picozero

Release 0.4.2

Raspberry Pi Foundation

May 15, 2023

CONTENTS:

A beginner-friendly library to help you use common electronics components with the Raspberry Pi Pico.

```
from picozero import LED, Button

led = LED(1)
button = Button(2)

button.when_pressed = led.on
button.when_released = led.off
```

CHAPTER

ONE

STATUS

Beta. There will be bugs and issues. API changes are likely. More devices will be added over time.

CHAPTER

TWO

DOCUMENTATION

Documentation is available at picozero.readthedocs.io:

- Installation and getting started guide
- Recipes and how-to's
- API
- Example code

CHAPTER

THREE

CODE

The code and project is at github.com/RaspberryPiFoundation/picozero.

Issues can be raised at github.com/RaspberryPiFoundation/picozero/issues (see Contributing).

The latest distribution is available at pypi.org/project/picozero/.

CHAPTER

FOUR

THANKS

picozero is inspired by gpiozero (and reuses some of its underlying structure), but is, by design, lighter weight and aligned with the Raspberry Pi Pico. Thank you to everyone who has contributed to the gpiozero project.

CHAPTER

FIVE

TABLE OF CONTENTS

5.1 Getting started

5.1.1 Install using Thonny

Requirements

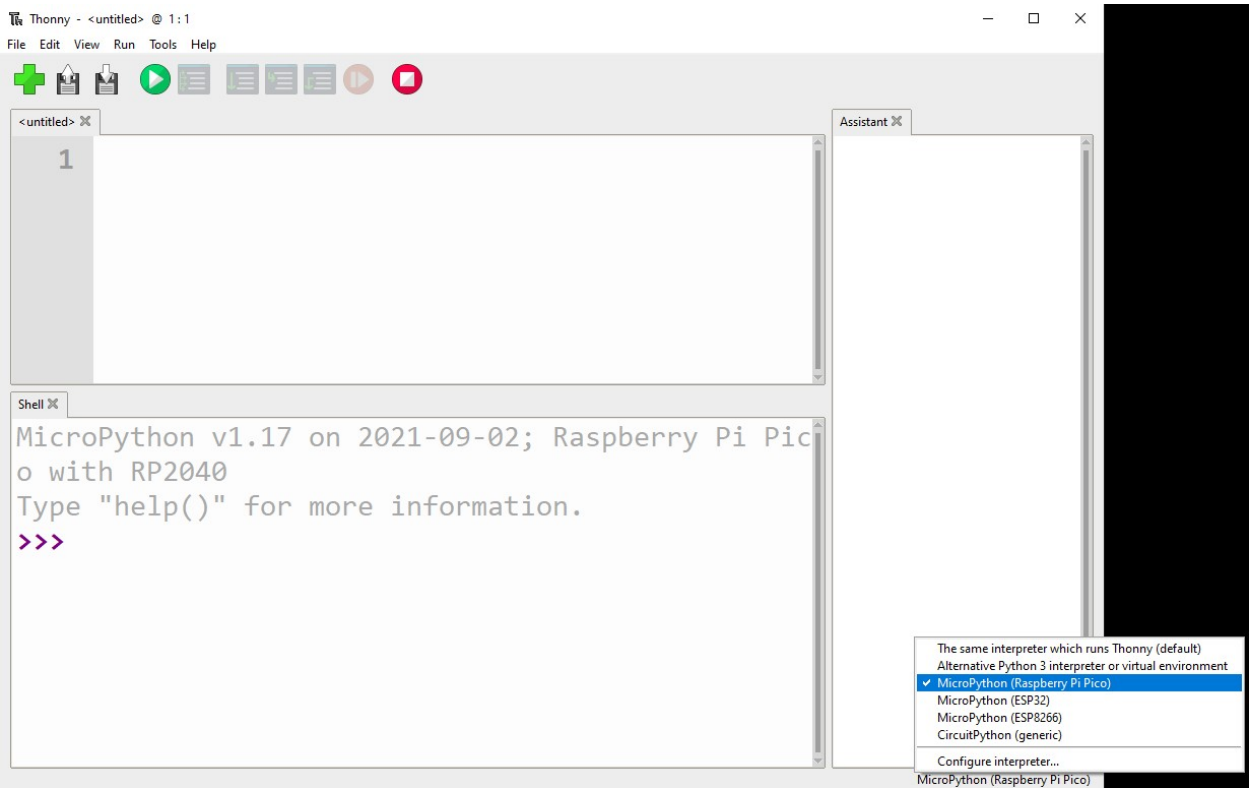
A Windows, macOS, or Linux computer with the Thonny Python IDE installed.

You can find information on how to install Thonny in the Introduction to Raspberry Pi Pico guide.

Once Thonny is installed, you will need to ensure that you are using the latest MicroPython firmware. Details on how to install or update the Raspberry Pi Pico MicroPython firmware can be found in the Pico guide.

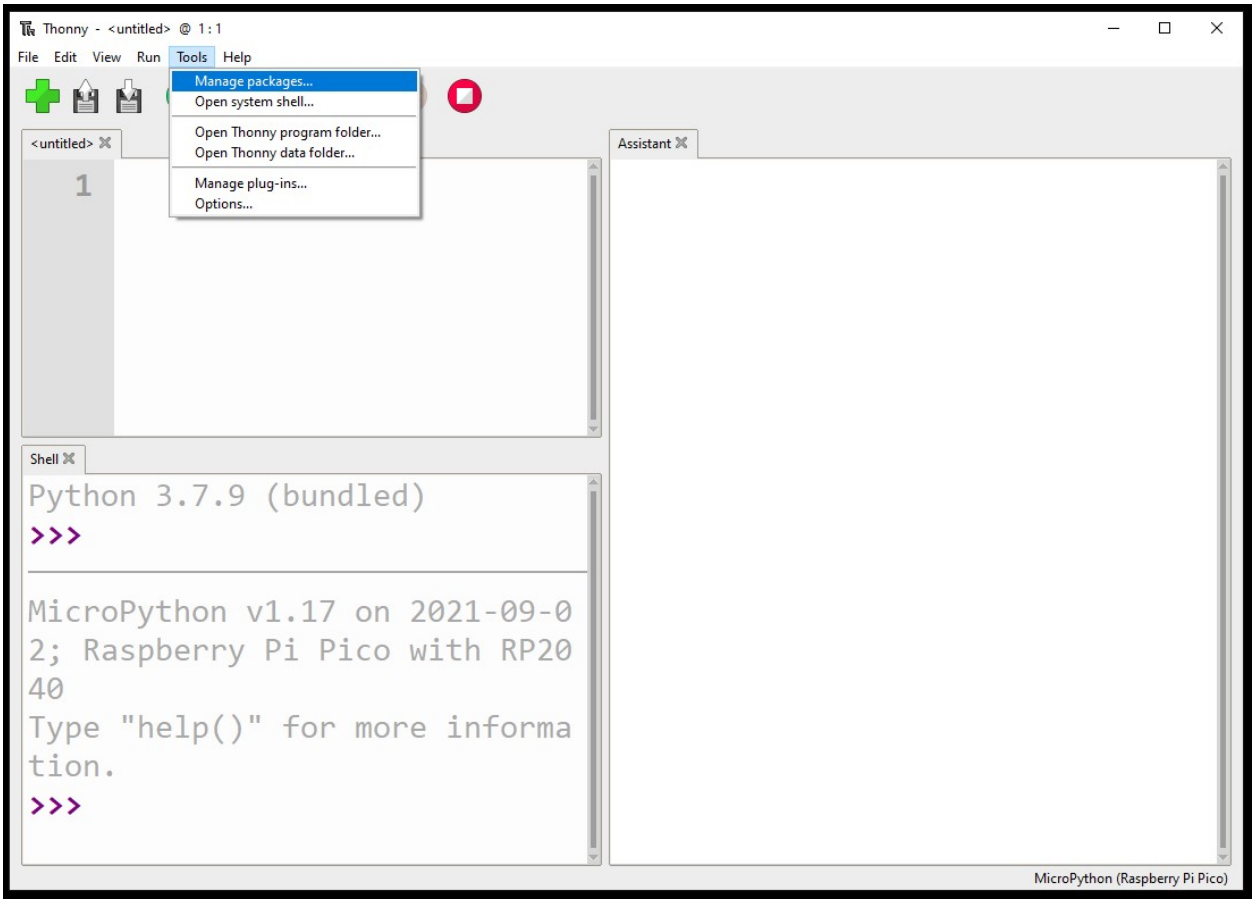
Select the MicroPython interpreter

You can change which interpreter you are using in Thonny by selecting the desired option at the bottom right of the screen. Make sure that **MicroPython (Raspberry Pi Pico)** is selected.

