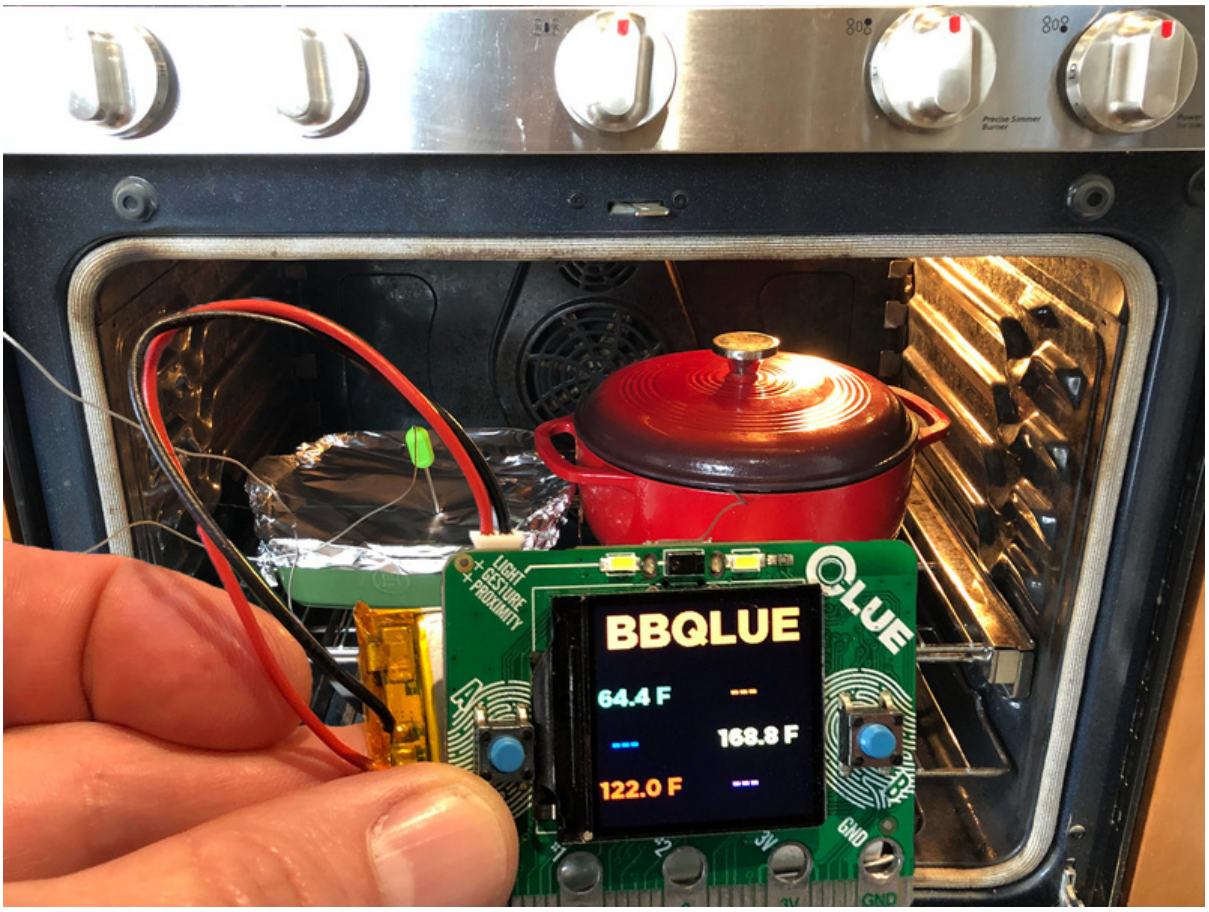
CircuitPython BLE Multi-Temperature Monitoring

Created by John Park



https://learn.adafruit.com/circuitpython-multi-temperature-ble-monitoring

Last updated on 2024-06-03 03:05:45 PM EDT

Table of Contents

Overview 3

- Parts

CircuitPython on CLUE 4

- Set up CircuitPython Quick Start!

