalInOut(board.D5)

# If you have an AirLift Featherwing or ItsyBitsy Airlift:
# esp32_cs = DigitalInOut(board.D13)
# esp32_ready = DigitalInOut(board.D11)
# esp32_reset = DigitalInOut(board.D12)

# If you have an externally connected ESP32:
# NOTE: You may need to change the pins to reflect your wiring
# esp32_cs = DigitalInOut(board.D9)
# esp32_ready = DigitalInOut(board.D10)
# esp32_reset = DigitalInOut(board.D5)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)

requests.set_socket(socket, esp)

if esp.status == adafruit_esp32spi.WL_IDLE_STATUS:
    print("ESP32 found and in idle mode")
```

```
print("Firmware vers.", esp.firmware_version)
print("MAC addr:", [hex(i) for i in esp.MAC_address])

for ap in esp.scan_networks():
    print("\t%s\t\tRSSI: %d" % (str(ap["ssid"], "utf-8"), ap["rssi"]))

print("Connecting to AP...")
while not esp.is_connected:
    try:
        esp.connect_AP(secrets["ssid"], secrets["password"])
    except RuntimeError as e:
        print("could not connect to AP, retrying: ", e)
        continue
print("Connected to", str(esp.ssid, "utf-8"), "\tRSSI:", esp.rssi)
print("My IP address is", esp.pretty_ip(esp.ip_address))
print(
    "IP lookup adafruit.com: %s" % esp.pretty_ip(esp.get_host_by_name("adafruit.com"))
)
print("Ping google.com: %d ms" % esp.ping("google.com"))

# esp._debug = True
print("Fetching text from", TEXT_URL)
r = requests.get(TEXT_URL)
print("-" * 40)
print(r.text)
print("-" * 40)
r.close()

print()
print("Fetching json from", JSON_URL)
r = requests.get(JSON_URL)
print("-" * 40)
print(r.json())
print("-" * 40)
r.close()

print("Done!")
```

And save it to your board, with the name `code.py`.

As mentioned, this first connection example doesn't use a secrets file - you'll hand-enter your SSID/password to verify connectivity first!

Then go down to this line

`esp.connect_AP(b'MY_SSID_NAME', b'MY_SSID_PASSWORD')`

and change `MY_SSID_NAME` and `MY_SSID_PASSWORD` to your access point name and password, keeping them within the '' quotes. (This example doesn't use the secrets' file, but it's also very stand-alone so if other things seem to not work you can always re-load this to verify basic connectivity.) You should get something like the following:

COM61 - PuTTY

```
ESP32 SPI webclient test
ESP32 found and in idle mode
Firmware vers. bytearray(b'1.2.2\x00')
MAC addr: ['0x1', '0x5c', '0xd', '0x33', '0x4f', '0xc4']
        MicroPython-d45f8a              RSSI: -44
        adafruit_tw             RSSI: -63
        FiOS-QOG1B              RSSI: -63
        adafruit                RSSI: -71
        AP819           RSSI: -73
        FiOS-K57GI              RSSI: -74
        AP819           RSSI: -77
        linksys_SES_2868                RSSI: -79
        linksys_SES_2868                RSSI: -79
        FiOS-K57GI              RSSI: -83
Connecting to AP...
Connected to adafruit   RSSI: -65
My IP address is 10.0.1.54
IP lookup adafruit.com: 104.20.38.240
Ping google.com: 30 ms
Fetching text from http://wifitest.adafruit.com/testwifi/index.html
----------------------------------------
This is a test of the CC3000 module!
If you can read this, its working :)
----------------------------------------

Fetching json from http://api.coindesk.com/v1/bpi/currentprice/USD.json
----------------------------------------
{'time': {'updated': 'Feb 27, 2019 03:11:00 UTC', 'updatedISO': '2019-02-2
7T03:11:00+00:00', 'updateduk': 'Feb 27, 2019 at 03:11 GMT'}, 'disclaimer'
: 'This data was produced from the CoinDesk Bitcoin Price Index (USD). Non
-USD currency data converted using hourly conversion rate from openexchang
erates.org', 'bpi': {'USD': {'code': 'USD', 'description': 'United States
Dollar', 'rate_float': 3832.74, 'rate': '3,832.7417'}}}
----------------------------------------
Done!
```

In order, the example code...

Initializes the ESP32 over SPI using the SPI port and 3 control pins:

```
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)
```

Tells our `requests` library the type of socket we're using (socket type varies by connectivity type - we'll be using the `adafruit_esp32spi_socket` for this example). We'll also set the interface to an `esp` object. This is a little bit of a hack, but it lets us use `requests` like CPython does.

```
requests.set_socket(socket, esp)
```

Verifies an ESP32 is found, checks the firmware and MAC address

```
if esp.status == adafruit_esp32spi.WL_IDLE_STATUS:
    print("ESP32 found and in idle mode")
print("Firmware vers.", esp.firmware_version)
print("MAC addr:", [hex(i) for i in esp.MAC_address])
```

Performs a scan of all access points it can see and prints out the name and signal strength:

```
for ap in esp.scan_networks():
    print("\t%s\t\tRSSI: %d" % (str(ap['ssid'], 'utf-8'), ap['rssi']))
```

Connects to the AP we've defined here, then prints out the local IP address, attempts to do a domain name lookup and ping google.com to check network connectivity (note sometimes the ping fails or takes a while, this isn't a big deal)