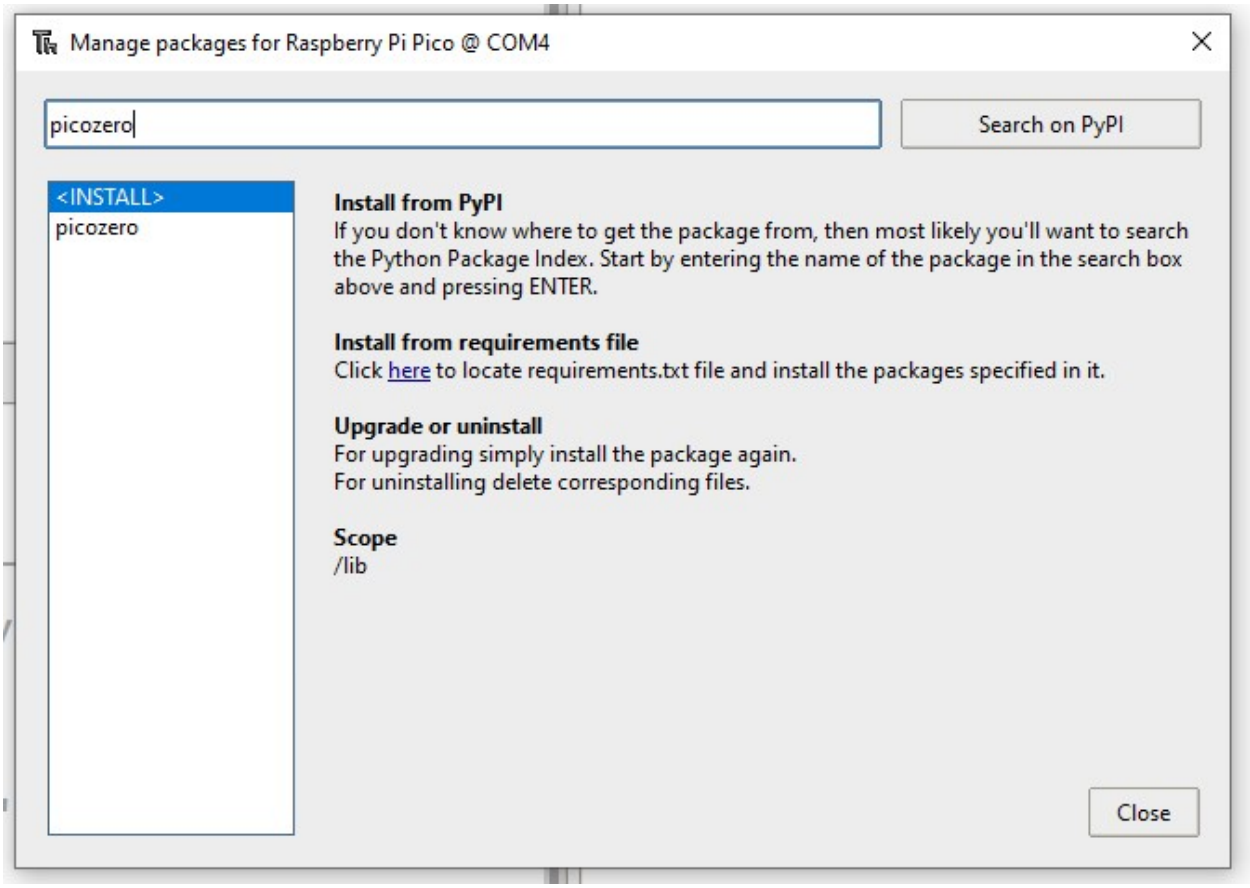


Install picozero from PyPI in Thonny

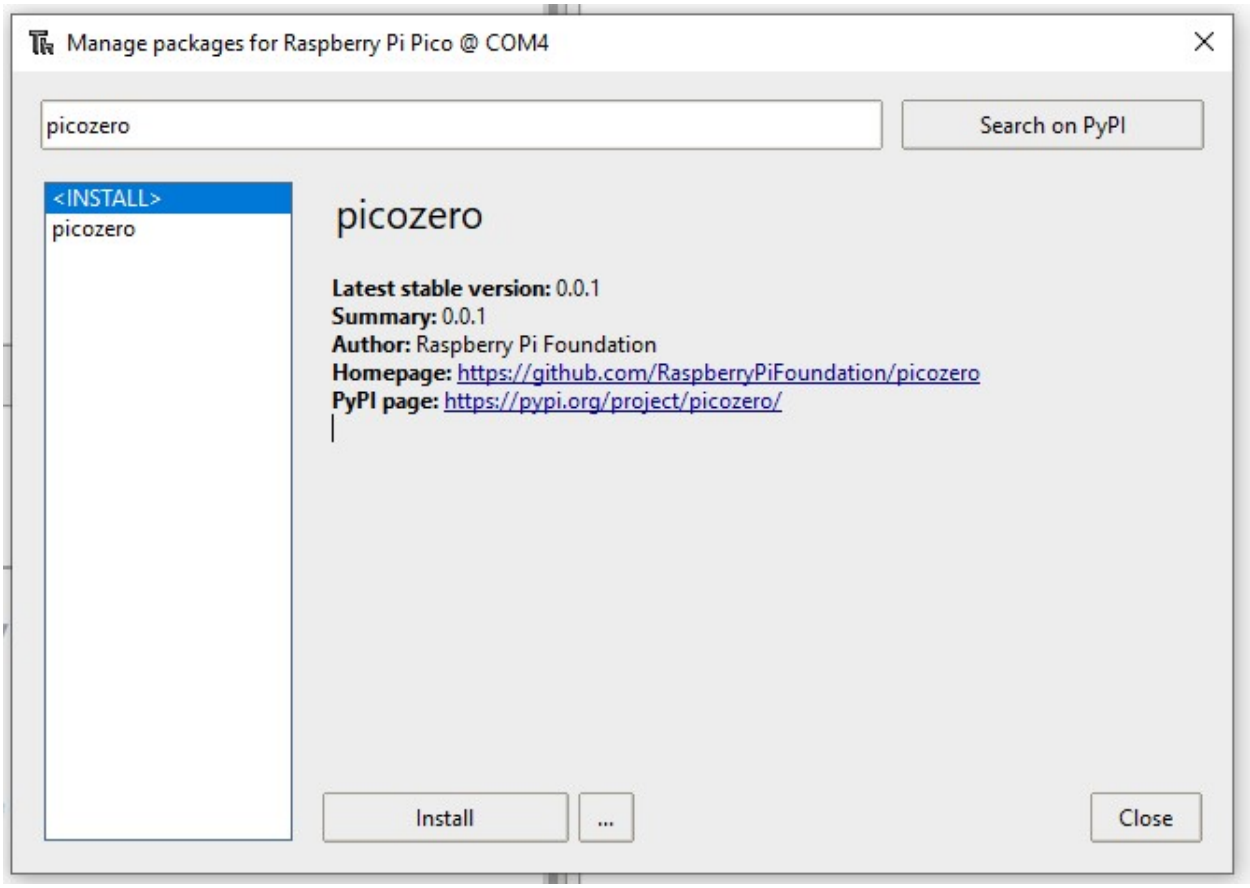
To install picozero within Thonny, select **Tools > Manage packages…**



Search for *picozero* on PyPI.



Click on **install** to download the package.



5.1.2 Manual install

picozero can be installed by copying the `picozero.py` code to your Raspberry Pi Pico.

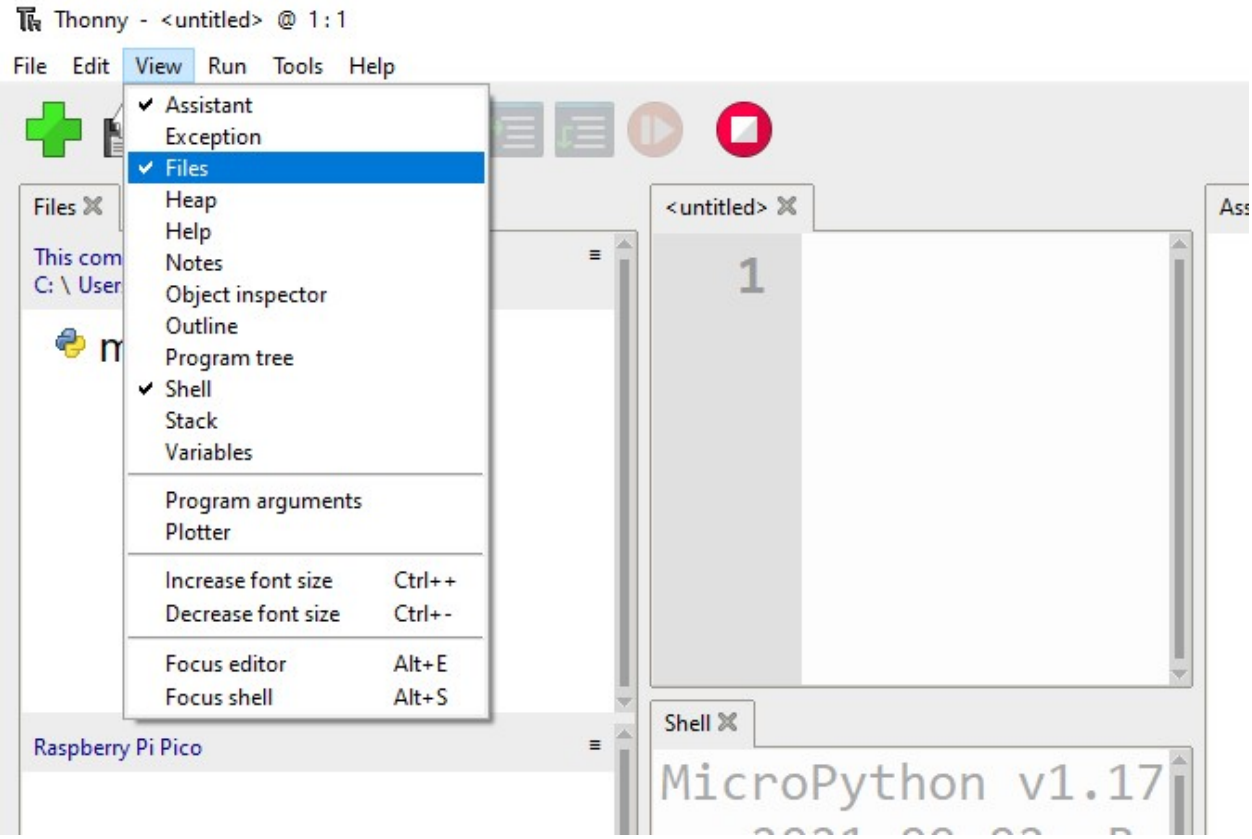
Either clone the picozero GitHub repository or copy the code from the picozero.py file and save it on your main computer.

Create a new file called picozero.py, copy code into the file and save it on your Raspberry Pi Pico.

Copy picozero.py using Thonny

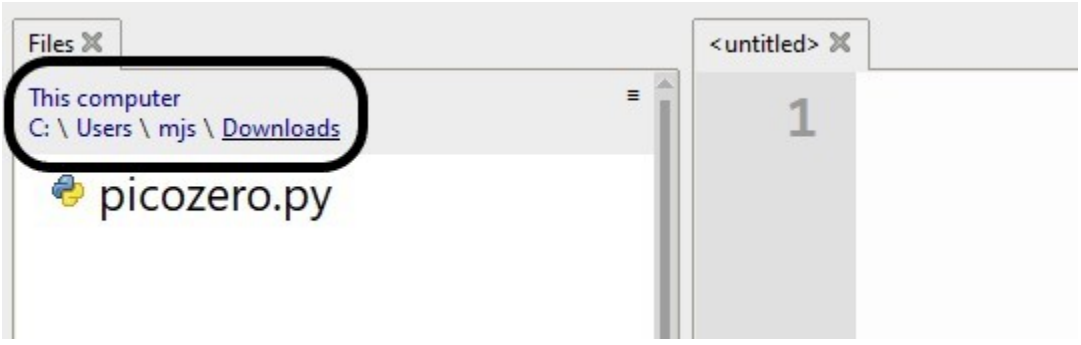
Alternatively, you can use the Thonny file manager to transfer the `picozero.py` file to your Raspberry Pi Pico.

In the **View** menu, ensure that the **Files** option has a tick. This will let you see the files.

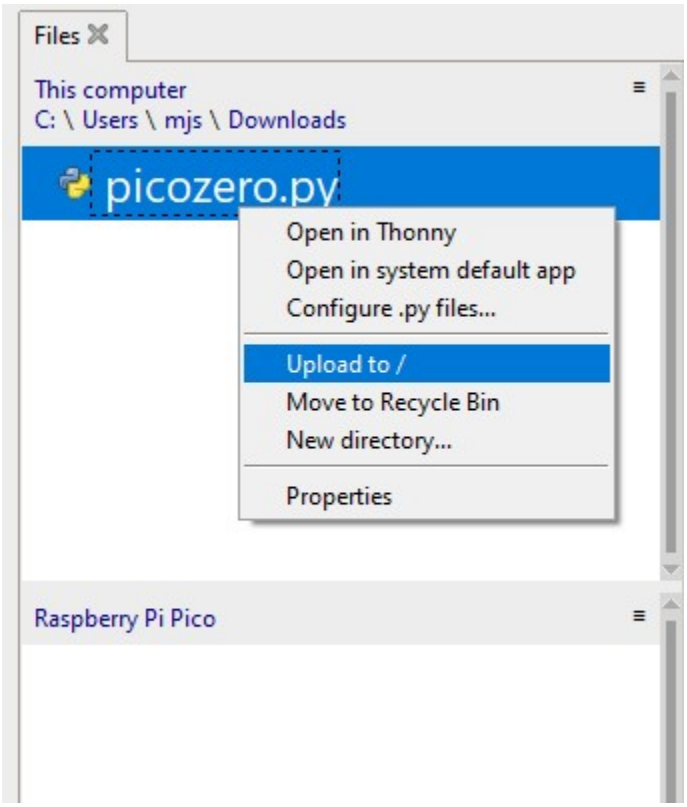


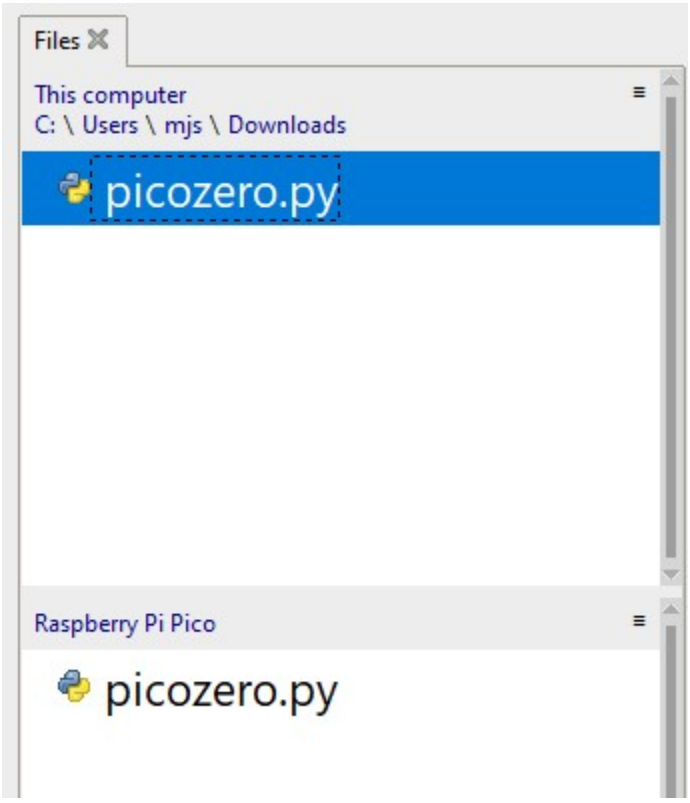
Either clone the picozero GitHub repository or copy the code from the picozero.py file and save it on your main computer.