Coding the BBQLUE 7

- Installing Project Code
- Code Explainer
- Connected

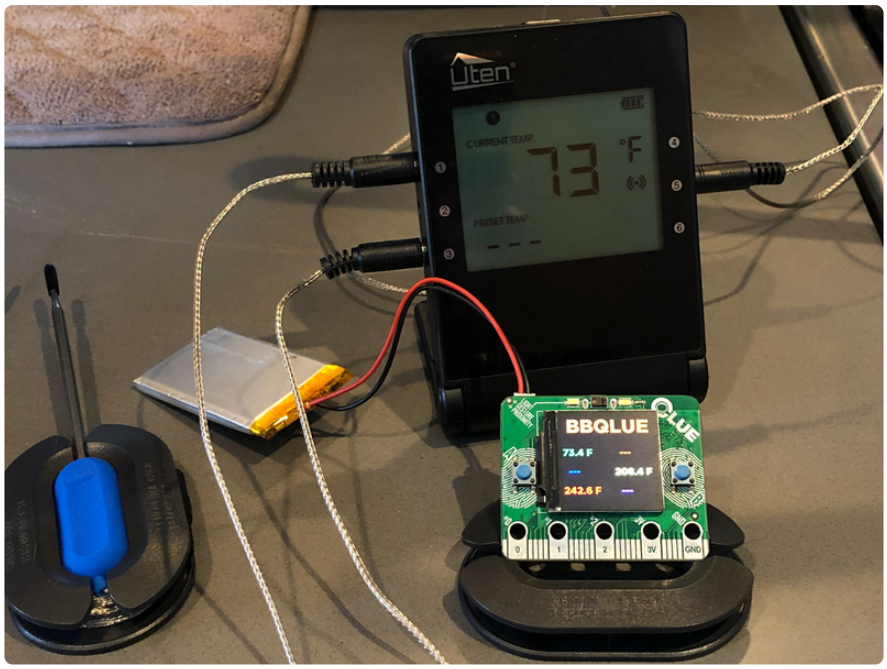
Using the BBQLUE 17

- Startup
- Probes
- Temperature Probe Display
- Unit Swap

Kitchen Science Experiment 19

- Boiling Point Elevation
- How it Works
- Experiment Materials
- Probe Prep
- Water
- Add Salt to Pot B
- Turn on the Heat
- Further Experiments

Overview



Are you doing a lot of slow cooking or smoking in the BBQ? Or perhaps you've got sourdough proofing on the counter and another in the oven? If so, you may find yourself looking to monitor a number of temperatures all at once.

Wireless Bluetooth Low Energy (BLE) temperature sensors are a great way to do just that, and now we can use the iBBQ protocol to display multiple temperature probe values simultaneously on the CLUE's display!

These affordable and easy-to-use temperature probes are designed to work with an app - but with a little CircuitPython, we can even read these probes into a microcontroller board for transfer over wifi, long term data storage, or analysis.

Parts

[Adafruit CLUE - nRF52840 Express with Bluetooth LE]
Do you feel like you just don't have a CLUE? Well, we can help with that - get a CLUE here at Adafruit by picking up this sensor-packed development board. We wanted to build some...
https://www.adafruit.com/product/4500

[USB cable - USB A to Micro-B]
This here is your standard A to micro-B USB cable, for USB 1.1 or 2.0. Perfect for connecting a PC to your Metro, Feather, Raspberry Pi or other dev-board or...
https://www.adafruit.com/product/592

In addition to the CLUE, you'll also need a BLE BBQ thermometer that utilizes the iBBQ protocol. Here are a few:

- InkBird (https://adafru.it/Kes)
- Easy BBQ from PyleUSA
- NutriChef (https://adafru.it/Ket)
- Evoland (https://adafru.it/Keu)

CircuitPython on CLUE

CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files on the **CIRCUITPY** flash drive to iterate.

The following instructions will show you how to install CircuitPython. If you've already installed CircuitPython but are looking to update it or reinstall it, the same steps work for that as well!

Set up CircuitPython Quick Start!

Follow this quick step-by-step for super-fast Python power :)

Download the latest version of CircuitPython for CLUE from circuitpython.org

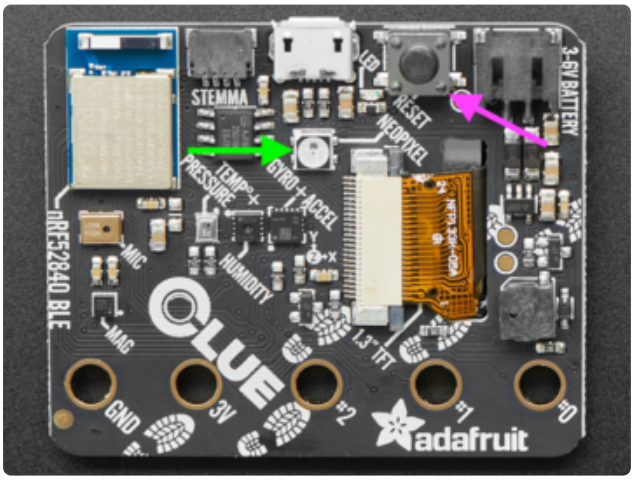
https://adafru.it/IHF

Click the link above to download the latest version of CircuitPython for the CLUE.

Download and save it to your desktop (or wherever is handy).

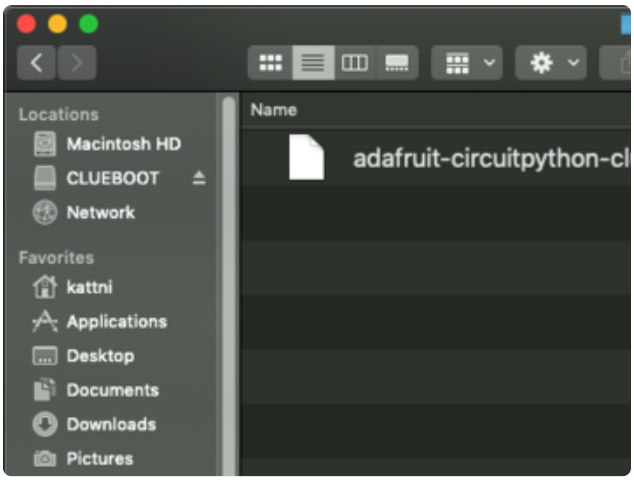
Plug your CLUE into your computer using a known-good USB cable.

A lot of people end up using charge-only USB cables and it is very frustrating! So make sure you have a USB cable you know is good for data sync.