```
    print("Connecting to AP...")
esp.connect_AP(b'MY_SSID_NAME', b'MY_SSID_PASSWORD')
print("Connected to", str(esp.ssid, 'utf-8'), "\tRSSI:", esp.rssi)
print("My IP address is", esp.pretty_ip(esp.ip_address))
print("IP lookup adafruit.com: %s" % esp.pretty_ip(esp.get_host_by_name("adafruit.com")))
print("Ping google.com: %d ms" % esp.ping("google.com"))
```

OK now we're getting to the really interesting part. With a SAMD51 or other large-RAM (well, over 32 KB) device, we can do a lot of neat tricks. Like for example we can implement an interface a lot like requests (https://adafru.it/E9o) - which makes getting data *really really easy*

To read in all the text from a web URL call `requests.get` - you can pass in `https` URLs for SSL connectivity

```
TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
print("Fetching text from", TEXT_URL)
r = requests.get(TEXT_URL)
print('-'*40)
print(r.text)
print('-'*40)
r.close()
```

Or, if the data is in structured JSON, you can get the json pre-parsed into a Python dictionary that can be easily queried or traversed. (Again, only for nRF52840, M4 and other high-RAM boards)

```
JSON_URL = "http://api.coindesk.com/v1/bpi/currentprice/USD.json"
print("Fetching json from", JSON_URL)
r = requests.get(JSON_URL)
print('-'*40)
print(r.json())
print('-'*40)
r.close()
```

Requests

We've written a requests-like (https://adafru.it/Kpa) library for web interfacing named Adafruit_CircuitPython_Requests (https://adafru.it/FpW). This library allows you to send HTTP/1.1 requests without "crafting" them and provides helpful methods for parsing the response from the server.

```
# adafruit_requests usage with an esp32spi_socket
import board
import busio
from digitalio import DigitalInOut
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
```

```
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
from adafruit_esp32spi import adafruit_esp32spi
import adafruit_requests as requests

# If you are using a board with pre-defined ESP32 Pins:
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)

# If you have an externally connected ESP32:
# esp32_cs = DigitalInOut(board.D9)
# esp32_ready = DigitalInOut(board.D10)
# esp32_reset = DigitalInOut(board.D5)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)

print("Connecting to AP...")
while not esp.is_connected:
    try:
        esp.connect_AP(b"MY_SSID_NAME", b"MY_SSID_PASSWORD")
    except RuntimeError as e:
        print("could not connect to AP, retrying: ", e)
        continue
print("Connected to", str(esp.ssid, "utf-8"), "\tRSSI:", esp.rssi)

# Initialize a requests object with a socket and esp32spi interface
requests.set_socket(socket, esp)

TEXT_URL = "http://wifitest.adafruit.com/testwifi/index.html"
JSON_GET_URL = "http://httpbin.org/get"
JSON_POST_URL = "http://httpbin.org/post"

print("Fetching text from %s" % TEXT_URL)
response = requests.get(TEXT_URL)
print("-" * 40)

print("Text Response: ", response.text)
print("-" * 40)
response.close()

print("Fetching JSON data from %s" % JSON_GET_URL)
response = requests.get(JSON_GET_URL)
print("-" * 40)

print("JSON Response: ", response.json())
print("-" * 40)
response.close()

data = "31F"
print("POSTing data to {0}: {1}".format(JSON_POST_URL, data))
response = requests.post(JSON_POST_URL, data=data)
print("-" * 40)

json_resp = response.json()
# Parse out the 'data' key from json_resp dict.
print("Data received from server:", json_resp["data"])
print("-" * 40)
response.close()
```

```
json_data = {"Date": "July 25, 2019"}
print("POSTing data to {0}: {1}".format(JSON_POST_URL, json_data))
response = requests.post(JSON_POST_URL, json=json_data)
print("-" * 40)

json_resp = response.json()
# Parse out the 'json' key from json_resp dict.
print("JSON Data received from server:", json_resp["json"])
print("-" * 40)
response.close()
```

The code first sets up the ESP32SPI interface. Then, it initializes a `request` object using an ESP32 `socket` and the `esp` object.

```
import board
import busio
from digitalio import DigitalInOut
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
from adafruit_esp32spi import adafruit_esp32spi
import adafruit_requests as requests

# If you are using a board with pre-defined ESP32 Pins:
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)

# If you have an externally connected ESP32:
# esp32_cs = DigitalInOut(board.D9)
# esp32_ready = DigitalInOut(board.D10)
# esp32_reset = DigitalInOut(board.D5)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)

print("Connecting to AP...")
while not esp.is_connected:
    try:
        esp.connect_AP(b'MY_SSID_NAME', b'MY_SSID_PASSWORD')
    except RuntimeError as e:
        print("could not connect to AP, retrying: ",e)
        continue
print("Connected to", str(esp.ssid, 'utf-8'), "\tRSSI:", esp.rssi)

# Initialize a requests object with a socket and esp32spi interface
requests.set_socket(socket, esp)
```

HTTP GET with Requests

The code makes a HTTP GET request to Adafruit's WiFi testing website
- http://wifitest.adafruit.com/testwifi/index.html (https://adafru.it/FpZ).

To do this, we'll pass the URL into `requests.get()`. We're also going to save the response *from* the server into a variable named `response`.

While we requested data from the server, we'd what the server responded with. Since we already saved the

server's `response`, we can read it back. Luckily for us, **requests automatically decodes the server's response into human-readable text,** you can read it back by calling `response.text`.

Lastly, we'll perform a bit of cleanup by calling `response.close()`. This closes, deletes, and collect's the response's data.

