# Table of Contents

**Overview**
- Parts
- Heart Rate Monitor
- Power
- Optional Enclosure
- Other materials

**Understanding BLE**
- BLE Basics
- Bluetooth LE Terms

**Heart Rate Service**
- Heart Rate Characteristics
- nRF Connect View

**CircuitPython on Circuit Playground Bluefruit**
- Install or Update CircuitPython

**Code the Heart Rate Display in CircuitPython**
- Code Quickstart
- Text Editor
- Code.py
- Code Explainer
- LED and NeoPixels Set Up
- Defining BPM and Connecting to a BLE Device
- Device Info
- Heart Rate and NeoPixels
- Calculating and Displaying the Heart Rate
- How long does each part last?
- The Calculations

**Heart Rate Pendant in Action**
- The Lanyard
- The Case
- Power
- Inspiration and Attribution
Overview

Are you going to a dance party soon? Or maybe a speed dating night? Would you like to wear your heart out?

In this project, you can visibly show off your heart rate in the form of a pulsing pendant. A Circuit Playground Bluefruit (CPB) connects to a heart rate monitor to display the heartbeat of the wearer. Using the CircuitPython programming language, the CPB blinks with red NeoPixels to display the heart rate.

The Circuit Playground Bluefruit will be referred to as the "CPB" for the remainder of this guide.
Parts

Heart Rate Monitor
You'll need a heart rate monitor that supports Bluetooth Low Energy (BLE). I'm using the Scosche RHYTHM+ but you should be able to use any monitor that uses the Bluetooth SIG Heart Rate service standard.

These work by flashing green (and sometimes yellow) LEDs against your skin and then measuring the reflected light that returns. The color changes/darkens during the pulse of your heart thanks to all that blood sloshing around!
Circuit Playground Bluefruit - Bluetooth Low Energy
Circuit Playground Bluefruit is our third board in the Circuit Playground series, another step towards a perfect introduction to electronics and programming. We've...
https://www.adafruit.com/product/4333

Power

Option 1: you can use a small Lithium Ion Polymer or "Lipo" battery and affix it to the back of the CPB.

![Lithium Ion Polymer Battery with Short Cable - 3.7V 420mAh](https://www.adafruit.com/product/4236)

Lithium-ion polymer (also known as 'lipo' or 'lipoly') batteries are thin, light, and powerful. The output ranges from 4.2V when completely charged to 3.7V. This...

Option 2: If you want to avoid lipos, you can use x3 AAA batteries with a battery back to get the same 3.7V to power the CPB.

![3 x AAA Battery Holder with On/Off Switch and 2-Pin JST](https://www.adafruit.com/product/727)

This battery holder connects 3 AAA batteries together in series for powering all kinds of projects. We spec'd these out because the box is slim, and 3 AAA's add up to about...

https://www.adafruit.com/product/727
Alkaline AAA batteries - 3 pack
Battery power for your portable project!
These batteries are good quality at a
good price, and work fantastic with any of
the kits or projects in the shop that use
AAA's. This is a...
https://www.adafruit.com/product/3520

Optional Enclosure

There's a great case for the CPB that also works for the Circuit Playground Express
(CPX). It is not required for this project but helps to make the NeoPixels a little more
glowy and provides protection for the electronics.

Adafruit Circuit Playground Express or
Bluefruit Enclosure
We've got nice cases for many of our
beloved boards, but the Circuit
Playground Express and
https://www.adafruit.com/product/3915

Other materials

• A lanyard or string for the pendant.
• double-sided tape for the battery.

Using metal or other conductive-based material for the lanyard may result in
shorting the CPB causing the project to not work properly. Try using a material
like plastic or a textile-based fabric like yarn.
Understanding BLE

BLE Basics

To understand how we communicate between the MagicLight Bulb and the Circuit Playground Bluefruit (CPB), it's first important to get an overview of how Bluetooth Low Energy (BLE) works in general.