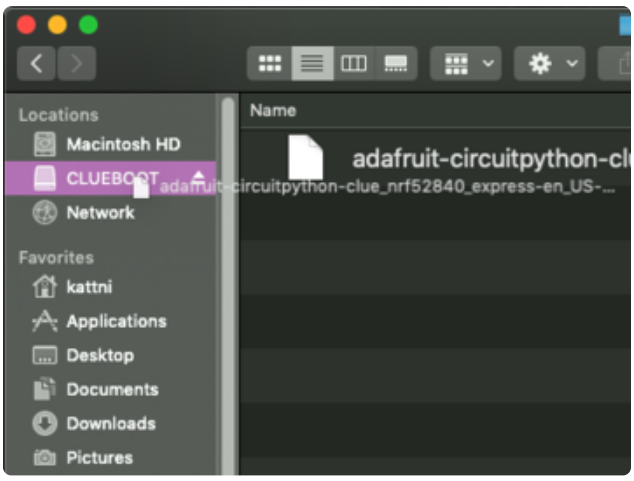


Double-click the **Reset** button on the top (magenta arrow) on your board, and you will see the NeoPixel RGB LED (green arrow) turn green. If it turns red, check the USB cable, try another USB port, etc. **Note:** The little red LED next to the USB connector will pulse red. That's ok!

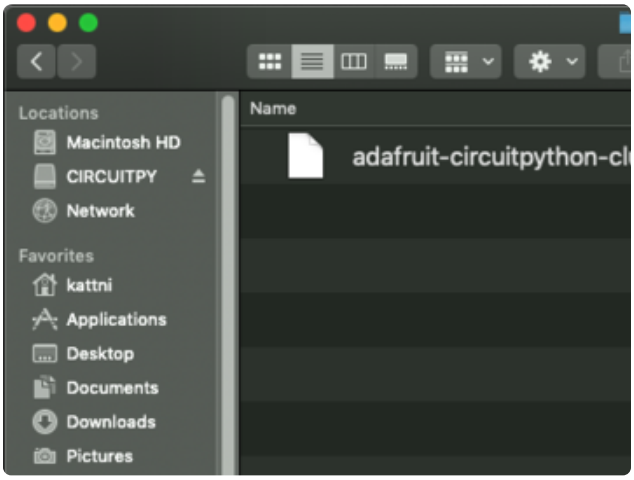
If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!



You will see a new disk drive appear called **CLUEBOOT**.



Drag the **adafruit-circuitpython-clue-etc.uf2** file to **CLUEBOOT.**



The LED will flash. Then, the **CLUEBOOT** drive will disappear and a new disk drive called **CIRCUITPY** will appear.

If this is the first time you're installing CircuitPython or you're doing a completely fresh install after erasing the filesystem, you will have two files - **boot_out.txt**, and **code.py**, and one folder - **lib** on your **CIRCUITPY** drive.

If CircuitPython was already installed, the files present before reloading CircuitPython should still be present on your **CIRCUITPY** drive. Loading CircuitPython will not create new files if there was already a CircuitPython filesystem present.

That's it, you're done! :)

Coding the BBQLUE

Here's how the CLUE will work:

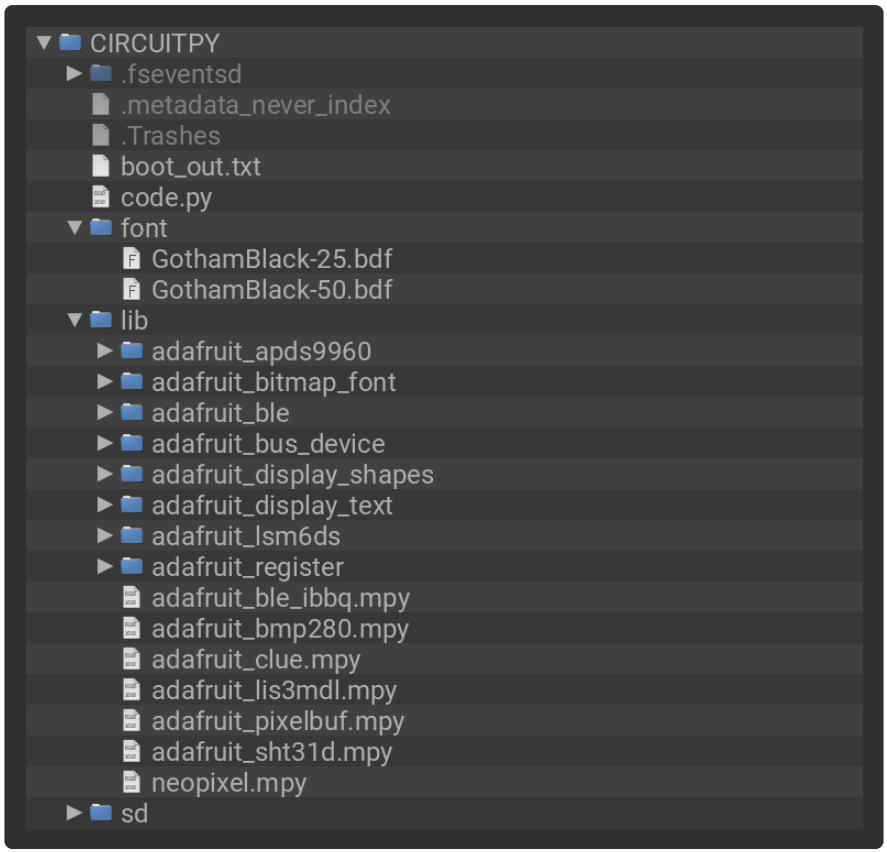
- On start, show a home screen logo with the word "BBQLUE" in a circle with a square at one corner
- Begin scanning for a BLE device advertising the ibbq service
- When such a device is found, connect to it, switch the temperatures screen layout with "BBQLUE" at the top and the probe temperatures listed in color coding
- Watch the A button for presses to switch between Fahrenheit and Celsius
- If connection is lost, swap back to home screen logo and begin scanning again