In Thonny, navigate to the cloned directory or location you saved the file in and find the `picozero.py` file.



Right click on the file and select the **Upload to /** option. You should see a copy of the `picozero.py` file on the Raspberry Pi Pico.


Files
This computer
C: \ Users \ mjs \ Downloads
picozero.py
Open in Thonny
Open in system default app
Configure .py files...
Upload to /
Move to Recycle Bin
New directory...
Properties
Raspberry Pi Pico


Files
This computer
C: \ Users \ mjs \ Downloads
picozero.py
Raspberry Pi Pico
picozero.py

5.1.3 Write a program to control the onboard LED

The following code will blink the onboard LED at a frequency of once per second.:

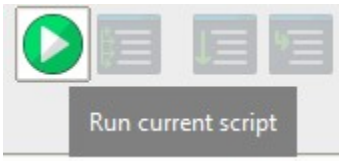
```
from picozero import pico_led
from time import sleep

while True:
    pico_led.on()
    sleep(0.5)
    pico_led.off()
    sleep(0.5)
```

Run the program on your computer

You can choose to run the program from your computer.

Click on the **Run current script** button.

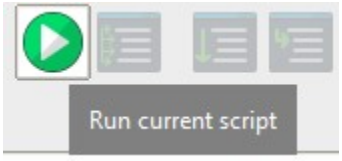


Choose to save the script on **This computer** and provide a filename.

Run the program on your Raspberry Pi Pico

You can choose to run the program from the Raspberry Pi Pico.

Click on the **Run current script** button.



Choose to save the script on **Raspberry Pi Pico** and provide a filename.

If you call the file `main.py`, it will run automatically when the Pico is powered on.

5.2 Recipes

The recipes provide examples of how you can use picozero.

5.2.1 Importing picozero

You will need add an *import* line to the top of your script to use picozero.

You can import just what you need, separating items with a comma `,`:

```
from picozero import pico_led, LED
```

Now you can use `pico_led` and *LED* in your script:

```
pico_led.on() # Turn on the LED on the Raspberry Pi Pico
led = LED(14) # Control an LED connected to pin GP14
led.on()
```

Alternatively, the whole picozero library can be imported:

```
import picozero
```

In this case, all references to picozero items must be prefixed:

```
picozero.pico_led.on()
led = picozero.LED(14)
```

5.2.2 Pico LED

To turn on the LED on your Raspberry Pi Pico:

```
from picozero import pico_led

pico_led.on()
```

Run your script to see the LED turn on.

Using the `pico_led` is equivalent to:

```
pico_led = LED(25)
```

You can use `pico_led` in the same way as external LEDs created using *LED*.

5.2.3 Pin out

You can output a *diagram* of the Raspberry Pi Pico which displays its pins and their numbers.

```
from picozero import pinout

pinout()
```

```
        ---usb---
GP0  1  |o     o| -1  VBUS
GP1  2  |o     o| -2  VSYS
GND  3  |o     o| -3  GND
GP2  4  |o     o| -4  3V3_EN
GP3  5  |o     o| -5  3V3(OUT)
GP4  6  |o     o| -6           ADC_VREF
GP5  7  |o     o| -7  GP28     ADC2
GND  8  |o     o| -8  GND      AGND
GP6  9  |o     o| -9  GP27     ADC1
GP7  10 |o     o| -10 GP26     ADC0
GP8  11 |o     o| -11 RUN
GP9  12 |o     o| -12 GP22
GND  13 |o     o| -13 GND
GP10 14 |o     o| -14 GP21
GP11 15 |o     o| -15 GP20
GP12 16 |o     o| -16 GP19
GP13 17 |o     o| -17 GP18
GND  18 |o     o| -18 GND
GP14 19 |o     o| -19 GP17
GP15 20 |o     o| -20 GP16
        ---------
```

5.2.4 LEDs

You can control external LEDs with a Raspberry Pi Pico.

Flash

Turn an *LED* on and off:

```
from picozero import LED
from time import sleep

led = LED(14)

led.on()
sleep(1)
led.off()
```

Toggle an *LED* to turn it from on to off or off to on:

```
from picozero import LED
from time import sleep

led = LED(14)

while True:
    led.toggle()
    sleep(1)
```

Alternatively, you can use the `blink()` method.

```
from picozero import LED

led = LED(14)

led.blink()
```

Brightness

Set the brightness of an *LED*:

```
from picozero import LED
from time import sleep

led = LED(14)

while True:
    led.brightness = 0  # off
    sleep(1)
    led.brightness = 0.5  # half brightness
```

(continues on next page)

(continued from previous page)

```
    sleep(1)
    led.brightness = 1  # full brightness
    sleep(1)
```

Create a pulse effect:

```
from picozero import LED
from time import sleep
from math import sin, radians

led = LED(14)

while True:
    for i in range(360):
        angle = radians(i)
        led.brightness = 0.5 + 0.5 * sin(angle)
        sleep(0.01)
```

Alternatively, you can use the `pulse()` method.

```
from picozero import LED

led = LED(14)

led.pulse()
```

5.2.5 Buttons

You can connect buttons and switches to a Raspberry Pi Pico and detect when they are pressed.

Check if a *Button* is pressed:

```
from picozero import Button
from time import sleep

button = Button(18)

while True:
    if button.is_pressed:
        print("Button is pressed")
    else:
        print("Button is not pressed")
    sleep(0.1)
```

Run a function every time a *Button* is pressed:

```
from picozero import Button, pico_led
from time import sleep

button = Button(18)

def led_on_off():
```

(continues on next page)

(continued from previous page)

```
    pico_led.on()
    sleep(1)
    pico_led.off()

button.when_pressed = led_on_off
```

Note: The line `button.when_pressed = led_on_off` does not run the function `led_on_off`, rather it creates a reference to the function to be called when the button is pressed. Accidental use of `button.when_pressed = led_on_off()` would set the `when_pressed` action to `None` (the return value of this function), which would mean nothing happens when the button is pressed.

Turn the `pico_led` on when a *Button* is pressed and off when it is released:

```
from picozero import Button, pico_led

button = Button(18)

button.when_pressed = pico_led.on
button.when_released = pico_led.off
```

5.2.6 RGB LEDs

Set colours with an *RGBLED*:

```
from picozero import RGBLED
from time import sleep

rgb = RGBLED(red=2, green=1, blue=0)

rgb.red = 255  # full red
sleep(1)
rgb.red = 128  # half red
sleep(1)

rgb.on() # white

rgb.color = (0, 255, 0)  # full green
sleep(1)
rgb.color = (255, 0, 255)  # magenta
sleep(1)
rgb.color = (255, 255, 0)  # yellow
sleep(1)
rgb.color = (0, 255, 255)  # cyan
sleep(1)
rgb.color = (255, 255, 255)  # white
sleep(1)

rgb.color = (0, 0, 0)  # off
sleep(1)
```

(continues on next page)

(continued from previous page)

```
# slowly increase intensity of blue
for n in range(255):
    rgb.blue = n
    sleep(0.01)

rgb.off()
```

Use *toggle()* and *invert()*:

```
from picozero import RGBLED
from time import sleep

rgb = RGBLED(red=2, green=1, blue=0)

rgb.color = (255, 165, 0) # orange
sleep(1)

for _ in range(6):
    rgb.toggle()
    sleep(1)

for _ in range(6):
    rgb.invert()
    sleep(1)

rgb.off()
```

Blink

Use *blink()* blink to change between colours. You can control which colours are used and how long the LED is set to each colour. The colour *(0, 0, 0)* represents off.

You can control whether *blink()* runs a fixed number of times and whether it waits until it has finished or returns immediately so other code can run.

```
from picozero import RGBLED
from time import sleep

rgb = RGBLED(1, 2, 3)

rgb.blink() # does not wait
sleep(6)
rgb.off()
sleep(1)

# blink purple 2 seconds, off 0.5 seconds
rgb.blink(on_times=(2, 0.5), colors=((1, 0, 1), (0, 0, 0)), wait=True, n=3)

rgb.off()
sleep(1)
```

(continues on next page)

(continued from previous page)

```
# blink red 1 second, green 0.5 seconds, blue 0.25 seconds
rgb.blink((1, 0.5, 0.25), colors=((1, 0, 0), (0, 1, 0), (0, 0, 1)), wait=True, n=2)
```

Pulse

Use *pulse()* to gradually change the LED colour. The default will pulse between red and off, then green and off, then blue and off.

```
from picozero import RGBLED
from time import sleep

rgb = RGBLED(1, 2, 3)

rgb.pulse() # does not wait
sleep(6)
rgb.off()
sleep(1)

# 2 second to fade from purple to off, 0.5 seconds to change from off to purple
rgb.pulse(fade_times=(2, 0.5), colors=((1, 0, 1), (0, 0, 0)), wait=True, n=3)

rgb.off()
sleep(1)

# 4 seconds to change from red to green, 2 to change from green to blue, then 1 to␣
↪change from blue back to red
rgb.pulse((4, 2, 1), colors=((1, 0, 0), (0, 1, 0), (0, 0, 1)), wait=True, n=2)
```

Cycle

The default for *cycle()* is to cycle from red to green, then green to blue, then blue to red.

```
from picozero import RGBLED
from time import sleep

rgb = RGBLED(1, 2, 3)

# Gradually colour cycle through colours between red and green, green and blue then blue␣
↪and red
rgb.cycle()
sleep(4)
rgb.off()
sleep(1)

# Colour cycle slower in the opposite direction
rgb.cycle(fade_times=3, colors=((0, 0, 1), (0, 1, 0), (1, 0, 0)), wait=True, n=2)
rgb.off()
```

5.2.7 Potentiometer

Print the value, voltage, and percent reported by a potentiometer:

```
# Potentiometer connected to GP26 (ADC0), GND and 3V

from time import sleep
from pico import Pot

pot = Pot(26)

while True:
    print(pot.value, pot.voltage)
    sleep(0.1)
```

Note: In the Thonny Python editor, choose **View** > **Plotter** to plot the output of `print()`.