```
print("Fetching text from %s"%TEXT_URL)
response = requests.get(TEXT_URL)
print('-'*40)

print("Text Response: ", response.text)
print('-'*40)
response.close()
```

While some servers respond with text, some respond with json-formatted data consisting of attribute–value pairs.

CircuitPython_Requests can convert a JSON-formatted response from a server into a CPython `dict.` object.

We can also fetch and parse **json** data. We'll send a HTTP get to a url we know returns a json-formatted response (instead of text data).

Then, the code calls `response.json()` to convert the response to a CPython `dict`.

```
print("Fetching JSON data from %s"%JSON_GET_URL)
response = requests.get(JSON_GET_URL)
print('-'*40)

print("JSON Response: ", response.json())
print('-'*40)
response.close()
```

HTTP POST with Requests

Requests can also **POST data to a server by calling the `requests.post` method, passing it a `data` value.**

```
data = '31F'
print("POSTing data to {0}: {1}".format(JSON_POST_URL, data))
response = requests.post(JSON_POST_URL, data=data)
print('-'*40)

json_resp = response.json()
# Parse out the 'data' key from json_resp dict.
print("Data received from server:", json_resp['data'])
print('-'*40)
response.close()
```

You can also post json-formatted data to a server by passing `json_data` into the `requests.post` method.

```
    json_data = {"Date" : "July 25, 2019"}
print("POSTing data to {0}: {1}".format(JSON_POST_URL, json_data))
response = requests.post(JSON_POST_URL, json=json_data)
print('-'*40)

json_resp = response.json()
# Parse out the 'json' key from json_resp dict.
print("JSON Data received from server:", json_resp['json'])
print('-'*40)
response.close()
```

Advanced Requests Usage

Want to send custom HTTP headers, parse the response as raw bytes, or handle a response's http status code in your CircuitPython code?

We've written an example to show advanced usage of the requests module below.

```
import board
import busio
from digitalio import DigitalInOut
import adafruit_esp32spi.adafruit_esp32spi_socket as socket
from adafruit_esp32spi import adafruit_esp32spi
import adafruit_requests as requests

# If you are using a board with pre-defined ESP32 Pins:
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)

# If you have an externally connected ESP32:
# esp32_cs = DigitalInOut(board.D9)
# esp32_ready = DigitalInOut(board.D10)
# esp32_reset = DigitalInOut(board.D5)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)

print("Connecting to AP...")
while not esp.is_connected:
    try:
        esp.connect_AP(b"MY_SSID_NAME", b"MY_SSID_PASSWORD")
    except RuntimeError as e:
        print("could not connect to AP, retrying: ", e)
        continue
print("Connected to", str(esp.ssid, "utf-8"), "\tRSSI:", esp.rssi)

# Initialize a requests object with a socket and esp32spi interface
requests.set_socket(socket, esp)

JSON_GET_URL = "http://httpbin.org/get"

# Define a custom header as a dict.
headers = {"user-agent": "blinka/1.0.0"}

print("Fetching JSON data from %s..." % JSON_GET_URL)
response = requests.get(JSON_GET_URL, headers=headers)
print("-" * 60)

json_data = response.json()
headers = json_data["headers"]
print("Response's Custom User-Agent Header: {0}".format(headers["User-Agent"]))
print("-" * 60)

# Read Response's HTTP status code
print("Response HTTP Status Code: ", response.status_code)
print("-" * 60)

# Read Response, as raw bytes instead of pretty text
print("Raw Response: ", response.content)

# Close, delete and collect the response data
response.close()
```

WiFi Manager

That simpletest example works but it's a little finicky - you need to constantly check WiFi status and have many loops to manage connections and disconnections. For more advanced uses, we recommend using the WiFiManager object. It will wrap the connection/status/requests loop for you - reconnecting if WiFi drops, resetting the ESP32 if it gets into a bad state, etc.

Here's a more advanced example that shows the WiFi manager and also how to POST data with some extra headers:

```
# SPDX-FileCopyrightText: 2019 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

import time
import board
import busio
from digitalio import DigitalInOut
import neopixel
from adafruit_esp32spi import adafruit_esp32spi
from adafruit_esp32spi import adafruit_esp32spi_wifimanager

print("ESP32 SPI webclient test")

# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise

# If you are using a board with pre-defined ESP32 Pins:
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)

# If you have an externally connected ESP32:
# esp32_cs = DigitalInOut(board.D9)
# esp32_ready = DigitalInOut(board.D10)
# esp32_reset = DigitalInOut(board.D5)

spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)
"""Use below for Most Boards"""
status_light = neopixel.NeoPixel(
    board.NEOPIXEL, 1, brightness=0.2
)  # Uncomment for Most Boards
"""Uncomment below for ItsyBitsy M4"""
# status_light = dotstar.DotStar(board.APA102_SCK, board.APA102_MOSI, 1, brightness=0.2)
# Uncomment below for an externally defined RGB LED
# import adafruit_rgbled
# from adafruit_esp32spi import PWMOut
# RED_LED = PWMOut.PWMOut(esp, 26)
# GREEN_LED = PWMOut.PWMOut(esp, 27)
# BLUE_LED = PWMOut.PWMOut(esp, 25)
# status_light = adafruit_rgbled.RGBLED(RED_LED, BLUE_LED, GREEN_LED)
wifi = adafruit_esp32spi_wifimanager.ESPSPI_WiFiManager(esp, secrets, status_light)