Installing Project Code

To use with CircuitPython, you need to first install a few libraries, into the lib folder on your **CIRCUITPY** drive. Then you need to update **code.py** with the example script.

Thankfully, we can do this in one go. In the example below, click the **Download Project Bundle** button below to download the necessary libraries and the **code.py** file in a zip file. Extract the contents of the zip file, open the directory **CLUE_BBQ/** and then click on the directory that matches the version of CircuitPython you're using and copy the contents of that directory to your **CIRCUITPY** drive.

Your **CIRCUITPY** drive should now look similar to the following image:




```
# SPDX-FileCopyrightText: 2020 John Park for Adafruit Industries
#
# SPDX-License-Identifier: MIT

# Adafruit BBQ display works with ibbq protocol-based BLE temperature probes

import time

import displayio
import _bleio
import adafruit_ble
from adafruit_ble.advertising.standard import ProvideServicesAdvertisement
from adafruit_ble_ibbq import IBBQService
from adafruit_clue import clue
from adafruit_display_shapes.circle import Circle
from adafruit_display_text import label
from adafruit_bitmap_font import bitmap_font

clue.display.brightness = 1.0
homescreen_screen = displayio.Group()
temperatures_screen = displayio.Group()

# define custom colors
GREEN = 0x00D929
BLUE = 0x0000FF
RED = 0xFF0000
ORANGE = 0xFF6A00
YELLOW = 0xFFFF00
PURPLE = 0xE400FF
BLACK = 0x000000
WHITE = 0xFFFFFF
BURNT = 0xBB4E00

unit_mode = False  # set the temperature unit_mode. True = centigrade, False =
farenheit

# Setup homescreen
```

```
color_bitmap = displayio.Bitmap(120, 120, 1)
color_palette = displayio.Palette(1)
color_palette[0] = BURNT
bg_sprite = displayio.TileGrid(color_bitmap, x=120, y=0, pixel_shader=color_palette)
homescreen_screen.append(bg_sprite)

clue_color = [GREEN, BLUE, RED, ORANGE, YELLOW, PURPLE]

outer_circle = Circle(120, 120, 119, fill=BLACK, outline=BURNT)
homescreen_screen.append(outer_circle)


title_font = bitmap_font.load_font("/font/GothamBlack-50.bdf")
title_font.load_glyphs("BQLUE".encode("utf-8"))
title_label = label.Label(title_font, text="BBQLUE", color=clue.ORANGE)
title_label.x = 12
title_label.y = 120
homescreen_screen.append(title_label)

clue.display.root_group = homescreen_screen

# Setup temperatures screen
temp_font = bitmap_font.load_font("/font/GothamBlack-25.bdf")
temp_font.load_glyphs("0123456789FC.-<".encode("utf-8"))

my_labels_config = [
    (0, "", GREEN, 2, 100),
    (1, "", BLUE, 2, 150),
    (2, "", RED, 2, 200),
    (3, "", ORANGE, 135, 100),
    (4, "", YELLOW, 135, 150),
    (5, "", PURPLE, 135, 200),
]

my_labels = {}  # dictionary of configured my_labels

text_group = displayio.Group()

for label_config in my_labels_config:
    (name, text, color, x, y) = label_config  # unpack a tuple into five var names
    templabel = label.Label(temp_font, text=text, color=color)
    templabel.x = x
    templabel.y = y
    my_labels[name] = templabel
    text_group.append(templabel)

temperatures_screen.append(text_group)

temp_title_label = label.Label(title_font, text="BBQLUE", color=clue.ORANGE)
temp_title_label.x = 12
temp_title_label.y = 30
temperatures_screen.append(temp_title_label)

# PyLint can't find BLERadio for some reason so special case it here.
ble = adafruit_ble.BLERadio()  # pylint: disable=no-member

ibbq_connection = None

while True:
    # re-display homescreen here
    clue.display.root_group = homescreen_screen

    print("Scanning...")
    for adv in ble.start_scan(ProvideServicesAdvertisement, timeout=5):
        clue.pixel.fill((50, 50, 0))
        if IBBQService in adv.services:
            print("found an IBBq advertisement")
            ibbq_connection = ble.connect(adv)
            print("Connected")
```

```
            break

    # Stop scanning whether or not we are connected.
    ble.stop_scan()

    try:
        if ibbq_connection and ibbq_connection.connected:
            clue.pixel.fill((0, 0, 50))
            ibbq_service = ibbq_connection[IBBQService]
            ibbq_service.init()
            while ibbq_connection.connected:

                if clue.button_a:  # hold a to swap between C and F
                    print("unit_mode swapped")
                    unit_mode = not unit_mode
                    clue.red_led = True
                    clue.play_tone(1200, 0.1)
                    clue.red_led = False
                    time.sleep(0.1)  # debounce

                temps = ibbq_service.temperatures
                batt = ibbq_service.battery_level
                if temps is not None:
                    probe_count = len(temps)  # check how many probes there are
                    for i in range(probe_count):
                        if temps[i] != 0 and temps[i] < 1000:  # unplugged probes
                            if unit_mode:
                                clue.pixel.fill((50, 0, 0))
                                temp = temps[i]
                                my_labels[i].text = "{} C".format(temp)
                                clue.pixel.fill((0, 0, 0))
                                print("Probe", i + 1, "Temperature:", temp, "C")
                            else:  # F
                                clue.pixel.fill((50, 0, 0))
                                temp = temps[i] * 9 / 5 + 32
                                my_labels[i].text = "{} F".format(temp)
                                clue.pixel.fill((0, 0, 0))
                                print("Probe", i + 1, "Temperature:", temp, "F")
                        else:
                            print(
                                "Probe", i + 1, "is unplugged",
                            )
                            my_labels[i].text = "  ---"
                    clue.display.root_group = temperatures_screen

    except _bleio.ConnectionError:
        continue
```