Use a potentiometer to control the brightness of an LED:

```
from picozero import Pot, LED

# Potentiometer connected to GP26 (ADC0), GND and 3V
# LED connected to GP0

pot = Pot(26)
led =  LED(0)

while True:
    led.value = pot.value
```

5.2.8 Buzzer

Control an active buzzer that plays a note when powered:

```
# Active Buzzer that plays a note when powered
from time import sleep
from picozero import Buzzer

buzzer = Buzzer(10)

buzzer.on()
sleep(1)
buzzer.off()
sleep(1)

buzzer.beep()
sleep(4)
buzzer.off()
```

5.2.9 Speaker

Control a passive buzzer or speaker that can play different tones or frequencies:

```
from picozero import Speaker
from time import sleep

speaker = Speaker(5)

def tada():
    c_note = 523
    speaker.play(c_note, 0.1)
    sleep(0.1)
    speaker.play(c_note, 0.9)

def chirp():
    global speaker
    for _ in range(5):
        for i in range(5000, 2999, -100):
          speaker.play(i, 0.01)
        sleep(0.2)


try:
    tada()
    sleep(1)
    chirp()

finally: # Turn the speaker off if interrupted
    speaker.off()
```

Play a tune

Play a tune of note names and durations in beats:

```
from picozero import Speaker

speaker = Speaker(5)

BEAT = 0.25 # 240 BPM

liten_mus = [ ['d5', BEAT / 2], ['d#5', BEAT / 2], ['f5', BEAT], ['d6', BEAT], ['a#5',␣
↪BEAT], ['d5', BEAT],
              ['f5', BEAT], ['d#5', BEAT], ['d#5', BEAT], ['c5', BEAT / 2],['d5', BEAT /␣
↪2], ['d#5', BEAT],
              ['c6', BEAT], ['a5', BEAT], ['d5', BEAT], ['g5', BEAT], ['f5', BEAT], ['f5
↪', BEAT], ['d5', BEAT / 2],
              ['d#5', BEAT / 2], ['f5', BEAT], ['g5', BEAT], ['a5', BEAT], ['a#5', BEAT],
↪ ['a5', BEAT], ['g5', BEAT],
              ['g5', BEAT], ['', BEAT / 2], ['a#5', BEAT / 2], ['c6', BEAT / 2], ['d6',␣
↪BEAT / 2], ['c6', BEAT / 2],
              ['a#5', BEAT / 2], ['a5', BEAT / 2], ['g5', BEAT / 2], ['a5', BEAT / 2], [
```

(continues on next page)

(continued from previous page)

```
↪'a#5', BEAT / 2], ['c6', BEAT],
              ['f5', BEAT], ['f5', BEAT], ['f5', BEAT / 2], ['d#5', BEAT / 2], ['d5',␣
↪BEAT], ['f5', BEAT], ['d6', BEAT],
              ['d6', BEAT / 2], ['c6', BEAT / 2], ['b5', BEAT], ['g5', BEAT], ['g5',␣
↪BEAT], ['c6', BEAT / 2],
              ['a#5', BEAT / 2], ['a5', BEAT], ['f5', BEAT], ['d6', BEAT], ['a5', BEAT],␣
↪['a#5', BEAT * 1.5]]

try:
    speaker.play(liten_mus)

finally: # Turn speaker off if interrupted
    speaker.off()
```

Play individual notes

Play individual notes and control the timing or perform another action:

```
from picozero import Speaker
from time import sleep

speaker = Speaker(5)

BEAT = 0.4

liten_mus = [ ['d5', BEAT / 2], ['d#5', BEAT / 2], ['f5', BEAT], ['d6', BEAT], ['a#5',␣
↪BEAT], ['d5', BEAT],
              ['f5', BEAT], ['d#5', BEAT], ['d#5', BEAT], ['c5', BEAT / 2],['d5', BEAT /␣
↪2], ['d#5', BEAT],
              ['c6', BEAT], ['a5', BEAT], ['d5', BEAT], ['g5', BEAT], ['f5', BEAT], ['f5
↪', BEAT], ['d5', BEAT / 2],
              ['d#5', BEAT / 2], ['f5', BEAT], ['g5', BEAT], ['a5', BEAT], ['a#5', BEAT],
↪ ['a5', BEAT], ['g5', BEAT],
              ['g5', BEAT], ['', BEAT / 2], ['a#5', BEAT / 2], ['c6', BEAT / 2], ['d6',␣
↪BEAT / 2], ['c6', BEAT / 2],
              ['a#5', BEAT / 2], ['a5', BEAT / 2], ['g5', BEAT / 2], ['a5', BEAT / 2], [
↪'a#5', BEAT / 2], ['c6', BEAT],
              ['f5', BEAT], ['f5', BEAT], ['f5', BEAT / 2], ['d#5', BEAT / 2], ['d5',␣
↪BEAT], ['f5', BEAT], ['d6', BEAT],
              ['d6', BEAT / 2], ['c6', BEAT / 2], ['b5', BEAT], ['g5', BEAT], ['g5',␣
↪BEAT], ['c6', BEAT / 2],
              ['a#5', BEAT / 2], ['a5', BEAT], ['f5', BEAT], ['d6', BEAT], ['a5', BEAT],␣
↪['a#5', BEAT * 1.5]]

try:
    for note in liten_mus:
        speaker.play(note)
        sleep(0.1) # leave a gap between notes

finally: # Turn speaker off if interrupted
    speaker.off()
```

5.2.10 Servo

A servo motor connected to a single pin, 3.3v and ground.

Move the servo to its minimum, mid and maximum positions.

```
from picozero import Servo
from time import sleep

servo = Servo(1)

servo.min()
sleep(1)

servo.mid()
sleep(1)

servo.max()
sleep(1)

servo.off()
```

Pulse the servo between its minumum and maximum position.

```
from picozero import Servo

servo = Servo(1)

servo.pulse()
```

Move the servo gradually from its minimum to maximum position in 100 increments.

```
from picozero import Servo
from time import sleep

servo = Servo(1)

for i in range(0, 100):
    servo.value = i / 100
    sleep(0.1)

servo.off()
```

5.2.11 Motor

Move a motor connected via two pins (forward and backward) and a motor controller board:

```
from picozero import Motor
from time import sleep

motor = Motor(14, 15)

motor.move()
sleep(1)
motor.stop()
```

5.2.12 Robot rover

Make a simple two-wheeled robot rover.

Move the rover forward for 1 second and stop:

```
from picozero import Robot
from time import sleep

robot_rover = Robot(left=(14,15), right=(12,13))

# move forward
robot_rover.forward()
sleep(1)
robot_rover.stop()
```

Move the rover (roughly) in a square:

```
from picozero import Robot

robot_rover = Robot(left=(14,15), right=(12,13))

for i in range(4):
    # move forward for 1 second
    robot_rover.forward(t=1, wait=True)
    # rotate to the left for 1 second
    robot_rover.left(t=1, wait=True)
```

5.2.13 Internal temperature sensor

Check the internal temperature of the Raspberry Pi Pico in degrees Celcius:

```
# Choose View -> Plotter in Thonny to see a graph of the results

from picozero import pico_temp_sensor
from time import sleep

```

(continues on next page)

(continued from previous page)

```
while True:
    print(pico_temp_sensor.temp)
    sleep(0.1)
```

5.2.14 Ultrasonic distance sensor

Get the distance in metres from an ultrasonic distance sensor (HC-SR04):

```
from picozero import DistanceSensor
from time import sleep

ds = DistanceSensor(echo=2, trigger=3)

while True:
    print(ds.distance)
    sleep(0.1)
```

5.3 picozero API

5.3.1 LED

picozero.**LED**(*pin*, *pwm=True*, *active_high=True*, *initial_value=False*)

Returns an instance of *DigitalLED* or *PWMLED* depending on the value of the *pwm* parameter.

```
from picozero import LED

my_pwm_led = LED(1)

my_digital_led = LED(2, pwm=False)
```

Parameters

- **pin** (*int*) – The pin that the device is connected to.
- **pin** – If *pwm* is `True` (the default), a *PWMLED* will be returned. If *pwm* is `False`, a *DigitalLED* will be returned. A *PWMLED* can control the brightness of the LED but uses 1 PWM channel.
- **active_high** (*bool*) – If `True` (the default), the `on()` method will set the Pin to HIGH. If `False`, the `on()` method will set the Pin to LOW (the `off()` method always does the opposite).
- **initial_value** (*bool*) – If `False` (the default), the device will be off initially. If `True`, the device will be switched on initially.

5.3.2 DigitalLED

class picozero.**DigitalLED**(*pin*, *active_high=True*, *initial_value=False*)

Bases: *DigitalOutputDevice*

Represents a simple LED, which can be switched on and off.

Parameters

- **pin** (*int*) – The pin that the device is connected to.
- **active_high** (*bool*) – If True (the default), the *on()* method will set the Pin to HIGH. If False, the *on()* method will set the Pin to LOW (the *off()* method always does the opposite).
- **initial_value** (*bool*) – If False (the default), the LED will be off initially. If True, the LED will be switched on initially.

blink(*on_time=1*, *off_time=None*, *n=None*, *wait=False*)

Makes the device turn on and off repeatedly.

Parameters

- **on_time** (*float*) – The length of time in seconds that the device will be on. Defaults to 1.
- **off_time** (*float*) – The length of time in seconds that the device will be off. If *None*, it will be the same as on_time. Defaults to *None*.
- **n** (*int*) – The number of times to repeat the blink operation. If None is specified, the device will continue blinking forever. The default is None.
- **wait** (*bool*) – If True, the method will block until the device stops turning on and off. If False, the method will return and the device will turn on and off in the background. Defaults to False.

close()

Closes the device and turns the device off. Once closed, the device can no longer be used.

off()

Turns the device off.

on(*value=1*, *t=None*, *wait=False*)

Turns the device on.

Parameters

- **value** (*float*) – The value to set when turning on. Defaults to 1.
- **t** (*float*) – The time in seconds that the device should be on. If None is specified, the device will stay on. The default is None.
- **wait** (*bool*) – If True, the method will block until the time *t* has expired. If False, the method will return and the device will turn on in the background. Defaults to False. Only effective if *t* is not None.

toggle()

If the device is off, turn it on. If it is on, turn it off.

property active_high

Sets or returns the active_high property. If True, the *on()* method will set the Pin to HIGH. If False, the *on()* method will set the Pin to LOW (the *off()* method always does the opposite).

property is_active

Returns `True` if the device is on.

property is_lit

Returns `True` if the device is on.

property pin

Returns the pin number used by the device.

property value

Sets or returns a value representing the state of the device: 1 is on, 0 is off.

5.3.3 PWMLED

class picozero.**PWMLED**(*pin*, *freq=100*, *duty_factor=65535*, *active_high=True*, *initial_value=False*)

Bases: *PWMOutputDevice*

Represents an LED driven by a PWM pin; the brightness of the LED can be changed.

Parameters

- **pin** (*int*) – The pin that the device is connected to.
- **freq** (*int*) – The frequency of the PWM signal in hertz. Defaults to 100.
- **duty_factor** (*int*) – The duty factor of the PWM signal. This is a value between 0 and 65535. Defaults to 65535.
- **active_high** (*bool*) – If `True` (the default), the *on()* method will set the Pin to HIGH. If `False`, the *on()* method will set the Pin to LOW (the *off()* method always does the opposite).
- **initial_value** (*bool*) – If `False` (the default), the LED will be off initially. If `True`, the LED will be switched on initially.

blink(*on_time=1*, *off_time=None*, *n=None*, *wait=False*, *fade_in_time=0*, *fade_out_time=None*, *fps=25*)

Makes the device turn on and off repeatedly.

Parameters

- **on_time** (*float*) – The length of time in seconds the device will be on. Defaults to 1.
- **off_time** (*float*) – The length of time in seconds the device will be off. If *None*, it will be the same as `on_time`. Defaults to *None*.
- **n** (*int*) – The number of times to repeat the blink operation. If *None*, the device will continue blinking forever. The default is *None*.
- **wait** (*bool*) – If True, the method will block until the LED stops blinking. If False, the method will return and the LED will blink in the background. Defaults to False.
- **fade_in_time** (*float*) – The length of time in seconds to spend fading in. Defaults to 0.
- **fade_out_time** (*float*) – The length of time in seconds to spend fading out. If *None*, it will be the same as `fade_in_time`. Defaults to *None*.
- **fps** (*int*) – The frames per second that will be used to calculate the number of steps between off/on states when fading. Defaults to 25.

close()

Closes the device and turns the device off. Once closed, the device can no longer be used.

off()

Turns the device off.

on(*value=1*, *t=None*, *wait=False*)

Turns the device on.

Parameters

- **value** (*float*) – The value to set when turning on. Defaults to 1.
- **t** (*float*) – The time in seconds that the device should be on. If None is specified, the device will stay on. The default is None.
- **wait** (*bool*) – If True, the method will block until the time *t* has expired. If False, the method will return and the device will turn on in the background. Defaults to False. Only effective if *t* is not None.

pulse(*fade_in_time=1*, *fade_out_time=None*, *n=None*, *wait=False*, *fps=25*)

Makes the device pulse on and off repeatedly.