counter = 0

while True:
    try:
```

```
    try:
        print("Posting data...", end="")
        data = counter
        feed = "test"
        payload = {"value": data}
        response = wifi.post(
            "https://io.adafruit.com/api/v2/"
            + secrets["aio_username"]
            + "/feeds/"
            + feed
            + "/data",
            json=payload,
            headers={"X-AIO-KEY": secrets["aio_key"]},
        )
        print(response.json())
        response.close()
        counter = counter + 1
        print("OK")
    except (ValueError, RuntimeError) as e:
        print("Failed to get data, retrying\n", e)
        wifi.reset()
        continue
    response = None
    time.sleep(15)
```

You'll note here we use a secrets.py file to manage our SSID info. The wifimanager is given the ESP32 object, secrets and a neopixel for status indication.

Note, you'll need to add a some additional information to your secrets file so that the code can query the Adafruit IO API:

- `aio_username`
- `aio_key`

You can go to your adafruit.io View AIO Key link to get those two values and add them to the secrets file, which will now look something like this:

```
# This file is where you keep secret settings, passwords, and tokens!
# If you put them in the code you risk committing that info or sharing it

secrets = {
    'ssid' : '_your_ssid_',
    'password' : '_your_wifi_password_',
    'timezone' : "America/Los_Angeles", # http://worldtimeapi.org/timezones
    'aio_username' : '_your_aio_username_',
    'aio_key' : '_your_aio_key_',
    }
```

Next, **set up an Adafruit IO feed named `test`**

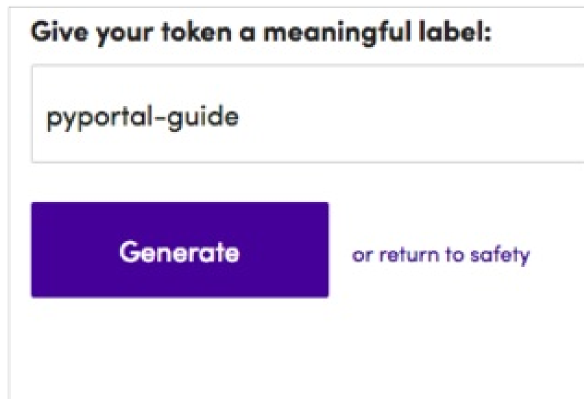
- If you do not know how to set up a feed, follow this page and come back when you've set up a feed named `test`. (https://adafru.it/f5k)

We can then have a simple loop for posting data to Adafruit IO without having to deal with connecting or initializing the hardware!

Take a look at your **test** feed on Adafruit.io and you'll see the value increase each time the CircuitPython board posts data to it!

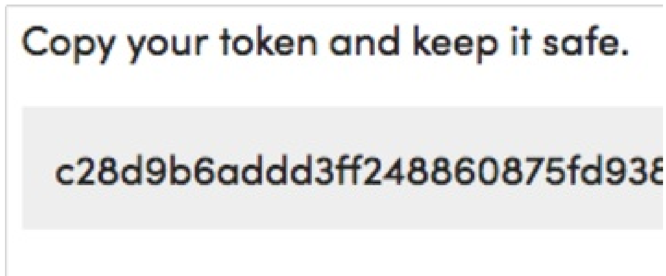
Obtain your LIFX Access Token

All requests to the LIFX bulbs require an **access token**. Obtaining one is simple!



Navigate to the LIFX account settings page (https://adafru.it/Evv), **name your access token** something unique, and **click generate**.

Then, **copy this token to a file or somewhere safe** - you will **not** be able to view it again when you navigate away from this page.



CircuitPython Library Installation

First make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/Egk) for your board.

Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafru.it/zdx) matching your version of CircuitPython. PyPortal requires at least CircuitPython version 4.0.0.

Before continuing make sure your board's **lib** folder has the following files and folders copied over.

- **adafruit_lifx**
- **adafruit_bitmap_font**
- **adafruit_bus_device**
- **adafruit_button**

- **adafruit_display_shapes**
- **adafruit_display_text**
- **adafruit_esp32spi**
- **adafruit_touchscreen**
- **neopixel**

Secrets File Setup

If you have not yet set up a **secrets.py** file in your **CIRCUITPY** drive and connected to the internet using it, follow this guide and come back to this page when you've successfully connected to the Internet (https://adafru.it/Eao).

Add the LIFX Access Token you generated to to the **secrets.py** file:

```
secrets = {
    'ssid' : '_your_wifi_ssid_',
    'password : '_your_wifi_password_',
    'lifx_token' : '_your_really_long_lifx_token_'
}
```

Add CircuitPython Code and Project Assets

In the embedded code element below, click on the **Download: Project Zip** link, and save the .zip archive file to your computer.

Then, **uncompress the .zip file**, it will unpack to a folder named **pyportal_lifx_controller**.

Copy the contents of the **pyportal_lifx_controller** directory to your PyPortal's **CIRCUITPY** drive.

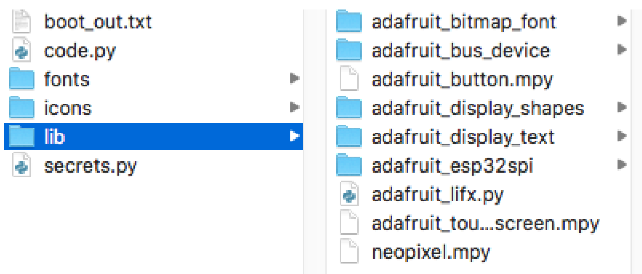
```
"""
PyPortal Smart Lighting Controller
-------------------------------------------------------------
https://learn.adafruit.com/pyportal-smart-lighting-controller