Code Explainer

Let's now look at how the code works.

Libraries

First, we'll import the necessary libraries, including those needed for displayio, BLE, adafruit_clue, shapes, text, fonts, and the iBBQ service.

```
import time

import displayio
import _bleio
```

```
import adafruit_ble
from adafruit_ble.advertising.standard import ProvideServicesAdvertisement
from adafruit_ble_ibbq import IBBQService
from adafruit_clue import clue
from adafruit_display_shapes.circle import Circle
from adafruit_display_text import label
from adafruit_bitmap_font import bitmap_font
```

Display Groups

We'll set up two different displayio Groups, you can think of these as two distinct screens that we and switch between. One for the homescreen logo, the other for the temperature readouts.

```
clue.display.brightness = 1.0
homescreen_screen = displayio.Group()
temperatures_screen = displayio.Group()
```

Variables

We'll use a few different colors throughout the code, so here we'll create a set of variables that give us nice names to use for specific color hex values.

We'll also create a variable called `unit_mode` to switch between Celsius and Fahrenheit.

```
GREEN = 0x00D929
BLUE = 0x0000FF
RED = 0xFF0000
ORANGE = 0xFF6A00
YELLOW = 0xFFFF00
PURPLE = 0xE400FF
BLACK = 0x000000
WHITE = 0xFFFFFF
BURNT = 0xBB4E00

unit_mode = False  # set the temperature unit_mode. True = celsius, False =
fahrenheit
```

Homescreen

Here we will set up the homescreen. This will consist of a 120x120 pixel square in the upper right corner called `bg_sprite`, in the `BURNT` umber color.

The displayio system is a hierarchy of objects with a displayio.Group at the top of the hierarchy. In this case, we append the `bg_sprite` object to the `homescreen_screen` group.

We next create a black circle with a `BURNT` outline named `outer_circle` and append it, too, too the `homescreen_screen` group.

To add the BBQLUE title, we load a bitmap font, preload the glyphs (characters) we'll need, and create a `label` object named `title_label`. The label is positioned on x and y at the center of the screen, and then it is appended to the `homescreen_screen` group.

Finally, we show all of this on the CLUE display with the `clue.display.root_group = homescreen_screen` command.

```
color_bitmap = displayio.Bitmap(120, 120, 1)
color_palette = displayio.Palette(1)
color_palette[0] = BURNT
bg_sprite = displayio.TileGrid(color_bitmap, x=120, y=0, pixel_shader=color_palette)
homescreen_screen.append(bg_sprite)

clue_color = [GREEN, BLUE, RED, ORANGE, YELLOW, PURPLE]

outer_circle = Circle(120, 120, 119, fill=BLACK, outline=BURNT)
homescreen_screen.append(outer_circle)


title_font = bitmap_font.load_font("/font/GothamBlack-50.bdf")
title_font.load_glyphs("BQLUE".encode('utf-8'))
title_label = label.Label(title_font, text="BBQLUE", color=clue.ORANGE)
title_label.x = 12
title_label.y = 120
homescreen_screen.append(title_label)

clue.display.root_group = homescreen_screen
```

Temperature Screen

The `temperature_screen` is created in much the same way as the `homescreen_screen`. However, there will be no bitmap shapes, only text.

Since we want a different position and color for each label correlated to a temperature probe, we create a list of tuples called `my_labels_config` to contain these values. Each tuple (set of multiple values) in the list represents one of the temperature probes.

By creating the `my_labels` dictionary, we can unpack the list of tuples into nicely named objects such as "name", "text", "color", and so on, instead of referring to them as indexed numbers.

We'll append each of these label objects to a text group with their corresponding colors and positions, and then append the text group to the `temperatures_screen`.