The nRF52840 chip on the CPB uses Bluetooth Low Energy, or BLE. BLE is a wireless communication protocol used by many devices, including mobile devices. You can communicate between your CPB and peripherals such as the Magic Light, mobile devices, and even other CPB boards!

There are a few terms and concepts commonly used in BLE with which you may want to familiarize yourself. This will help you understand what your code is doing when you're using CircuitPython and BLE.

Two major concepts to know about are the two modes of BLE devices:

- Broadcasting mode (also called GAP for Generic Access Profile)
- Connected device mode (also called GATT for Generic Attribute Profile).

GAP mode deals with broadcasting peripheral advertisements, such as "I'm a device named LEDBlue-19592CBC", as well as advertising information necessary to establish a dedicated device connection if desired. The peripheral may also be advertising available services.

GATT mode deals with communications and attribute transfer between two devices once they are connected, such as between a heart monitor and a phone, or between your CPB and the Magic Light.
Bluetooth LE Terms

GAP Mode

Device Roles:

- Peripheral - The low-power device that broadcasts advertisements. Examples of peripherals include: heart rate monitor, smart watch, fitness tracker, iBeacon, and the Magic Light. The CPB can also work as a peripheral.
- Central - The host "computer" that observes advertisements being broadcast by the Peripherals. This is often a mobile device such as a phone, tablet, desktop or laptop, but the CPB can also act as a central (which it will in this project).

Terms:

- Advertising - Information sent by the peripheral before a dedicated connection has been established. All nearby Centrals can observe these advertisements. When a peripheral device advertises, it may be transmitting the name of the device, describing its capabilities, and/or some other piece of data. Central can look for advertising peripherals to connect to, and use that information to determine each peripheral's capabilities (or Services offered, more on that below).
GATT Mode

Device Roles:

- **Server** - In connected mode, a device may take on a new role as a Server, providing a Service available to clients. It can now send and receive data packets as requested by the Client device to which it now has a connection.
- **Client** - In connected mode, a device may also take on a new role as Client that can send requests to one or more of a Server's available Services to send and receive data packets.

**NOTE:** A device in GATT mode can take on the role of both Server and Client while connected to another device.

Terms:

- **Profile** - A pre-defined collection of Services that a BLE device can provide. For example, the Heart Rate Profile, or the Cycling Sensor (bike computer) Profile. These Profiles are defined by the Bluetooth Special Interest Group (SIG). For devices that don’t fit into one of the pre-defined Profiles, the manufacturer creates their own Profile. For example, there is not a "Smart Bulb" profile, so the Magic Light manufacturer has created their own unique one.
- **Service** - A function the Server provides. For example, a heart rate monitor armband may have separate Services for Device Information, Battery Service, and Heart Rate itself. Each Service is comprised of collections of information called Characteristics. In the case of the Heart Rate Service, the two Characteristics are Heart Rate Measurement and Body Sensor Location. The peripheral advertises its services.
- **Characteristic** - A Characteristic is a container for the value, or attribute, of a piece of data along with any associated metadata, such as a human-readable name. A characteristic may be readable, writable, or both. For example, the Heart Rate Measurement Characteristic can be served up to the Client device and will report the heart rate measurement as a number, as well as the unit string "bpm" for beats-per-minute. The Magic Light Server has a Characteristic for the RGB value of the bulb which can be written to by the Central to change the color. Characteristics each have a Universal Unique Identifier (UUID) which is a 16-bit or 128-bit ID.
- **Packet** - Data transmitted by a device. BLE devices and host computers transmit and receive data in small bursts called packets.
This guide is another good introduction to the concepts of BLE, including GAP, GATT, Profiles, Services, and Characteristics.

Heart Rate Service

The Bluetooth Special Interest Group has a standardized GATT (General Attribute Profile) for heart rate monitors called the Heart Rate profile. (You can see a list of all the GATT services here.)