Brent Rubell for Adafruit Industries, 2019
"""
import board
import displayio
from adafruit_bitmap_font import bitmap_font
from adafruit_button import Button
import adafruit_touchscreen
from digitalio import DigitalInOut

import busio
import neopixel
from adafruit_esp32spi import adafruit_esp32spi
from adafruit_esp32spi import adafruit_esp32spi_wifimanager

# import lifx library
import adafruit_lifx

# Get wifi details and more from a secrets.py file
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
```

```
    raise

# ESP32 SPI
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)
spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)
status_light = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=0.2)
wifi = adafruit_esp32spi_wifimanager.ESPSPI_WiFiManager(esp, secrets, status_light)

# These pins are used as both analog and digital! XL, XR and YU must be analog
# and digital capable. YD just need to be digital
ts = adafruit_touchscreen.Touchscreen(board.TOUCH_XL, board.TOUCH_XR,
                                      board.TOUCH_YD, board.TOUCH_YU,
                                      calibration=((5200, 59000), (5800, 57000)),
                                      size=(320, 240))

# Set this to your LIFX personal access token in secrets.py
# (to obtain a token, visit: https://cloud.lifx.com/settings)
lifx_token = secrets['lifx_token']

# Initialize the LIFX API Helper
lifx = adafruit_lifx.LIFX(wifi, lifx_token)

# Set these to your LIFX light selector (https://api.developer.lifx.com/docs/selectors)
lifx_lights = ['label:Lamp', 'label:Bedroom']
# set default light properties
current_light = lifx_lights[0]
light_brightness = 1.0

# Make the display context
button_group = displayio.Group(max_size=20)
board.DISPLAY.show(button_group)
# preload the font
print('loading font...')
font = bitmap_font.load_font("/fonts/Arial-12.bdf")
glyphs = b'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ-,.: '
font.load_glyphs(glyphs)
# button properties
BUTTON_WIDTH = 60
BUTTON_HEIGHT = 60
buttons = []

# button fill colors (from https://api.developer.lifx.com/docs/colors)
button_colors = {'red':0xFF0000, 'white':0xFFFFFF,
                 'orange':0xFF9900, 'yellow':0xFFFF00,
                 'green':0x00FF00, 'blue':0x0000FF,
                 'purple':0x9900FF, 'pink': 0xFF00FF}

print('loading buttons...')

# list of color buttons and their properties
color_btn = [
    {'name':'red', 'pos':(15, 80), 'color':button_colors['red']},
    {'name':'white', 'pos':(85, 80), 'color':button_colors['white']},
    {'name':'orange', 'pos':(155, 80), 'color':button_colors['orange']},
    {'name':'yellow', 'pos':(225, 80), 'color':button_colors['yellow']},
    {'name':'pink', 'pos':(15, 155), 'color':button_colors['pink']},
    {'name':'green', 'pos':(85, 155), 'color':button_colors['green']},
```

```
    {'name':'blue', 'pos':(155, 155), 'color':button_colors['blue']},
    {'name':'purple', 'pos':(225, 155), 'color':button_colors['purple']}
]

# generate color buttons from color_btn list
for i in color_btn:
    button = Button(x=i['pos'][0], y=i['pos'][1],
                    width=BUTTON_WIDTH, height=BUTTON_HEIGHT, name=i['name'],
                    fill_color=i['color'], style=Button.ROUNDRECT)
    buttons.append(button)

# light property buttons and their properties
prop_btn = [
    {'name':'onoff', 'pos':(15, 15), 'label':'on/off'},
    {'name':'up', 'pos':(75, 15), 'label':'+'},
    {'name':'down', 'pos':(135, 15), 'label':'-'},
    {'name':'lamp', 'pos':(195, 15), 'label':'lamp'},
    {'name':'room', 'pos':(245, 15), 'label':'room'}
]

# generate property buttons from prop_btn list
for i in prop_btn:
    button = Button(name=i['name'], x=i['pos'][0], y=i['pos'][1],
                    width=40, height=40, label=i['label'],
                    label_font=font, style=Button.SHADOWROUNDRECT)
    buttons.append(button)

# add buttons to the group
for b in buttons:
    button_group.append(b.group)

while True:
    touch = ts.touch_point
    if touch:
        for i, button in enumerate(buttons):
            if button.contains(touch):
                button.selected = True
                if button.name == 'lamp':
                    current_light = lifx_lights[0]
                    print('Switching to ', current_light)
                elif button.name == 'room':
                    current_light = lifx_lights[1]
                    print('Switching to ', current_light)
                elif button.name == 'onoff':
                    print('Toggling {0}...'.format(current_light))
                    lifx.toggle_light(current_light)
                elif button.name == 'up':
                    light_brightness += 0.25
                    print('Setting {0} brightness to {1}'.format(current_light, light_brightness))
                    lifx.set_brightness(current_light, light_brightness)
                elif button.name == 'down':
                    light_brightness -= 0.25
                    print('Setting {0} brightness to {1}'.format(current_light, light_brightness))
                    lifx.set_brightness(current_light, light_brightness)
                else:
                    print('Setting {0} color to {1}'.format(current_light, button.name))
                    lifx.set_color(current_light, 'on', button.name, light_brightness)
                button.selected = False
            else:
```