Parameters

- **fade_in_time** (*float*) – The length of time in seconds that the device will take to turn on. Defaults to 1.
- **fade_out_time** (*float*) – The length of time in seconds that the device will take to turn off. Defaults to 1.
- **fps** (*int*) – The frames per second that will be used to calculate the number of steps between off/on states. Defaults to 25.
- **n** (*int*) – The number of times to pulse the LED. If None, the LED will pulse forever. Defaults to None.
- **wait** (*bool*) – If True, the method will block until the LED stops pulsing. If False, the method will return and the LED will pulse in the background. Defaults to False.

toggle()

If the device is off, turn it on. If it is on, turn it off.

property active_high

Sets or returns the active_high property. If `True`, the *on()* method will set the Pin to HIGH. If `False`, the *on()* method will set the Pin to LOW (the *off()* method always does the opposite).

property brightness

Sets or returns a value representing the state of the device: 1 is on, 0 is off.

property freq

Returns the current frequency of the device.

property is_active

Returns `True` if the device is on.

property pin

Returns the pin number used by the device.

property value

Sets or returns a value representing the state of the device: 1 is on, 0 is off.

5.3.4 RGBLED

class picozero.**RGBLED**(*red=None*, *green=None*, *blue=None*, *active_high=True*, *initial_value=(0, 0, 0)*, *pwm=True*)

Bases: `OutputDevice`, `PinsMixin`

Extends `OutputDevice` and represents a full colour LED component (composed of red, green, and blue LEDs). Connect the common cathode (longest leg) to a ground pin; connect each of the other legs (representing the red, green, and blue anodes) to any GP pins. You should use three limiting resistors (one per anode). The following code will make the LED yellow:

```
from picozero import RGBLED
rgb = RGBLED(1, 2, 3)
rgb.color = (1, 1, 0)
```

0–255 colours are also supported:

```
rgb.color = (255, 255, 0)
```

Parameters

- **red** (*int*) – The GP pin that controls the red component of the RGB LED.
- **green** (*int*) – The GP pin that controls the green component of the RGB LED.
- **blue** (*int*) – The GP pin that controls the blue component of the RGB LED.
- **active_high** (*bool*) – Set to `True` (the default) for common cathode RGB LEDs. If you are using a common anode RGB LED, set this to `False`.
- **initial_value** (*Color or tuple*) – The initial color for the RGB LED. Defaults to black `(0, 0, 0)`.
- **pwm** (*bool*) – If `True` (the default), construct *PWMLED* instances for each component of the RGBLED. If `False`, construct *DigitalLED* instances.

blink(*on_times=1*, *fade_times=0*, *colors=((1, 0, 0), (0, 1, 0), (0, 0, 1))*, *n=None*, *wait=False*, *fps=25*)

Makes the device blink between colours repeatedly.

Parameters

- **on_times** (*float*) – Single value or tuple of numbers of seconds to stay on each colour. Defaults to 1 second.
- **fade_times** (*float*) – Single value or tuple of times to fade between each colour. Must be 0 if *pwm* was `False` when the class was constructed.
- **colors** (tuple Tuple of colours to blink between, use `(0, 0, 0)` for off.) – The colours to blink between. Defaults to red, green, blue.
- **n** (*int or None*) – Number of times to blink; `None` (the default) means forever.
- **wait** (*bool*) – If `False` (the default), use a Timer to manage blinking, continue blinking, and return immediately. If `False`, only return when the blinking is finished (warning: the default value of *n* will result in this method never returning).

close()

Turns the device off.

cycle(*fade_times=1*, *colors=((1, 0, 0), (0, 1, 0), (0, 0, 1))*, *n=None*, *wait=False*, *fps=25*)

Makes the device fade in and out repeatedly.

Parameters

- **fade_times** (*float*) – Single value or tuple of numbers of seconds to spend fading between colours. Defaults to 1.
- **fade_times** – Number of seconds to spend fading out. Defaults to 1.
- **on_color** – Tuple of colours to cycle between. Defaults to red, green, blue.
- **n** (*int or None*) – Number of times to cycle; `None` (the default) means forever.

invert()

Inverts the state of the device. If the device is currently off (*value* is `(0, 0, 0)`), this changes it to "fully" on (*value* is `(1, 1, 1)`). If the device has a specific colour, this method inverts the colour.

off()

Turns the device off.

on()

Turns the LED on. This is equivalent to setting the LED color to white, e.g. `(1, 1, 1)`.

pulse(*fade_times=1*, *colors=((0, 0, 0), (1, 0, 0), (0, 0, 0), (0, 1, 0), (0, 0, 0), (0, 0, 1))*, *n=None*, *wait=False*, *fps=25*)

Makes the device fade between colours repeatedly.

Parameters

- **fade_times** (*float*) – Single value or tuple of numbers of seconds to spend fading. Defaults to 1.
- **fade_out_time** (*float*) – Number of seconds to spend fading out. Defaults to 1.
- **on_color** – Tuple of colours to pulse between in order. Defaults to red, off, green, off, blue, off.
- **n** (*int or None*) – Number of times to pulse; `None` (the default) means forever.

toggle()

Toggles the state of the device. If the device has a specific colour, then that colour is saved and the device is turned off. If the device is off, it will be changed to the last colour it had when it was on or, if none, to fully on (*value* is `(1, 1, 1)`).

property active_high

Sets or returns the active_high property. If `True`, the *on()* method will set the Pin to HIGH. If `False`, the *on()* method will set the Pin to LOW (the *off()* method always does the opposite).

property blue

Represents the blue component of the LED as a value between 0 and 255 if *pwm* was `True` when the class was constructed (but only takes values of 0 or 255 otherwise).

property color

Represents the colour of the LED as an RGB 3-tuple of `(red, green, blue)` where each value is between 0 and 255 if *pwm* was `True` when the class was constructed (but only takes values of 0 or 255 otherwise). For example, red would be `(255, 0, 0)` and yellow would be `(255, 255, 0)`, whereas orange would be `(255, 127, 0)`.

property colour

Represents the colour of the LED as an RGB 3-tuple of `(red, green, blue)` where each value is between 0 and 255 if *pwm* was `True` when the class was constructed (but only takes values of 0 or 255 otherwise). For example, red would be `(255, 0, 0)` and yellow would be `(255, 255, 0)`, whereas orange would be `(255, 127, 0)`.

property green

Represents the green component of the LED as a value between 0 and 255 if *pwm* was `True` when the class was constructed (but only takes values of 0 or 255 otherwise).

property is_active

Returns `True` if the LED is currently active (not black) and `False` otherwise.

property is_lit

Returns `True` if the LED is currently active (not black) and `False` otherwise.

property pins

Returns a tuple of pins used by the device.

property red

Represents the red component of the LED as a value between 0 and 255 if *pwm* was `True` when the class was constructed (but only takes values of 0 or 255 otherwise).

property value

Represents the colour of the LED as an RGB 3-tuple of `(red, green, blue)` where each value is between 0 and 1 if *pwm* was `True` when the class was constructed (but only takes values of 0 or 1 otherwise). For example, red would be `(1, 0, 0)` and yellow would be `(1, 1, 0)`, whereas orange would be `(1, 0.5, 0)`.

5.3.5 Buzzer

class picozero.**Buzzer**(*pin*, *active_high=True*, *initial_value=False*)

Bases: *DigitalOutputDevice*

Represents an active or passive buzzer, which can be turned on or off.

Parameters

- **pin** (*int*) – The pin that the device is connected to.
- **active_high** (*bool*) – If `True` (the default), the *on()* method will set the Pin to HIGH. If `False`, the *on()* method will set the Pin to LOW (the *off()* method always does the opposite).
- **initial_value** (*bool*) – If `False` (the default), the Buzzer will be off initially. If `True`, the Buzzer will be switched on initially.

beep(*on_time=1*, *off_time=None*, *n=None*, *wait=False*)

Makes the device turn on and off repeatedly.

Parameters

- **on_time** (*float*) – The length of time in seconds that the device will be on. Defaults to 1.
- **off_time** (*float*) – The length of time in seconds that the device will be off. If *None*, it will be the same as `on_time`. Defaults to *None*.

- **n** (*int*) – The number of times to repeat the blink operation. If None is specified, the device will continue blinking forever. The default is None.
- **wait** (*bool*) – If True, the method will block until the device stops turning on and off. If False, the method will return and the device will turn on and off in the background. Defaults to False.

blink(*on_time=1*, *off_time=None*, *n=None*, *wait=False*)

Makes the device turn on and off repeatedly.

Parameters

- **on_time** (*float*) – The length of time in seconds that the device will be on. Defaults to 1.
- **off_time** (*float*) – The length of time in seconds that the device will be off. If *None*, it will be the same as `on_time`. Defaults to *None*.
- **n** (*int*) – The number of times to repeat the blink operation. If None is specified, the device will continue blinking forever. The default is None.
- **wait** (*bool*) – If True, the method will block until the device stops turning on and off. If False, the method will return and the device will turn on and off in the background. Defaults to False.

close()

Closes the device and turns the device off. Once closed, the device can no longer be used.

off()

Turns the device off.

on(*value=1*, *t=None*, *wait=False*)

Turns the device on.

Parameters

- **value** (*float*) – The value to set when turning on. Defaults to 1.
- **t** (*float*) – The time in seconds that the device should be on. If None is specified, the device will stay on. The default is None.
- **wait** (*bool*) – If True, the method will block until the time *t* has expired. If False, the method will return and the device will turn on in the background. Defaults to False. Only effective if *t* is not None.

toggle()

If the device is off, turn it on. If it is on, turn it off.

property active_high

Sets or returns the active_high property. If `True`, the *on()* method will set the Pin to HIGH. If `False`, the *on()* method will set the Pin to LOW (the *off()* method always does the opposite).

property is_active

Returns `True` if the device is on.

property pin

Returns the pin number used by the device.

property value

Sets or returns a value representing the state of the device: 1 is on, 0 is off.

5.3.6 PWMBuzzer

class picozero.**PWMBuzzer**(*pin*, *freq=440*, *duty_factor=1023*, *active_high=True*, *initial_value=False*)

Bases: *PWMOutputDevice*

Represents a passive buzzer driven by a PWM pin; the volume of the buzzer can be changed.

Parameters

- **pin** (*int*) – The pin that the buzzer is connected to.
- **freq** (*int*) – The frequency of the PWM signal in hertz. Defaults to 440.
- **duty_factor** (*int*) – The duty factor of the PWM signal. This is a value between 0 and 65535. Defaults to 1023.
- **active_high** (*bool*) – If `True` (the default), the *on()* method will set the Pin to HIGH. If `False`, the *on()* method will set the Pin to LOW (the *off()* method always does the opposite).
- **initial_value** (*bool*) – If `False` (the default), the buzzer will be off initially. If `True`, the buzzer will be switched on initially.

beep(*on_time=1*, *off_time=None*, *n=None*, *wait=False*, *fade_in_time=0*, *fade_out_time=None*, *fps=25*)

Makes the device turn on and off repeatedly.

Parameters

- **on_time** (*float*) – The length of time in seconds the device will be on. Defaults to 1.
- **off_time** (*float*) – The length of time in seconds the device will be off. If *None*, it will be the same as `on_time`. Defaults to *None*.
- **n** (*int*) – The number of times to repeat the blink operation. If *None*, the device will continue blinking forever. The default is *None*.
- **wait** (*bool*) – If True, the method will block until the LED stops blinking. If False, the method will return and the LED will blink in the background. Defaults to False.
- **fade_in_time** (*float*) – The length of time in seconds to spend fading in. Defaults to 0.
- **fade_out_time** (*float*) – The length of time in seconds to spend fading out. If *None*, it will be the same as `fade_in_time`. Defaults to *None*.
- **fps** (*int*) – The frames per second that will be used to calculate the number of steps between off/on states when fading. Defaults to 25.

blink(*on_time=1*, *off_time=None*, *n=None*, *wait=False*, *fade_in_time=0*, *fade_out_time=None*, *fps=25*)

Makes the device turn on and off repeatedly.