This defines the commands and data that can be exchanged between the heart rate sensor device and the client device such as a phone, tablet, or BLE capable microcontroller (like we'll use in our project).

If you want to see how the Bluetooth SIG defines a GATT, such as the Heart Rate Service, you can look at the official XML file here.

Even better, run that URL through a code beautifier, such as codebeatify.org for a more human-readable version.

Heart Rate Characteristics

The Heart Rate service defines a three characteristics that can be served from the heart rate monitor (HRM) to a connected device.
Heart Rate Measurement

The most important for most needs is the Heart Rate Measurement Values characteristic which serves up the following information:

- Heart rate, in beats per minute (BPM)
- Contact -- if the device is in contact with the body or not
- Energy Expended, in kilojoules
- RR Intervals in 1024ths of seconds -- this is the measurement of intervals between beats

Not all heart monitors support all of the above characteristics, so it isn't uncommon to see 'None' returned for certain values.

Body Sensor Location

Heart rate monitors will also include a characteristic for the intended location of the monitor on the body. This is built into the sensor firmware, not something that the device is determining on the fly! Standard values include:

- Wrist
- Chest
- Finger
- Hand
- Ear Lobe
- Foot
- Other

Heart Rate Control Point

If the HRM includes the Energy Expended feature, the heart rate control point characteristic is used to allow the client device to write control points to the HRM.

nRF Connect View

We can use the nRF Connect app from Nordic on iOS and Android to connect to a heart rate monitor and look at the service, characteristics, and data.
When we first connect to the device, we can see some data advertised including the device name, available services, connection parameters, manufacturer name, revision number, and more.

In this image we can see the HRM device (RHYTHM+) has been connected, and the Heart Rate Measurement characteristic is reporting its data.

If we request a read of the Body Sensor Location characteristic, we receive it as shown here:
CircuitPython on Circuit Playground Bluefruit

Install or Update CircuitPython

Follow this quick step-by-step to install or update CircuitPython on your Circuit Playground Bluefruit.

Download the latest version of CircuitPython for this board via circuitpython.org

Click the link above and download the latest UF2 file

Download and save it to your Desktop (or wherever is handy)
Plug your Circuit Playground Bluefruit into your computer using a known-good data-capable 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 small Reset button in the middle of the CPB (indicated by the red arrow in the image). The ten NeoPixel LEDs will all turn red, and then will all turn green. If they turn all red and stay red, check the USB cable, try another USB port, etc. The little red LED next to the USB connector will pulse red - this is ok!

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

(If double-clicking doesn't do it, try a single-click!)
You will see a new disk drive appear called CPLAYBTBOOT.

Drag the adafruit_circuitpython/etc.uf2 file to CPLAYBTBOOT.

The LEDs will turn red. Then, the CPLAYBTBOOT drive will disappear and a new disk drive called CIRCUITPY will appear.

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

©Adafruit Industries
If you’d just like to get the code working fast, click "Download Project Bundle" from the code below, copy the files over to your CIRCUITPY drive and verify the file system looks the same as here. Turn on your heart rate monitor and the program should be working!

Text Editor

If you’d like to edit the code, Adafruit recommends using the Mu editor. You can get more info in [this guide](#).

Alternatively, you can use any text editor that saves files.

Code.py

If you’d like to see the messages printed in the REPL, your CPB isn't working as intended, or you’d just like to dig into the code consider the following:

Copy the code shown below, paste it into Mu. Plug your CPB into your computer via a known good USB cable. In your operating system's file explorer/finder, you should see a new flash drive named CIRCUITPY. Save the code from Mu to the Feather's CIRCUITPY drive as code.py.

```python
# SPDX-FileCopyrightText: 2022 Isaac Wellish for Adafruit Industries
#
# SPDX-License-Identifier: MIT

Circuit Playground Bluefruit BLE Heart Rate Display
Read heart rate data from a heart rate peripheral using
the standard BLE heart rate service.
```