We won't show this screen yet, however. We'll wait until we've received data from a sensor.

```
temp_font = bitmap_font.load_font("/font/GothamBlack-25.bdf")
temp_font.load_glyphs("0123456789FC.-&lt;".encode('utf-8'))

my_labels_config = [(0, "", GREEN, 2, 100),
                    (1, "", BLUE, 2, 150),
                    (2, "", RED, 2, 200),
                    (3, "", ORANGE, 135, 100),
                    (4, "", YELLOW, 135, 150),
                    (5, "", PURPLE, 135, 200)
]

my_labels = {}  # dictionary of configured my_labels

text_group = displayio.Group()

for label_config in my_labels_config:
    (name, text, color, x, y) = label_config  # unpack a tuple into five var names
    templabel = label.Label(temp_font, text=text, color=color)
    templabel.x = x
    templabel.y = y
    my_labels[name] = templabel
    text_group.append(templabel)

temperatures_screen.append(text_group)

temp_title_label = label.Label(title_font, text="BBQLUE", color=clue.ORANGE)
temp_title_label.x = 12
temp_title_label.y = 30
temperatures_screen.append(temp_title_label)
```

BLE Prep

The radio is instantiated and a variable called `ibbq_connection` is created to hold the status of connections to a device advertising the ibbq service.

```
ble = adafruit_ble.BLERadio()  # pylint: disable=no-member

ibbq_connection = None
```

Main Loop

Here's the main loop of the code that repeats over and over now that things are set up.

The `homescreen_screen` is redrawn (this will be useful later when a connection is dropped).

Then, any BLE advertisements that are found are checked to see if they are advertising the ibbq service. If found, we connect!

```
while True:
    # re-display homescreen here
    clue.display.root_group = homescreen_screen

    print("Scanning...")
    for adv in ble.start_scan(ProvideServicesAdvertisement, timeout=5):
        clue.pixel.fill((50, 50, 0))
        if IBBQService in adv.services:
            print("found an IBBq advertisement")
            ibbq_connection = ble.connect(adv)
            print("Connected")
            break

    # Stop scanning whether or not we are connected.
    ble.stop_scan()
```

Connected

Once the CLUE is connected to a device with the ibbq service available, the service is initialized so it will send data.

```
try:
        if ibbq_connection and ibbq_connection.connected:
            clue.pixel.fill((0, 0, 50))
            ibbq_service = ibbq_connection[IBBQService]
            ibbq_service.init()
            while ibbq_connection.connected:
```

Button Unit Swap

In order to switch between Fahrenheit and Celsius, we set up the `clue.button_a`. It will flip the `unit_mode` between `True` and `False` each time it is pressed, as well as play a small beep on the built-in buzzer so the user knows the press has been registered.

```
if clue.button_a:  # hold a to swap between C and F
                    print("unit_mode swapped")
                    unit_mode = not unit_mode
                    clue.red_led = True
                    clue.play_tone(1200, 0.1)
                    clue.red_led = False
                    time.sleep(0.1)  # debounce
```

Temperature Display

To read and display the temperature data coming from each probe, first a variable called `temps` is set to the values of the `ibbq_service.temperatures` attribute that has been received from the BLE temperature device.

If the `temps` value is not `None` (at first when the devices connect this value is empty, so we ignore that until real data comes through) we then iterate through each probe value that has been received.

If the value isn't 0 and is less than 1000, it means there is a temperature probe plugged into that port.

Depending on the `unit_mode` we will then display the Celsius value directly, or first calculate the Fahrenheit value with the formula `* 9 / 5 +32`.

This value is displayed in the proper label color and position on screen, and then this is repeated for each subsequent probe.

If the value is 0 or greater than 1000, then there is not a probe plugged in and we'll display three dashes "---".

Once the labels are updated we re-draw the screen with the `clue.display.root_group = temperatures_screen` command.

```
temps = ibbq_service.temperatures
                batt = ibbq_service.battery_level
                if temps != None:
                    probe_count = len(temps)  # check how many probes there are
                    for i in range(probe_count):
                        if temps[i] is not 0 and temps[i] &lt; 1000:  # unplugged
probes
                            if unit_mode:
                                clue.pixel.fill((50, 0, 0))
                                temp = temps[i]
                                my_labels[i].text = "{} C".format(temp)
                                clue.pixel.fill((0, 0, 0))
                                print("Probe", i + 1, "Temperature:", temp, "C")
                            else:  # F
                                clue.pixel.fill((50, 0, 0))
                                temp = temps[i] * 9 / 5 + 32
                                my_labels[i].text = "{} F".format(temp)
                                clue.pixel.fill((0, 0, 0))
                                print("Probe", i + 1, "Temperature:", temp, "F")
                        else:
                            print(
                                "Probe", i + 1, "is unplugged",
                            )
                            my_labels[i].text = "  ---"
                    clue.display.root_group = temperatures_screen
```

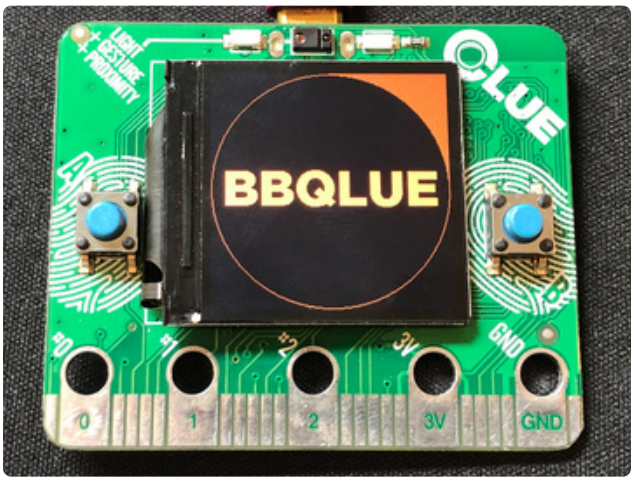
Connection Lost

If the connection is lost, we go back to the top and draw the `homescreen_screen` will attempting to find a new connection!

```
except _bleio.ConnectionError:
        continue
```