```
            button.selected = False
```

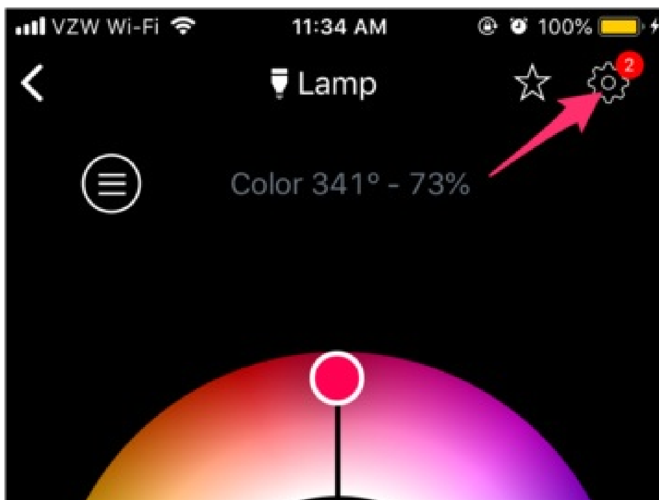
This is what the final contents of the **CIRCUITPY** drive will look like:

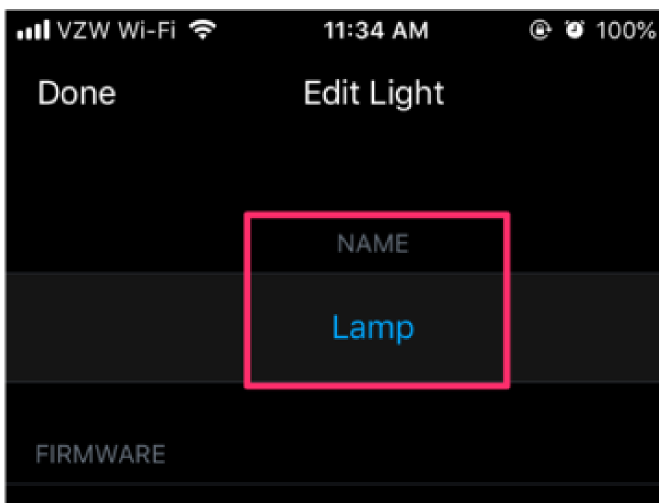


Setup Code

Before you can use the code, you'll need to modify the code.py **file to add a label specifying which light you're controlling.**

LIFX uses an identifier called a selector (https://adafru.it/Evw) to identify the lights associated with your account. With a selector, you can use the light's unique identifier, location (bedroom, workbench, living room) or lighting groups. For simplicity, this guide assumes that you will be using a label as a selector.





To find the bulb's label:

Navigate to the LIFX app and select the light you'd like to control.

Tap the gear icon to reveal the detailed settings of the light.

The **Name** of the light is the light's label.

Next, modify the `lifx_lights` variable within **code.py** to include the label from the App.

For example, if you're adding a new LIFX bulb called **workbench**, change the code *from*

`lifx_lights = ['label:lamp', 'label:Main Room']`

to

`lifx_lights = ['label:lamp', 'label:workbench']`

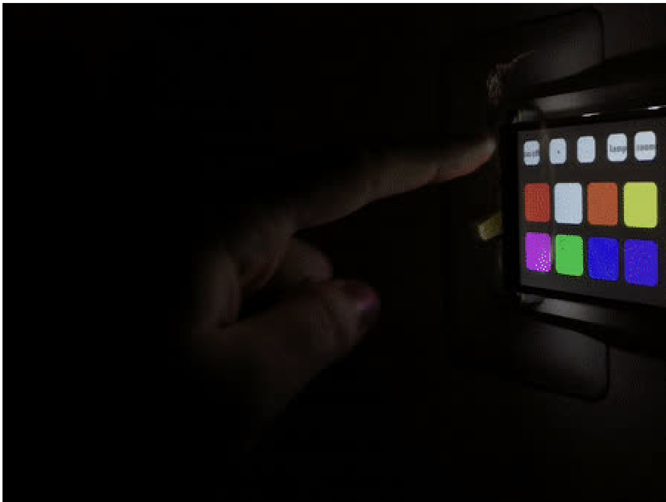
Be sure to include `label:` before the name of your light.

For more information on using LIFX light selectors, [check out their API documentation on this topic here (https://adafru.it/Evw)](https://adafru.it/Evw).

Code Usage

From the Mu Editor, **click the Serial button** to open the REPL. You should see the REPL display the code's status as it loads the interface.

Select a light by tapping the name of the light you'd like to control. If none is selected, the code will default to the first light specified within the `lifx_lights` list.

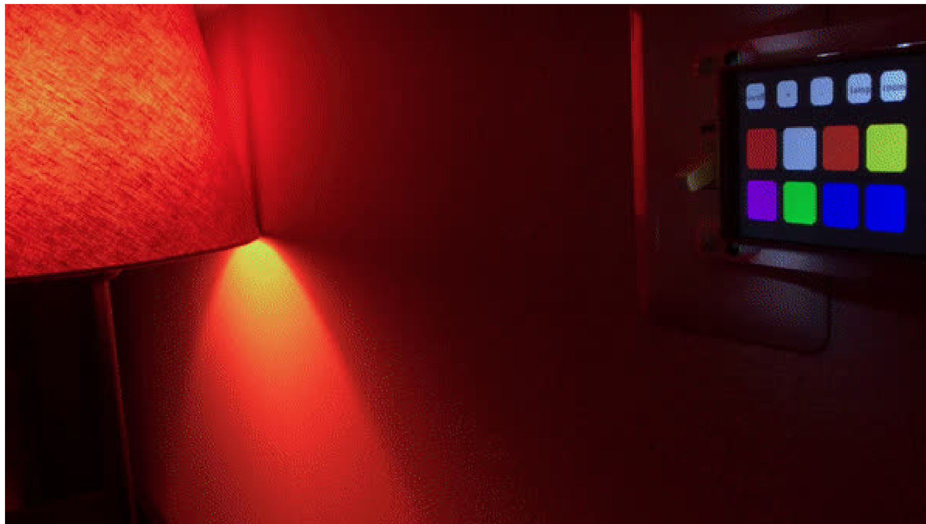


Tap the on/off button to toggle the power of the light you selected. Keep in mind that LIFX lights save state - if your light was previously set to **red** and turned off, it'll turn on with the same color.

- **Buttons invert their color while they make a request** to the LIFX API. **Once the request is complete, the button changes back to its original color.**
- **If you're having issues with your light not responding:** scroll down to the *Diagnosing Errors* section below.

Tapping a color button will set the light to the button's fill color with the brightness specified by the *+* or *-* buttons.

- Note: We used the color names for this guide to set different colors. The LIFX HTTP Remote Control API allows you to go beyond basic colors - you can set a bulb's color by specifying hue, saturation, brightness, kelvin or hex values. (https://adafru.it/Evx)



Next up, we'll take a look at the code you loaded onto your PyPortal

Importing Libraries