©Adafruit Industries
The heart rate monitor connects to the CPB via BLE. LEDs on CPB blink to the heart rate of the user.

```python
import time
import board
import neopixel
import adafruit_ble
from adafruit_ble.advertising.standard import ProvideServicesAdvertisement
from adafruit_ble.services.standard.device_info import DeviceInfoService
from adafruit_ble_heart_rate import HeartRateService
from digitalio import DigitalInOut, Direction

# on-board status LED setup
red_led = DigitalInOut(board.D13)
red_led.direction = Direction.OUTPUT
red_led.value = True

# NeoPixel code
pixels = neopixel.NeoPixel(board.NEOPIXEL, 10, brightness=0.2, auto_write=False)
RED = (255, 0, 0)
LIGHTRED = (20, 0, 0)

def color_chase(color, wait):
    for i in range(10):
        pixels[i] = color
        time.sleep(wait)
        pixels.show()
        time.sleep(0.5)

# animation to show initialization of program
color_chase(RED, 0.1)  # Increase the number to slow down the color chase

# starting bpm value
bpm = 60

# PyLint can't find BLERadio for some reason so special case it here.
ble = adafruit_ble.BLERadio()    # pylint: disable=no-member
hr_connection = None

# Start with a fresh connection.
if ble.connected:
    time.sleep(1)
    for connection in ble.connections:
        if HeartRateService in connection:
            connection.disconnect()
        break

while True:
    print("Scanning...")
    red_led.value = True
    time.sleep(1)
    for adv in ble.start_scan(ProvideServicesAdvertisement, timeout=5):
        if HeartRateService in adv.services:
            print("found a HeartRateService advertisement")
            hr_connection = ble.connect(adv)
            time.sleep(2)
            print("Connected")
            red_led.value = False
            break

    # Stop scanning whether or not we are connected.
    ble.stop_scan()
    print("Stopped scan")
    red_led.value = False
    time.sleep(0.5)
```
if hr_connection and hr_connection.connected:
    print("Fetch connection")
    if DeviceInfoService in hr_connection:
        dis = hr_connection[DeviceInfoService]
        try:
            manufacturer = dis.manufacturer
        except AttributeError:
            manufacturer = "(Manufacturer Not specified)"
        try:
            model_number = dis.model_number
        except AttributeError:
            model_number = "(Model number not specified)"
        print("Device:", manufacturer, model_number)
    else:
        print("No device information")
    hr_service = hr_connection[HeartRateService]
    print("Location:", hr_service.location)

    while hr_connection.connected:
        values = hr_service.measurement_values
        print(values)  # returns the full heart_rate data set
        if values:
            bpm = (values.heart_rate)
            if values.heart_rate == 0:
                print("-")
            else:
                time.sleep(0.1)
                print(bpm)
        if bpm != 0:  # prevent from divide by zero
            # find interval time between beats
            bps = bpm / 60
            period = 1 / bps
            time_on = 0.375 * period
            time_off = period - time_on

            # Blink leds at the given BPM
            pixels.fill(RED)
            pixels.show()
            time.sleep(time_on)
            pixels.fill(LIGHTRED)
            pixels.show()
            time.sleep(time_off)

**Code Explainer**

The code is doing several important things here.

First, the necessary libraries are loaded such as `board`, `time` and `neopixel`. Additionally, a slew of various BLE support libraries are loaded.

```python
import time
import board
import neopixel
import adafruit_ble
from adafruit_ble.advertising.standard import ProvideServicesAdvertisement
from adafruit_ble_heart_rate import HeartRateService
from digitalio import DigitalInOut, Direction
```
LED and NeoPixels Set Up

Next, the on board status LED is set up.