Parameters

- **on_time** (*float*) – The length of time in seconds the device will be on. Defaults to 1.
- **off_time** (*float*) – The length of time in seconds the device will be off. If *None*, it will be the same as `on_time`. Defaults to *None*.
- **n** (*int*) – The number of times to repeat the blink operation. If *None*, the device will continue blinking forever. The default is *None*.
- **wait** (*bool*) – If True, the method will block until the LED stops blinking. If False, the method will return and the LED will blink in the background. Defaults to False.
- **fade_in_time** (*float*) – The length of time in seconds to spend fading in. Defaults to 0.

- **fade_out_time** (*float*) – The length of time in seconds to spend fading out. If *None*, it will be the same as fade_in_time. Defaults to *None*.
- **fps** (*int*) – The frames per second that will be used to calculate the number of steps between off/on states when fading. Defaults to 25.

close()

Closes the device and turns the device off. Once closed, the device can no longer be used.

off()

Turns the device off.

on(*value=1*, *t=None*, *wait=False*)

Turns the device on.

Parameters

- **value** (*float*) – The value to set when turning on. Defaults to 1.
- **t** (*float*) – The time in seconds that the device should be on. If None is specified, the device will stay on. The default is None.
- **wait** (*bool*) – If True, the method will block until the time *t* has expired. If False, the method will return and the device will turn on in the background. Defaults to False. Only effective if *t* is not None.

pulse(*fade_in_time=1*, *fade_out_time=None*, *n=None*, *wait=False*, *fps=25*)

Makes the device pulse on and off repeatedly.

Parameters

- **fade_in_time** (*float*) – The length of time in seconds that the device will take to turn on. Defaults to 1.
- **fade_out_time** (*float*) – The length of time in seconds that the device will take to turn off. Defaults to 1.
- **fps** (*int*) – The frames per second that will be used to calculate the number of steps between off/on states. Defaults to 25.
- **n** (*int*) – The number of times to pulse the LED. If None, the LED will pulse forever. Defaults to None.
- **wait** (*bool*) – If True, the method will block until the LED stops pulsing. If False, the method will return and the LED will pulse in the background. Defaults to False.

toggle()

If the device is off, turn it on. If it is on, turn it off.

property active_high

Sets or returns the active_high property. If True, the *on()* method will set the Pin to HIGH. If False, the *on()* method will set the Pin to LOW (the *off()* method always does the opposite).

property freq

Returns the current frequency of the device.

property is_active

Returns True if the device is on.

property pin

Returns the pin number used by the device.

property value

Sets or returns a value representing the state of the device: 1 is on, 0 is off.

property volume

Sets or returns a value representing the state of the device: 1 is on, 0 is off.

5.3.7 Speaker

class picozero.**Speaker**(*pin*, *initial_freq=440*, *initial_volume=0*, *duty_factor=1023*, *active_high=True*)

Bases: OutputDevice, PinMixin

Represents a speaker driven by a PWM pin.

Parameters

- **pin** (*int*) – The pin that the speaker is connected to.
- **initial_freq** (*int*) – The initial frequency of the PWM signal in hertz. Defaults to 440.
- **initial_volume** (*int*) – The initial volume of the PWM signal. This is a value between 0 and 1. Defaults to 0.
- **duty_factor** (*int*) – The duty factor of the PWM signal. This is a value between 0 and 65535. Defaults to 1023.
- **active_high** (*bool*) – If True (the default), the *on()* method will set the Pin to HIGH. If False, the *on()* method will set the Pin to LOW (the *off()* method always does the opposite).

beep(*on_time=1*, *off_time=None*, *n=None*, *wait=False*, *fade_in_time=0*, *fade_out_time=None*, *fps=25*)

Makes the buzzer turn on and off repeatedly.

Parameters

- **on_time** (*float*) – The length of time in seconds that the device will be on. Defaults to 1.
- **off_time** (*float*) – The length of time in seconds that the device will be off. If *None*, it will be the same as on_time. Defaults to *None*.
- **n** (*int*) – The number of times to repeat the beep operation. If *None*, the device will continue beeping forever. The default is *None*.
- **wait** (*bool*) – If True, the method will block until the buzzer stops beeping. If False, the method will return and the buzzer will beep in the background. Defaults to False.
- **fade_in_time** (*float*) – The length of time in seconds to spend fading in. Defaults to 0.
- **fade_out_time** (*float*) – The length of time in seconds to spend fading out. If *None*, it will be the same as fade_in_time. Defaults to *None*.
- **fps** (*int*) – The frames per second that will be used to calculate the number of steps between off/on states when fading. Defaults to 25.

blink(*on_time=1*, *off_time=None*, *n=None*, *wait=False*)

Makes the device turn on and off repeatedly.

Parameters

- **on_time** (*float*) – The length of time in seconds that the device will be on. Defaults to 1.

- **off_time** (*float*) – The length of time in seconds that the device will be off. If *None*, it will be the same as on_time. Defaults to *None*.
- **n** (*int*) – The number of times to repeat the blink operation. If None is specified, the device will continue blinking forever. The default is None.
- **wait** (*bool*) – If True, the method will block until the device stops turning on and off. If False, the method will return and the device will turn on and off in the background. Defaults to False.

close()

Turns the device off.

off()

Turns the device off.

on(*volume=1*)

Turns the device on.

Parameters

- **value** (*float*) – The value to set when turning on. Defaults to 1.
- **t** (*float*) – The time in seconds that the device should be on. If None is specified, the device will stay on. The default is None.
- **wait** (*bool*) – If True, the method will block until the time *t* has expired. If False, the method will return and the device will turn on in the background. Defaults to False. Only effective if *t* is not None.

play(*tune=440*, *duration=1*, *volume=1*, *n=1*, *wait=True*)

Plays a tune for a given duration.

Parameters

- **tune** (*int*) – The tune to play can be specified as:
  - a single "note", represented as: + a frequency in Hz e.g. *440* + a midi note e.g. *60* + a note name as a string e.g. *"E4"*
  - a list of notes and duration e.g. *[440, 1]* or *["E4", 2]*
  - a list of two value tuples of (note, duration) e.g. *[(440,1), (60, 2), ("e4", 3)]*

  Defaults to *440*.
- **volume** (*int*) – The volume of the tune; 1 is maximum volume, 0 is mute. Defaults to 1.
- **duration** (*float*) – The duration of each note in seconds. Defaults to 1.
- **n** (*int*) – The number of times to play the tune. If None, the tune will play forever. Defaults to 1.
- **wait** (*bool*) – If True, the method will block until the tune has finished. If False, the method will return and the tune will play in the background. Defaults to True.

toggle()

If the device is off, turn it on. If it is on, turn it off.

property active_high

Sets or returns the active_high property. If True, the *on()* method will set the Pin to HIGH. If False, the *on()* method will set the Pin to LOW (the *off()* method always does the opposite).

property freq

Sets or returns the current frequency of the speaker.

property is_active

Returns `True` if the device is on.

property pin

Returns the pin number used by the device.

property value

Sets or returns the value of the speaker. The value is a tuple of (freq, volume).

property volume

Sets or returns the volume of the speaker: 1 for maximum volume, 0 for off.

5.3.8 Servo

class picozero.**Servo**(*pin*, *initial_value=None*, *min_pulse_width=0.001*, *max_pulse_width=0.002*, *frame_width=0.02*, *duty_factor=65535*)

Bases: *PWMOutputDevice*

Represents a PWM-controlled servo motor.

Setting the *value* to 0 will move the servo to its minimum position, 1 will move the servo to its maximum position. Setting the *value* to `None` will turn the servo "off" (i.e. no signal is sent).

Parameters

- **pin** (*int*) – The pin the servo motor is connected to.
- **initial_value** (*bool*) – If `0`, the servo will be set to its minimum position. If `1`, the servo will set to its maximum position. If `None` (the default), the position of the servo will not change.
- **min_pulse_width** (*float*) – The pulse width corresponding to the servo's minimum position. This defaults to 1ms.
- **max_pulse_width** (*float*) – The pulse width corresponding to the servo's maximum position. This defaults to 2ms.
- **frame_width** (*float*) – The length of time between servo control pulses measured in seconds. This defaults to 20ms which is a common value for servos.
- **duty_factor** (*int*) – The duty factor of the PWM signal. This is a value between 0 and 65535. Defaults to 65535.

blink(*on_time=1*, *off_time=None*, *n=None*, *wait=False*, *fade_in_time=0*, *fade_out_time=None*, *fps=25*)

Makes the device turn on and off repeatedly.

Parameters

- **on_time** (*float*) – The length of time in seconds the device will be on. Defaults to 1.
- **off_time** (*float*) – The length of time in seconds the device will be off. If *None*, it will be the same as `on_time`. Defaults to *None*.
- **n** (*int*) – The number of times to repeat the blink operation. If *None*, the device will continue blinking forever. The default is *None*.
- **wait** (*bool*) – If True, the method will block until the LED stops blinking. If False, the method will return and the LED will blink in the background. Defaults to False.

- **fade_in_time** (*float*) – The length of time in seconds to spend fading in. Defaults to 0.
- **fade_out_time** (*float*) – The length of time in seconds to spend fading out. If *None*, it will be the same as fade_in_time. Defaults to *None*.
- **fps** (*int*) – The frames per second that will be used to calculate the number of steps between off/on states when fading. Defaults to 25.

close()

Closes the device and turns the device off. Once closed, the device can no longer be used.

max()

Set the servo to its maximum position.

mid()

Set the servo to its mid-point position.

min()

Set the servo to its minimum position.

off()

Turn the servo "off" by setting the value to *None*.

on(*value=1*, *t=None*, *wait=False*)

Turns the device on.

Parameters

- **value** (*float*) – The value to set when turning on. Defaults to 1.
- **t** (*float*) – The time in seconds that the device should be on. If None is specified, the device will stay on. The default is None.
- **wait** (*bool*) – If True, the method will block until the time *t* has expired. If False, the method will return and the device will turn on in the background. Defaults to False. Only effective if *t* is not None.

pulse(*fade_in_time=1*, *fade_out_time=None*, *n=None*, *wait=False*, *fps=25*)

Makes the device pulse on and off repeatedly.

Parameters

- **fade_in_time** (*float*) – The length of time in seconds that the device will take to turn on. Defaults to 1.
- **fade_out_time** (*float*) – The length of time in seconds that the device will take to turn off. Defaults to 1.
- **fps** (*int*) – The frames per second that will be used to calculate the number of steps between off/on states. Defaults to 25.
- **n** (*int*) – The number of times to pulse the LED. If None, the LED will pulse forever. Defaults to None.
- **wait** (*bool*) – If True, the method will block until the LED stops pulsing. If False, the method will return and the LED will pulse in the background. Defaults to False.

toggle()

If the device is off, turn it on. If it is on, turn it off.

property active_high

Sets or returns the active_high property. If `True`, the *on()* method will set the Pin to HIGH. If `False`, the *on()* method will set the Pin to LOW (the *off()* method always does the opposite).

property freq

Returns the current frequency of the device.

property is_active

Returns `True` if the device is on.

property pin

Returns the pin number used by the device.

property value

Sets or returns a value representing the state of the device: 1 is on, 0 is off.

5.3.9 Motor

class picozero.**Motor**(*forward*, *backward*, *pwm=True*)

Bases: `PinsMixin`

Represents a motor connected to a motor controller that has a two-pin input. One pin drives the motor "forward", the other drives the motor "backward".

Parameters

- **forward** (*int*) – The GP pin that controls the "forward" motion of the motor.
- **backward** (*int*) – The GP pin that controls the "backward" motion of the motor.
- **pwm** (*bool*) – If `True` (the default), PWM pins are used to drive the motor. When using PWM pins, values between 0 and 1 can be used to set the speed.

backward(*speed=1*, *t=None*, *wait=False*)

Makes the motor turn "backward".

Parameters

- **speed** (*float*) – The speed as a value between 0 and 1: 1 is full speed, 0 is stop. Defaults to 1.
- **t** (*float*) – The time in seconds that the motor should turn for. If None is specified, the motor will stay on. The default is None.
- **wait** (*bool*) – If True, the method will block until the time *t* has expired. If False, the method will return and the motor will turn on in the background. Defaults to False. Only effective if *t* is not None.

close()

Closes the device and releases any resources. Once closed, the device can no longer be used.

forward(*speed=1*, *t=None*, *wait=False*)

Makes the motor turn "forward".