```
import board
import displayio
from adafruit_bitmap_font import bitmap_font
from adafruit_button import Button
import adafruit_touchscreen
from digitalio import DigitalInOut

import busio
import neopixel
from adafruit_esp32spi import adafruit_esp32spi
from adafruit_esp32spi import adafruit_esp32spi_wifimanager

# import lifx library
import adafruit_lifx
```

The code first imports all of the libraries required to run the Smart Lighting Controller.

Note that the code imports a special `adafruit_lifx` library. To communicate with the LIFX API, we wrote a CircuitPython helper module called `CircuitPython_LIFX`. This module makes HTTP requests (or, "talks") to the LIFX API server in order to interact with the bulbs.

- For more information about how HTTP works, check out our All The Internet of Things guide on this topic here (https://adafru.it/Evy).

Configuring the PyPortal

```
# ESP32 SPI
esp32_cs = DigitalInOut(board.ESP_CS)
esp32_ready = DigitalInOut(board.ESP_BUSY)
esp32_reset = DigitalInOut(board.ESP_RESET)
spi = busio.SPI(board.SCK, board.MOSI, board.MISO)
esp = adafruit_esp32spi.ESP_SPIcontrol(spi, esp32_cs, esp32_ready, esp32_reset)
status_light = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=0.2)
wifi = adafruit_esp32spi_wifimanager.ESPSPI_WiFiManager(esp, secrets, status_light)

# These pins are used as both analog and digital! XL, XR and YU must be analog
# and digital capable. YD just need to be digital
ts = adafruit_touchscreen.Touchscreen(board.TOUCH_XL, board.TOUCH_XR,
                                      board.TOUCH_YD, board.TOUCH_YU,
                                      calibration=((5200, 59000), (5800, 57000)),
                                      size=(320, 240))
```

The next chunk of code grabs information from a **secrets.py** file including wifi configuration and your LIFX Access Token. Then, it sets up the ESP32's SPI connections for use with the PyPortal. The `wifi` object is set up here too - it's used later in the code to communicate with the LIFX API.

Configuring the LIFX Helper Module

```
# Set this to your LIFX personal access token in secrets.py
# (to obtain a token, visit: https://cloud.lifx.com/settings)
lifx_token = secrets['lifx_token']

# Initialize the LIFX API Helper
lifx = adafruit_lifx.LIFX(wifi, lifx_token)

# Set these to your LIFX light selector (https://api.developer.lifx.com/docs/selectors)
lifx_lights = ['label:Lamp', 'label:Bedroom']
# set default light properties
current_light = lifx_lights[0]
light_brightness = 1.0
```

The `lifx_token` entry within secrets is set to a new variable, `lifx_token`, and passed into the LIFX helper along with the wifi object created earlier.

Then, the code sets up a list of two lights (identified by their selectors (https://adafru.it/Evw)). Later in this code, these lights are linked to buttons so the code can identify which light is being toggled.

Also, a default light brightness of 100% is set, along with a `current_light` variable used to track the light being addressed.

Button Setup

```
# Make the display context
button_group = displayio.Group(max_size=20)
board.DISPLAY.show(button_group)
# preload the font
print('loading font...')
font = bitmap_font.load_font("/fonts/Arial-12.bdf")
glyphs = b'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ-,.: '
font.load_glyphs(glyphs)
# button properties
BUTTON_WIDTH = 60
BUTTON_HEIGHT = 60
buttons = []
```

Before the code can create buttons, it needs to create a `displayio` group to hold them. Here - the code selects a font, preloads it with glyphs, and sets button properties for `BUTTON_WIDTH` and `BUTTON_HEIGHT`.

Button Generation