Then the NeoPixels are initialized and a special `color_chase` function is defined then called. This has a nice effect of red leads "circling" the board when the CPB boots up.

```python
# on-board status LED setup
red_led = DigitalInOut(board.D13)
red_led.direction = Direction.OUTPUT
red_led.value = True

# NeoPixel code
pixels = neopixel.NeoPixel(board.NEOPIXEL, 10, brightness=0.2, auto_write=False)
RED = (255, 0, 0)
LIGHTRED = (20, 0, 0)

def color_chase(color, wait):
    for i in range(10):
        pixels[i] = color
        time.sleep(wait)
        pixels.show()
        time.sleep(0.5)

# animation to show initialization of program
color_chase(RED, 0.1)  # Increase the number to slow down the color chase
```

Defining BPM and Connecting to a BLE Device

Next, an example `bpm` (Beats Per Minute) level is defined which will be changed immediately when a new bpm is detected.

We scan for a BLE device with the Heart Rate Service being advertised and set the status LED to match.

When we connect, the LED turns off and we stop scanning.

```python
# starting bpm value
bpm = 60

# PyLint can't find BLERadio for some reason so special case it here.
ble = adafruit_ble.BLERadio()  # pylint: disable=no-member
hr_connection = None

# Start with a fresh connection.
if ble.connected:
    time.sleep(1)
    for connection in ble.connections:
        if HeartRateService in connection:
            connection.disconnect()
        break
```

©Adafruit Industries
while True:
    print("Scanning...")
    red_led.value = True
    time.sleep(1)

    for adv in ble.start_scan(ProvideServicesAdvertisement, timeout=5):
        if HeartRateService in adv.services:
            print("found a HeartRateService advertisement")
            hr_connection = ble.connect(adv)
            time.sleep(2)
            print("Connected")
            red_led.value = False
            break

    # Stop scanning whether or not we are connected.
    ble.stop_scan()
    print("Stopped scan")
    red_led.value = False
    time.sleep(0.5)

Device Info

With the heart rate monitor connected, the code requests info that is displayed in the
Mu REPL, if your CPB is connected to your computer over USB. This is purely
informational for curiosity and debug purposes.

```
if hr_connection and hr_connection.connected:
    print("Fetch connection")
    if DeviceInfoService in hr_connection:
        dis = hr_connection[DeviceInfoService]
        try:
            manufacturer = dis.manufacturer
        except AttributeError:
            manufacturer = "(Manufacturer Not specified)"
        try:
            model_number = dis.model_number
        except AttributeError:
            model_number = "(Model number not specified)"
        print("Device:", manufacturer, model_number)
    else:
        print("No device information")

    hr_service = hr_connection[HeartRateService]
    print("Location:", hr_service.location)
```

Heart Rate and NeoPixels

Now this is “the heart” of the program! This is the code that loops over and over while
the devices are connected.

First, we cast the heart rate service’s measurement characteristic attributes that are
sent as `values`, and then we cast the heart rate value itself as `bpm`
We'll check at first for non-zero bpm readings, as the heart rate monitor sends a few zeros at first, and we'll ignore them so nobody gets too worried, and just print dashes to the REPL.

Calculating and Displaying the Heart Rate

Next, in order to convert the BPM into blinky heartbeats, we have to do a couple of calculations. I used the code from [this Hackaday project by Dillon Nichols](https://www.hackaday.io/project/21514) to do the calculations.

The general idea is that each heartbeat consists of two parts, a **ventricular diastole**, and a **ventricular systole**.

From the below diagram, the ventricular diastole is the start of the cycle and is when the heart fills up with blood. The ventricular systole is the 2nd phase where the blood vigorously pumps and ejects the blood into the rest of the body.
How long does each part last?

In terms of time, in general, the diastole is roughly 5/8s of the total cycle and the systole is about 3/8. So if one heartbeat took 1 second (60 BPM), the diastole would last about 0.625s (5/8) and the systole would like about 0.375s (3/8).