Using the BBQLUE

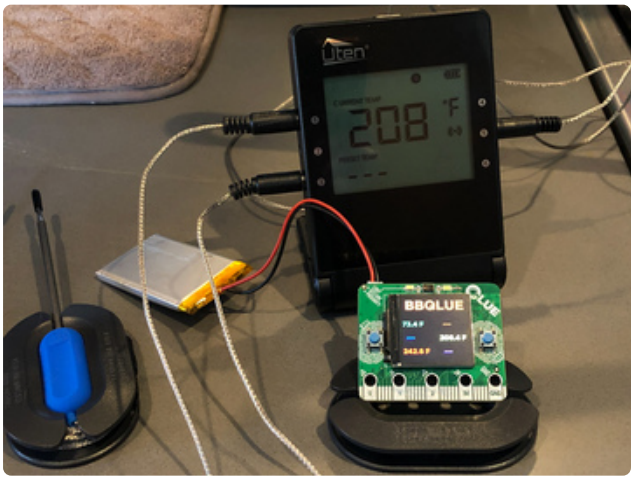
Here's how to use the BBQLUE.



Startup

When you power on the CLUE, you'll see the BBQLUE logo show up after it is done preloading the graphics and font glyphs.

It then begins searching for an iBBQ protocol BLE device being advertised.



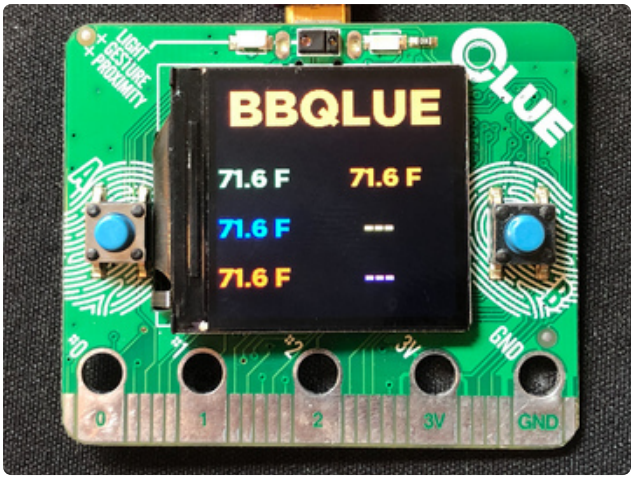
Probes

Plug in as many of the temperature probes as needed into the main unit (if your unit features multiple probes, some may only have one).

To keep the probe color coding consistent with the text color on the CLUE, use this order:

- probe 1 = green
- probe 2 = blue
- probe 3 = red
- probe 4 = orange
- probe 5 = yellow
- probe 6 = purple

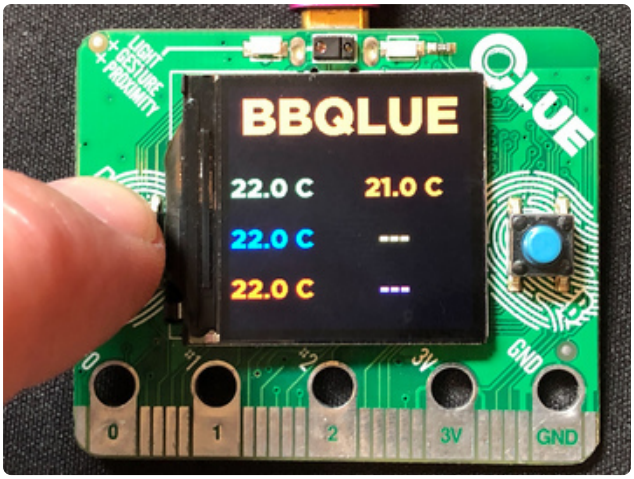
Turn on the BBQ thermometer main unit and it will begin broadcasting BLE advertisements of its iBBQ service availability.



Temperature Probe Display

The BBQLUE will then switch to the second display group and begin to display the advertised attribute values for the temperature probes.

For any probes that aren't plugged in, the value will be out of range and display as three dashes "---".



Unit Swap

By default, the temperatures are displayed in degrees Fahrenheit. Press the A button on the CLUE at any time to swap to Celsius. When you hear the beep, the button press has registered and you may let go.

Now, you can insert the temperature probes in your items that are cooking and remotely monitor the temperatures!

Kitchen Science Experiment

Which pot of water contains a secret ingredient NaCl? In this experiment, we can find out by using the CLUE temperature sensor display.

Boiling Point Elevation

Water boils at 212° Fahrenheit (100° Celsius). Or does it?!

When we say water boils at 212° F, we are talking about "typical" drinking water that comes from a tap, filter, or bottle. However, there's a common ingredient you can add to your drinking water to significantly raise its boiling point -- salt.

By dissolving enough salt in the water, you will observe a phenomenon called "boiling point elevation". You are essentially making the solution use an increased amount of heat energy to make the phase transition from liquid to gas begin.