```
# list of color buttons and their properties
color_btn = [
    {'name':'red', 'pos':(15, 80), 'color':button_colors['red']},
    {'name':'white', 'pos':(85, 80), 'color':button_colors['white']},
    {'name':'orange', 'pos':(155, 80), 'color':button_colors['orange']},
    {'name':'yellow', 'pos':(225, 80), 'color':button_colors['yellow']},
    {'name':'pink', 'pos':(15, 155), 'color':button_colors['pink']},
    {'name':'green', 'pos':(85, 155), 'color':button_colors['green']},
    {'name':'blue', 'pos':(155, 155), 'color':button_colors['blue']},
    {'name':'purple', 'pos':(225, 155), 'color':button_colors['purple']}
]

# generate color buttons from color_btn list
for i in color_btn:
    button = Button(x=i['pos'][0], y=i['pos'][1],
                    width=BUTTON_WIDTH, height=BUTTON_HEIGHT, name=i['name'],
                    fill_color=i['color'], style=Button.ROUNDRECT)
    buttons.append(button)

# light property buttons and their properties
prop_btn = [
    {'name':'onoff', 'pos':(15, 15), 'label':'on/off'},
    {'name':'up', 'pos':(75, 15), 'label':'+'},
    {'name':'down', 'pos':(135, 15), 'label':'-'},
    {'name':'lamp', 'pos':(195, 15), 'label':'lamp'},
    {'name':'room', 'pos':(245, 15), 'label':'room'}
]

# generate property buttons from prop_btn list
for i in prop_btn:
    button = Button(name=i['name'], x=i['pos'][0], y=i['pos'][1],
                    width=40, height=40, label=i['label'],
                    label_font=font, style=Button.SHADOWROUNDRECT)
    buttons.append(button)

# add buttons to the group
for b in buttons:
    button_group.append(b.group)
```

The next chunk of code creates buttons for the light colors and properties.

First, a list of dictionary items is created corresponding to button properties. The `color_btn` list contains information for the button such as the button's name, position on the display, and hex color value. The `prop_btn` list contains the button's name, position and text label.

Then, buttons are generated from the list and appended to a `button` list. When all the buttons have been added to `button` list, they are appended to the displayio `button_group` one-by-one.

Main Loop

```
while True:
    touch = ts.touch_point
    if touch:
        for i, button in enumerate(buttons):
            if button.contains(touch):
              button.selected = True
```

The main loop checks for if the screen was touched. If it was, it searches for which button in the `button` list was touched.

Once the button was determined, the button's `selected` property is toggled - inverting the button's color until the action is performed. By doing this, you are creating a visual status indicator.

```
                if button.name == 'lamp':
                    current_light = lifx_lights[0]
                    print('Switching to ', current_light)
                elif button.name == 'room':
                    current_light = lifx_lights[1]
                    print('Switching to ', current_light)
```

Then, the code checks the button's name against its function (what it does). If the button's name is lamp or room, `current_light` is toggled to the value of the button.

Whenever the code in the main loop sends data to the LIFX API to perform an action on a light, it uses the `current_light` variable to select the light.

```
                elif button.name == 'onoff':
                    print('Toggling {0}...'.format(current_light))
                    lifx.toggle_light(current_light)
                elif button.name == 'up':
                    light_brightness += 0.25
                    print('Setting {0} brightness to {1}'.format(current_light, light_brightness))
                    lifx.set_brightness(current_light, light_brightness)
                elif button.name == 'down':
                    light_brightness -= 0.25
                    print('Setting {0} brightness to {1}'.format(current_light, light_brightness))
                    lifx.set_brightness(current_light, light_brightness)
                else:
                    print('Setting {0} color to {1}'.format(current_light, button.name))
                    lifx.set_color(current_light, 'on', button.name, light_brightness)
                button.selected = False
```

The rest of the code performs an action based on the button's name by using the `adafruit_lifx` helper module.

When the on/off button is pressed - the code calls `lifx.toggle_light` and passes in the `current_light`.

- This button corresponds to the Toggle Power endpoint defined in the LIFX remote API (https://adafru.it/Evz).

When the brightness button (+ or -) is pressed, the code passes the `current_light` and `light_brightness` into `lifx.set_brightness`. Also, the `light_brightness` increments or decrements by 1/4 each time the button is pressed.

- This button corresponds to the Set State endpoint defined in the LIFX remote API (https://adafru.it/EvA).

If the button's name doesn't correspond to a predefined name, we set the color using the button's name as the color passed into `lifx.set_color`.

- This button corresponds to the Set State endpoint defined in the LIFX remote API (https://adafru.it/EvA).
- You're not limited by the colors defined in the button_colors list - you can adjust/mix colors and send those values to the LIFX lights, check out the colors documentation on their website. (https://adafru.it/Evx)

Finally, the code sets its selected property to false, returning it to its original color to indicate an action has been performed.

Going further!

This guide has only scratched the surface of the LIFX Light Remote HTTP API. There are lots more endpoints, options, and effects to play around with. If you are interested in seeing how this works (in a deeper level) - take a look at the `adafruit_lifx.py` file located in the helper library's repository (https://adafru.it/Evu). It might be helpful to keep the LIFX API website (https://adafru.it/EvB) open in another tab, for reference.

If you end up adding a feature to the library - pull requests on the GitHub repository are appreciated!

- If you are not sure how to create a pull request, Kattni has a fantastic guide about contributing to CircuitPython using GitHub (https://adafru.it/EvC).