A more nuanced explanation via Wikipedia:

At the start of the cycle, during ventricular diastole—early, the heart relaxes and expands while receiving blood into both ventricles through both atria; then, near the end of ventricular diastole—late, the two atria begin to contract (atrial systole), and each atrium pumps blood into the ventricle below it. During ventricular systole the ventricles are contracting and vigorously pulsing (or ejecting) two separated blood supplies from the heart—one to the lungs and one to all other body organs and systems—while the two atria are relaxed (atrial diastole). This precise coordination ensures that blood is efficiently collected and circulated throughout the body.

*Yes, the cardiac cycle is more complicated than just a ventricular diastole and systole. For the purposes of this guide, these elements are what we’re looking for to make the heartbeat look realistic and accurate enough with the NeoPixels.*

So with this in mind, we now need to find the time for the diastole and systole for a given BPM. These times will then be given to the NeoPixels to turn them on and off. The diastole will be the time the NeoPixels are dimmed (longer) and the systole will be the time they are brightened (shorter).

The Calculations

First, we convert BPM to beats per second. Then we find the period, which is the length of time for one heartbeat. Next, we multiply that by 3/8 or 0.375 to calculate the systole time. Lastly, the diastole is just the period - the systole.

Now we can take these values and tell the NeoPixels to brighten for the length of the systole and dim for the length of the diastole. How cool!
while hr_connection.connected:
    values = hr_service.measurement_values
    print(values)  # returns the full heart_rate data set
    if values:
        bpm = (values.heart_rate)
        if values.heart_rate == 0:
            print("-")
        else:
            time.sleep(0.1)
            print(bpm)
    if bpm != 0:  # prevent from divide by zero
        # find interval time between beats
        bps = bpm / 60
        period = 1 / bps
        time_on = 0.375 * period
        time_off = period - time_on

        # Blink leds at the given BPM
        pixels.fill(RED)
        pixels.show()
        time.sleep(time_on)
        pixels.fill(LIGHTRED)
        pixels.show()
        time.sleep(time_off)

Heart Rate Pendant in Action

The Lanyard

You may use any non-conductive material for the lanyard if you choose to string it through the holes of the CPB as shown in this guide. Yarn, hemp, twine, and plastic are all great materials that are easy to string through the CPB holes and tie into a necklace for easy wearing.
If you wish to use a conductive material for the lanyard such as a metal chain, you must ensure the material does not touch the CPB or you risk shorting the circuit and causing the CPB the malfunction or worse. One idea is you could use hot glue to glue the chain to the CPB plastic case. Be sure to use Kapton tape over the CPB pinholes to avoid a short.

Using metal or other conductive-based material for the lanyard may result in shorting the CPB causing the project to not work properly. Try using a material like plastic or a textile-based fabric like yarn.

The Case

You may want to enclose the CPB but it is not required. If you use the Adafruit Circuit Playground case, follow these steps to make sure the case fits properly:

- Place the CPB on top of the bottom piece of the case.
- Place the top of the case over the CPB, making sure the pins on the CPB align with the pins labeled on the case.
- Make sure to align the switch on the CPB with the case's switch.
- Firmly press both the top and bottom of the case. You should feel a click as the case is snapped into place.
- If the case is not firmly snapped into place, try applying pressure to the case on the JST plug side as well as the USB micro b side.
Power

You can use either a Lipo battery or a AAA battery pack with x3 AAA batteries to power the CPB and make it mobile. Just use some double-sided tape to affix the power source to the back of the CPB case (or the CPB itself if you are omitting the case).

That's it! Now go out there and show off your neat heart beat. Wear it loud, wear it proud.

Inspiration and Attribution

This project was inspired by Sidney San Martín's Heart Rate Collar build. Additionally, it uses a similar concept from Becky Stern's Heart Rate Badge guide. John Park's CircuitPython BLE Heart Rate Zone Trainer Display, was a huge help for the BLE connection code.