How it Works

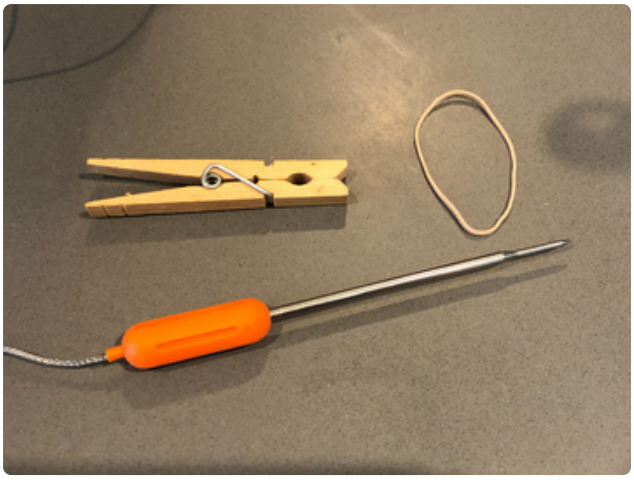
It takes heat energy to transition water from its liquid phase state to gas. When salt (a.k.a., sodium chloride or NaCl) is dissolved in water it splits into sodium and chlorine ions.

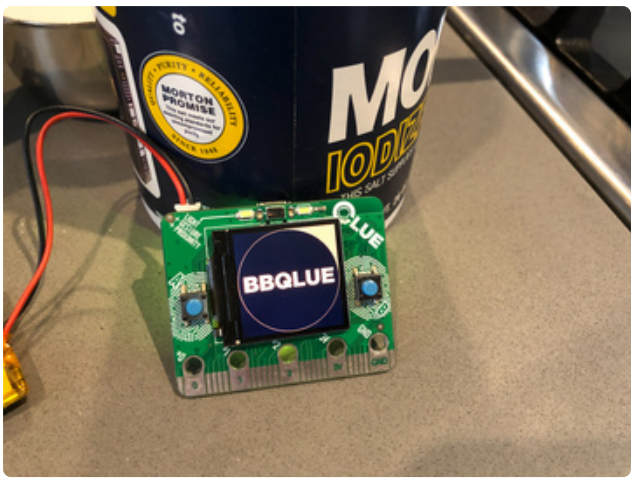
Water molecules (a.k.a., dihydrogen monoxide or H20) are a bit like magnets for each other. They have a negative side and a positive side. Breaking these bonds between molecules with heat causes the water to phase change from liquid to gas.

However, the dissolved salt has just provided some extra strength to these bonds. The sodium ions are positively charged and align with the negative oxygen side of the water molecules. The chlorine ions are negatively charged, thus align with the positively charged hydrogen side of the water molecules.

These convenient alignments strengthen the interactions between the water molecules, thus requiring more heat energy to boil.

Let's put this theory into practice and see how much we can raise the boiling point.

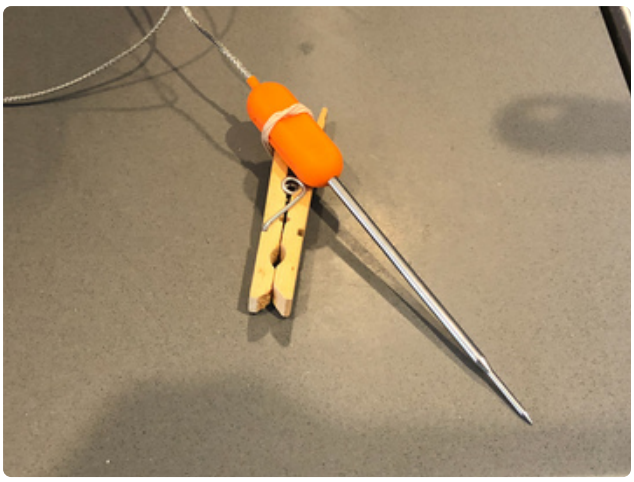




Experiment Materials

Here's what we'll need to run the experiment:

Two small pots (I used 1-1/4 Qt. pots)
Clothes pins
Rubber bands
Measuring cups
Salt
Water
Stove



Probe Prep

First, we'll create a rig to hold the probes with the metal tips near the center of the pot without touching. (We need to measure the water temperature, not the temperature of the pot.)

Wrap a rubber band a few times around the probe end and slip it over one leg of the clothespin.

Clip the clothespin to the side of the pot.

Water

Pour a quart of water into each pot. I used filtered tap water.

Add Salt to Pot B

Pot A will be the control pot, with regular water in it.

To pot B, add 1/2 C. of table salt. Stir to dissolve.

Turn on the Heat

With the pots positioned on two equal sized burners, turn up the heat to high. If you are using a gas range, make sure there is no danger of the probe, clothespin, or wire being in direct contact with the open flame.

You will see the temperatures rise on both pots on the CLUE's display. Note how the pot B is slower to raise its temperature due to the added thermal mass of the salt.

When the water is at a full boil on pot A, the temperature will read right about 212° F (100° C) and stay at that temperature. It can go no higher!

However, pot B with all of that salt will continue on well beyond -- mine went all the way up to 217.4° F (103° C)!

You have now solved the mystery of the hotter than usual boiling water!

Further Experiments

What happens if you increase or decrease the ratio of salt to water? Do other solutes behave similarly or differently?