Parameters

- **speed** (*float*) – The speed as a value between 0 and 1: 1 is full speed, 0 is stop. Defaults to 1.
- **t** (*float*) – The time in seconds that the motor should turn for. If None is specified, the motor will stay on. The default is None.

- **wait** (*bool*) – If True, the method will block until the time *t* has expired. If False, the method will return and the motor will turn on in the background. Defaults to False. Only effective if *t* is not None.

off()

Stops the motor turning.

on(*speed=1*, *t=None*, *wait=False*)

Turns the motor on and makes it turn.

Parameters

- **speed** (*float*) – The speed as a value between -1 and 1: 1 turns the motor at full speed in one direction, -1 turns the motor at full speed in the opposite direction. Defaults to 1.
- **t** (*float*) – The time in seconds that the motor should run for. If None is specified, the motor will stay on. The default is None.
- **wait** (*bool*) – If True, the method will block until the time *t* has expired. If False, the method will return and the motor will turn on in the background. Defaults to False. Only effective if *t* is not None.

start(*speed=1*, *t=None*, *wait=False*)

Turns the motor on and makes it turn.

Parameters

- **speed** (*float*) – The speed as a value between -1 and 1: 1 turns the motor at full speed in one direction, -1 turns the motor at full speed in the opposite direction. Defaults to 1.
- **t** (*float*) – The time in seconds that the motor should run for. If None is specified, the motor will stay on. The default is None.
- **wait** (*bool*) – If True, the method will block until the time *t* has expired. If False, the method will return and the motor will turn on in the background. Defaults to False. Only effective if *t* is not None.

stop()

Stops the motor turning.

property pins

Returns a tuple of pins used by the device.

property value

Sets or returns the motor speed as a value between -1 and 1: -1 is full speed "backward", 1 is full speed "forward", 0 is stopped.

5.3.10 Robot / Rover

class picozero.**Robot**(*left*, *right*, *pwm=True*)

Bases: object

Represents a generic dual-motor robot / rover / buggy.

Alias for Rover.

This class is constructed with two tuples representing the forward and backward pins of the left and right controllers. For example, if the left motor's controller is connected to pins 12 and 13, while the right motor's controller is connected to pins 14 and 15, then the following example will drive the robot forward:

```
from picozero import Robot

robot = Robot(left=(12, 13), right=(14, 15))
robot.forward()
```

Parameters

- **left** (*tuple*) – A tuple of two pins representing the forward and backward inputs of the left motor's controller.
- **right** (*tuple*) – A tuple of two pins representing the forward and backward inputs of the right motor's controller.
- **pwm** (*bool*) – If True (the default), pwm pins will be used, allowing variable speed control.

backward(*speed=1*, *t=None*, *wait=False*)

Makes the robot move "backward".

Parameters

- **speed** (*float*) – The speed as a value between 0 and 1: 1 is full speed, 0 is stop. Defaults to 1.
- **t** (*float*) – The time in seconds that the robot should move for. If None is specified, the robot will continue to move until stopped. The default is None.
- **wait** (*bool*) – If True, the method will block until the time *t* has expired. If False, the method will return and the motor will turn on in the background. Defaults to False. Only effective if *t* is not None.

close()

Closes the device and releases any resources. Once closed, the device can no longer be used.

forward(*speed=1*, *t=None*, *wait=False*)

Makes the robot move "forward".

Parameters

- **speed** (*float*) – The speed as a value between 0 and 1: 1 is full speed, 0 is stop. Defaults to 1.
- **t** (*float*) – The time in seconds that the robot should move for. If None is specified, the robot will continue to move until stopped. The default is None.
- **wait** (*bool*) – If True, the method will block until the time *t* has expired. If False, the method will return and the motor will turn on in the background. Defaults to False. Only effective if *t* is not None.

left(*speed=1*, *t=None*, *wait=False*)

Makes the robot turn "left" by turning the left motor backward and the right motor forward.

Parameters

- **speed** (*float*) – The speed as a value between 0 and 1: 1 is full speed, 0 is stop. Defaults to 1.
- **t** (*float*) – The time in seconds that the robot should turn for. If None is specified, the robot will continue to turn until stopped. The default is None.

- **wait** (*bool*) – If True, the method will block until the time *t* has expired. If False, the method will return and the motor will turn on in the background. Defaults to False. Only effective if *t* is not None.

right(*speed=1*, *t=None*, *wait=False*)

Makes the robot turn "right" by turning the left motor forward and the right motor backward.

Parameters

- **speed** (*float*) – The speed as a value between 0 and 1: 1 is full speed, 0 is stop. Defaults to 1.
- **t** (*float*) – The time in seconds that the robot should turn for. If None is specified, the robot will continue to turn until stopped. The default is None.
- **wait** (*bool*) – If True, the method will block until the time *t* has expired. If False, the method will return and the motor will turn on in the background. Defaults to False. Only effective if *t* is not None.

stop()

Stops the robot.

property left_motor

Returns the left *Motor*.

property right_motor

Returns the right *Motor*.

property value

Represents the motion of the robot as a tuple of (left_motor_speed, right_motor_speed) with `(-1, -1)` representing full speed backwards, `(1, 1)` representing full speed forwards, and `(0, 0)` representing stopped.

5.3.11 DigitalOutputDevice

class picozero.**DigitalOutputDevice**(*pin*, *active_high=True*, *initial_value=False*)

Bases: `OutputDevice`, `PinMixin`

Represents a device driven by a digital pin.

Parameters

- **pin** (*int*) – The pin that the device is connected to.
- **active_high** (*bool*) – If `True` (the default), the *on()* method will set the Pin to HIGH. If `False`, the *on()* method will set the Pin to LOW (the *off()* method always does the opposite).
- **initial_value** (*bool*) – If `False` (the default), the LED will be off initially. If `True`, the LED will be switched on initially.

blink(*on_time=1*, *off_time=None*, *n=None*, *wait=False*)

Makes the device turn on and off repeatedly.

Parameters

- **on_time** (*float*) – The length of time in seconds that the device will be on. Defaults to 1.

- **off_time** (*float*) – The length of time in seconds that the device will be off. If *None*, it will be the same as on_time. Defaults to *None*.
- **n** (*int*) – The number of times to repeat the blink operation. If None is specified, the device will continue blinking forever. The default is None.
- **wait** (*bool*) – If True, the method will block until the device stops turning on and off. If False, the method will return and the device will turn on and off in the background. Defaults to False.

close()

Closes the device and turns the device off. Once closed, the device can no longer be used.

off()

Turns the device off.

on(*value=1*, *t=None*, *wait=False*)

Turns the device on.

Parameters

- **value** (*float*) – The value to set when turning on. Defaults to 1.
- **t** (*float*) – The time in seconds that the device should be on. If None is specified, the device will stay on. The default is None.
- **wait** (*bool*) – If True, the method will block until the time *t* has expired. If False, the method will return and the device will turn on in the background. Defaults to False. Only effective if *t* is not None.

toggle()

If the device is off, turn it on. If it is on, turn it off.

property active_high

Sets or returns the active_high property. If True, the *on()* method will set the Pin to HIGH. If False, the *on()* method will set the Pin to LOW (the *off()* method always does the opposite).

property is_active

Returns True if the device is on.

property pin

Returns the pin number used by the device.

property value

Sets or returns a value representing the state of the device: 1 is on, 0 is off.

5.3.12 PWMOutputDevice

class picozero.**PWMOutputDevice**(*pin*, *freq=100*, *duty_factor=65535*, *active_high=True*, *initial_value=False*)

Bases: OutputDevice, PinMixin

Represents a device driven by a PWM pin.

Parameters

- **pin** (*int*) – The pin that the device is connected to.
- **freq** (*int*) – The frequency of the PWM signal in hertz. Defaults to 100.
- **duty_factor** (*int*) – The duty factor of the PWM signal. This is a value between 0 and 65535. Defaults to 65535.

- **active_high** (*bool*) – If `True` (the default), the *on()* method will set the Pin to HIGH. If `False`, the *on()* method will set the Pin to LOW (the *off()* method always does the opposite).
- **initial_value** (*bool*) – If `False` (the default), the LED will be off initially. If `True`, the LED will be switched on initially.

blink(*on_time=1*, *off_time=None*, *n=None*, *wait=False*, *fade_in_time=0*, *fade_out_time=None*, *fps=25*)

Makes the device turn on and off repeatedly.

Parameters

- **on_time** (*float*) – The length of time in seconds the device will be on. Defaults to 1.
- **off_time** (*float*) – The length of time in seconds the device will be off. If *None*, it will be the same as `on_time`. Defaults to *None*.
- **n** (*int*) – The number of times to repeat the blink operation. If *None*, the device will continue blinking forever. The default is *None*.
- **wait** (*bool*) – If True, the method will block until the LED stops blinking. If False, the method will return and the LED will blink in the background. Defaults to False.
- **fade_in_time** (*float*) – The length of time in seconds to spend fading in. Defaults to 0.
- **fade_out_time** (*float*) – The length of time in seconds to spend fading out. If *None*, it will be the same as `fade_in_time`. Defaults to *None*.
- **fps** (*int*) – The frames per second that will be used to calculate the number of steps between off/on states when fading. Defaults to 25.

close()

Closes the device and turns the device off. Once closed, the device can no longer be used.

off()

Turns the device off.

on(*value=1*, *t=None*, *wait=False*)

Turns the device on.

Parameters

- **value** (*float*) – The value to set when turning on. Defaults to 1.
- **t** (*float*) – The time in seconds that the device should be on. If None is specified, the device will stay on. The default is None.
- **wait** (*bool*) – If True, the method will block until the time *t* has expired. If False, the method will return and the device will turn on in the background. Defaults to False. Only effective if *t* is not None.

pulse(*fade_in_time=1*, *fade_out_time=None*, *n=None*, *wait=False*, *fps=25*)

Makes the device pulse on and off repeatedly.

Parameters

- **fade_in_time** (*float*) – The length of time in seconds that the device will take to turn on. Defaults to 1.
- **fade_out_time** (*float*) – The length of time in seconds that the device will take to turn off. Defaults to 1.
- **fps** (*int*) – The frames per second that will be used to calculate the number of steps between off/on states. Defaults to 25.

- **n** (*int*) – The number of times to pulse the LED. If None, the LED will pulse forever. Defaults to None.
- **wait** (*bool*) – If True, the method will block until the LED stops pulsing. If False, the method will return and the LED will pulse in the background. Defaults to False.

toggle()

If the device is off, turn it on. If it is on, turn it off.

property active_high

Sets or returns the active_high property. If `True`, the *on()* method will set the Pin to HIGH. If `False`, the *on()* method will set the Pin to LOW (the *off()* method always does the opposite).

property freq

Returns the current frequency of the device.

property is_active

Returns `True` if the device is on.

property pin

Returns the pin number used by the device.

property value

Sets or returns a value representing the state of the device: 1 is on, 0 is off.

5.3.13 Button

class picozero.**Button**(*pin*, *pull_up=True*, *bounce_time=0.02*)

Bases: *Switch*

Represents a push button, which can be either pressed or released.

Parameters

- **pin** (*int*) – The pin that the device is connected to.
- **pull_up** (*bool*) – If `True` (the default), the device will be pulled up to HIGH. If `False`, the device will be pulled down to LOW.
- **bounce_time** (*float*) – The bounce time for the device. If set, the device will ignore any button presses that happen within the bounce time after a button release. This is useful to prevent accidental button presses from registering as multiple presses. Defaults to 0.02 seconds.

close()

Closes the device and releases any resources. Once closed, the device can no longer be used.

property active_state

Sets or returns the active state of the device. If `None` (the default), the device will return the value that the pin is set to. If `True`, the device will return `True` if the pin is HIGH. If `False`, the device will return `False` if the pin is LOW.

property is_active

Returns `True` if the device is active.

property is_closed

Returns `True` if the device is active.

property is_inactive

Returns `True` if the device is inactive.

property is_open

Returns `True` if the device is inactive.

property is_pressed

Returns `True` if the device is active.

property is_released

Returns `True` if the device is inactive.

property pin

Returns the pin number used by the device.

property value

Returns the current value of the device. This is either `True` or `False` depending on the value of *active_state*.

property when_activated

Returns a `callback` that will be called when the device is activated.

property when_closed

Returns a `callback` that will be called when the device is activated.

property when_deactivated

Returns a `callback` that will be called when the device is deactivated.

property when_opened

Returns a `callback` that will be called when the device is deactivated.

property when_pressed

Returns a `callback` that will be called when the device is activated.

property when_released

Returns a `callback` that will be called when the device is deactivated.

5.3.14 Switch

class picozero.**Switch**(*pin*, *pull_up=True*, *bounce_time=0.02*)

Bases: *DigitalInputDevice*

Represents a toggle switch, which is either open or closed.

Parameters

- **pin** (*int*) – The pin that the device is connected to.
- **pull_up** (*bool*) – If `True` (the default), the device will be pulled up to HIGH. If `False`, the device will be pulled down to LOW.
- **bounce_time** (*float*) – The bounce time for the device. If set, the device will ignore any button presses that happen within the bounce time after a button release. This is useful to prevent accidental button presses from registering as multiple presses. Defaults to 0.02 seconds.

close()

Closes the device and releases any resources. Once closed, the device can no longer be used.

property active_state

Sets or returns the active state of the device. If `None` (the default), the device will return the value that the pin is set to. If `True`, the device will return `True` if the pin is HIGH. If `False`, the device will return `False` if the pin is LOW.

property is_active

Returns `True` if the device is active.

property is_closed

Returns `True` if the device is active.

property is_inactive

Returns `True` if the device is inactive.

property is_open

Returns `True` if the device is inactive.

property pin

Returns the pin number used by the device.

property value

Returns the current value of the device. This is either `True` or `False` depending on the value of *active_state*.

property when_activated

Returns a `callback` that will be called when the device is activated.

property when_closed

Returns a `callback` that will be called when the device is activated.

property when_deactivated

Returns a `callback` that will be called when the device is deactivated.

property when_opened

Returns a `callback` that will be called when the device is deactivated.

5.3.15 Potentiometer / Pot

class picozero.**Potentiometer**(*pin*, *active_state=True*, *threshold=0.5*)

Bases: `AnalogInputDevice`

Represents a potentiometer, which outputs a variable voltage between 0 and 3.3V.

Alias for `Pot`.

Parameters

- **pin** (*int*) – The pin that the device is connected to.
- **active_state** – The active state of the device. If `True` (the default), the `AnalogInputDevice` will assume that the device is active when the pin is high and above the threshold. If `active_state` is `False`, the device will be active when the pin is low and below the threshold.
- **threshold** (*float*) – The threshold that the device must be above or below to be considered active. The default is 0.5.

property active_state

Sets or returns the active state of the device. If `None` (the default), the device will return the value that the pin is set to. If `True`, the device will return `True` if the pin is HIGH. If `False`, the device will return `False` if the pin is LOW.

property is_active

Returns `True` if the device is active.

property pin

Returns the pin number used by the device.

property threshold

The threshold that the device must be above or below to be considered active. The default is 0.5.

property value

Returns the current value of the device. This is either `True` or `False` depending on the value of *active_state*.

property voltage

Returns the voltage of the analogue device.

5.3.16 TemperatureSensor / TempSensor / Thermistor

class picozero.**TemperatureSensor**(*pin*, *active_state=True*, *threshold=0.5*, *conversion=None*)

Bases: `AnalogInputDevice`

Represents a TemperatureSensor, which outputs a variable voltage. The voltage can be converted to a temperature using a *conversion* function passed as a parameter.

Alias for `Thermistor` and `TempSensor`.

Parameters

- **pin** (*int*) – The pin that the device is connected to.
- **active_state** – The active state of the device. If `True` (the default), the `AnalogInputDevice` will assume that the device is active when the pin is high and above the threshold. If `active_state` is `False`, the device will be active when the pin is low and below the threshold.
- **threshold** (*float*) – The threshold that the device must be above or below to be considered active. The default is 0.5.
- **conversion** (*float*) – A function that takes a voltage and returns a temperature.

  e.g. The internal temperature sensor has a voltage range of 0.706V to 0.716V and would use the follow conversion function:

```
def temp_conversion(voltage):
    return 27 - (voltage - 0.706)/0.001721

temp_sensor = TemperatureSensor(pin, conversion=temp_conversion)
```

  If `None` (the default), the `temp` property will return `None`.

property active_state

Sets or returns the active state of the device. If `None` (the default), the device will return the value that the pin is set to. If `True`, the device will return `True` if the pin is HIGH. If `False`, the device will return `False` if the pin is LOW.

property conversion

Sets or returns the conversion function for the device.

property is_active

Returns `True` if the device is active.

property pin

Returns the pin number used by the device.

property temp

Returns the temperature of the device. If the conversion function is not set, this will return `None`.

property threshold

The threshold that the device must be above or below to be considered active. The default is 0.5.

property value

Returns the current value of the device. This is either `True` or `False` depending on the value of *active_state*.

property voltage

Returns the voltage of the analogue device.

5.3.17 DistanceSensor

class picozero.**DistanceSensor**(*echo*, *trigger*, *max_distance=1*)

Bases: `PinsMixin`

Represents a HC-SR04 ultrasonic distance sensor.

Parameters

- **echo** (*int*) – The pin that the ECHO pin is connected to.
- **trigger** (*int*) – The pin that the TRIG pin is connected to.
- **max_distance** (*float*) – The *value* attribute reports a normalized value between 0 (too close to measure) and 1 (maximum distance). This parameter specifies the maximum distance expected in meters. This defaults to 1.

property distance

Returns the current distance measured by the sensor in meters. Note that this property will have a value between 0 and max_distance.

property max_distance

Returns the maximum distance that the sensor will measure in metres.

property pins

Returns a tuple of pins used by the device.

property value

Returns a value between 0, indicating the reflector is either touching the sensor or is sufficiently near that the sensor can't tell the difference, and 1, indicating the reflector is at or beyond the specified max_distance. A return value of None indicates that the echo was not received before the timeout.

5.3.18 DigitalInputDevice

class picozero.**DigitalInputDevice**(*pin*, *pull_up=False*, *active_state=None*, *bounce_time=None*)

Bases: InputDevice, PinMixin

Represents a generic input device with digital functionality e.g. buttons that can be either active or inactive.

Parameters

- **pin** (*int*) – The pin that the device is connected to.
- **pull_up** (*bool*) – If True, the device will be pulled up to HIGH. If False (the default), the device will be pulled down to LOW.
- **active_state** (*bool*) – If True (the default), the device will return True if the pin is HIGH. If False, the device will return False if the pin is LOW.
- **bounce_time** (*float*) – The bounce time for the device. If set, the device will ignore any button presses that happen within the bounce time after a button release. This is useful to prevent accidental button presses from registering as multiple presses. The default is None.

close()

Closes the device and releases any resources. Once closed, the device can no longer be used.

property active_state

Sets or returns the active state of the device. If None (the default), the device will return the value that the pin is set to. If True, the device will return True if the pin is HIGH. If False, the device will return False if the pin is LOW.

property is_active

Returns True if the device is active.

property is_inactive

Returns True if the device is inactive.

property pin

Returns the pin number used by the device.

property value

Returns the current value of the device. This is either True or False depending on the value of *active_state*.

property when_activated

Returns a callback that will be called when the device is activated.

property when_deactivated

Returns a callback that will be called when the device is deactivated.

5.3.19 pinout

picozero.**pinout**(*output=True*)

Returns a textual representation of the Raspberry Pi pico pins and functions.

Parameters

output (*bool*) – If True (the default) the pinout will be "printed".

5.4 Development

Instructions on how build and deploy picozero.

5.4.1 Pre-requisites

To build and deploy picozero, you need to install the dependencies

```
pip3 install twine sphinx
```

5.4.2 Build

1. Update version numbers in the `setup.py`, `picozero/__init__.py`, and `docs/conf.py` files
2. Add release to `docs/changelog.rst`
3. Run `setup.py` and create a source distribution

   ```
   python3 setup.py sdist
   ```

4. Upload to PyPI

   ```
   twine upload dist/*
   ```

5. Push all changes to `master` branch
6. Create a release in github and upload `picozero-#-#-#.tar.gz` source file to the release

5.4.3 Documentation

The documentation site is built using Sphinx.

Install sphinx using

```
pip3 install sphinx
```

To test the documentation build, run the following command from the docs directory

```
$ ./make html
```

The website will be built in the directory docs/_build/html.

Documentation can be viewed at picozero.readthedocs.io and is automatically built and deployed on push to github.

5.4.4 Tests

The tests are designed to be run on a Raspberry Pi Pico.

1. Install the picozero package
2. Install the micropython-unittest package
3. Copy the `test_picozero.py` file to the Pico
4. Run the `test_picozero.py` file

If a test fails, it is helpful to be able to see verbose error messages. To see error messages, you need to modify the `lib/unittest.py` file on the Pico.

Locate the following code in the `run_class` function:

```
# Uncomment to investigate failure in detail
#raise
```

Uncomment `raise`:

```
# Uncomment to investigate failure in detail
raise
```

5.5 Contributing

Contributions to picozero are welcome. Please keep in mind that picozero is 'lightweight'. It is designed to be easy to use but also needs to run on a microcontroller; please take this into account when considering feature requests or raising issues.

For more details, please see the following advice.

5.5.1 Status

As picozero is currently in Beta, pre-release 1.0, you should consider the following:

- The API is not yet set, however, this doesn't mean that backwards compatibility is not important! It is a balancing act.
- Requests for new features will need to be prioritised and responses to feature requests may take some time.
- Refactoring of the code base is very likely and, as a result, pull requests may need rework.
- Issues are likely to exist within the code base. Be kind!

5.5.2 Suggestions

If you have an idea for a new feature or would like to see a device included in picozero, please raise an issue. Please explain your reasoning clearly.

5.5.3 Bugs

Please raise an issue for any bugs found. Please include code examples and circuit diagrams if appropriate.

5.5.4 Pull requests

All pull requests should be based on the dev branch of picozero.

5.6 Change log

5.6.1 0.4.2 - 2023-05-12

- Bug fix relating to DigitalInputDevice bounce times
- Updated tests after a change in micropython 1.20+

5.6.2 0.4.1 - 2022-12-22

- Introduced `pinout()`
- Bug fix with `DigitalInputDevice.when_deactivated` decorator
- Documentation tidy up and minor fixes

5.6.3 0.4.0 - 2022-11-18

- Introduced `Servo` class
- Documentation fixes

5.6.4 0.3.0 - 2022-08-12

- Introduced `Motor`, `Robot`, and `DistanceSensor` classes.
- Renamed `LED` factory `use_pwm` parameter to `pwm` to match other classes. **Note:** This is an API breaking change.
- Resolved issue with `RGBLED` when not using `pwm`.
- Resolved issue where `blink` / `pulse` rates of `0` raised a traceback error.
- Other minor bug fixes.
- Documentation updates.

5.6.5 0.2.0 - 2022-06-29

- Pico W compatibility fix for onboard LED.

5.6.6 0.1.1 - 2022-06-08

- Minor fixes for bugs found during testing.
- Small improvements to exception messages.
- Added close methods to Speaker and PWMOutputDevice.
- Added unit tests.
- Added RGBLED.colour as an alias to RGBLED.color.

5.6.7 0.1.0 - 2022-04-08

- Beta release.
- Documentation updates.
- Minor bug fixes and refactoring.

5.6.8 0.0.2 - 2022-03-31

- Bug fixes and documentation updates.

5.6.9 0.0.1 - 2022-03-21

- Initial alpha release to test installation process.

PYTHON MODULE INDEX

INDEX

L

M

O

P

R

S

T

V

W