bluedot Documentation

Release 2.0.0

Martin O'Hanlon

Jan 03, 2023

Contents

1 Getting Started **1**
- 1.1 Installation . . . 1
- 1.2 Pairing . . . 2
- 1.3 Code . . . 2
- 1.4 Connecting . . . 2
- 1.5 Where next . . . 2

2 Pair a Raspberry Pi and Android phone **3**
- 2.1 Using the Desktop . . . 3
- 2.2 Using the Command Line . . . 3

3 Pair 2 Raspberry Pis **5**
- 3.1 Using the Desktop . . . 5
- 3.2 Using the Command Line . . . 5

4 Recipes **7**
- 4.1 Button . . . 7
- 4.2 Joystick . . . 9
- 4.3 Appearance . . . 12
- 4.4 Layout . . . 12
- 4.5 Slider . . . 20
- 4.6 Swiping . . . 21
- 4.7 Rotating . . . 22
- 4.8 Multiple Blue Dot Apps . . . 23
- 4.9 Bluetooth . . . 24
- 4.10 Testing . . . 25

5 Blue Dot Android App **27**
- 5.1 Start . . . 27

6 Blue Dot Python App **29**
- 6.1 Start . . . 29
- 6.2 Options . . . 30

7 Blue Dot API **33**
- 7.1 BlueDot . . . 33
- 7.2 BlueDotButton . . . 37
- 7.3 BlueDotPosition . . . 39
- 7.4 BlueDotInteraction . . . 40
- 7.5 BlueDotSwipe . . . 40
- 7.6 BlueDotRotation . . . 41

8 Bluetooth Comm API **43**
8.1 BluetoothServer . . . 43
8.2 BluetoothClient . . . 45
8.3 BluetoothAdapter . . . 46

9 Mock API **49**
9.1 MockBlueDot . . . 49
9.2 MockBluetoothServer . . . 50
9.3 MockBluetoothClient . . . 51

10 Protocol **53**
10.1 Bluetooth . . . 53
10.2 Specification . . . 53
10.3 Example . . . 55
10.4 Versions . . . 56

11 Build **57**
11.1 Develop . . . 57
11.2 Test . . . 57
11.3 Deploy . . . 58

12 Change log **59**
12.1 Bluedot Python library . . . 59
12.2 Android app . . . 62

Python Module Index **65**

Index **67**

CHAPTER 1

Getting Started

In order to use Blue Dot you will need:

- A Raspberry Pi
  - with built-in Bluetooth (such as the Raspberry Pi 3, 4 or Zero W)
  - or a USB Bluetooth dongle
- An Android phone or 2nd Raspberry Pi for the remote
- An Internet connection (for the install)

1.1 Installation

These instructions assume your Raspberry Pi is running the latest version of Raspbian[1].

1.1.1 Android App

If you're using an Android phone, the Blue Dot app[2] can be installed from the Google Play Store.

1.1.2 Python Library

Open a terminal (click *Menu* → *Accessories* → *Terminal*), then enter:

```
sudo pip3 install bluedot
```

To upgrade to the latest version:

```
sudo pip3 install bluedot --upgrade
```

[1] https://www.raspberrypi.org/downloads/raspbian/
[2] http://play.google.com/store/apps/details?id=com.stuffaboutcode.bluedot

1.2 Pairing

In order to use Blue Dot you will need to pair the Raspberry Pi to the remote *Android phone* (page 3) or *2nd Raspberry Pi* (page 5).

1.3 Code

1. Start up Python 3 (e.g. *Menu* → *Programming* → *Thonny Python IDE*)
2. Create a new program
3. Enter the following code:

```
from bluedot import BlueDot
bd = BlueDot()
bd.wait_for_press()
print("You pressed the blue dot!")
```

4. Save your program as `mydot.py`
5. Run the program:

```
Server started ##:##:##:##:##:##
Waiting for connection
```

Warning: Do not save your program as `bluedot.py` as Python will try and import your program rather than the bluedot module and you will get the error `ImportError: cannot import name BlueDot`.

1.4 Connecting

Start-up the Blue Dot app[3] on your Android phone or run the *Blue Dot Python App* (page 29) on your 2nd Raspberry Pi:

1. Select your Raspberry Pi from the list

Note: Your python program will need to be running and `Waiting for connection` before the BlueDot app will be able to connect to your Raspberry Pi.

2. Press the Blue Dot

1.5 Where next

Check out the *Recipes* (page 7) and the *Blue Dot API* (page 33) documentation for more ideas on using Blue Dot.

[3] http://play.google.com/store/apps/details?id=com.stuffaboutcode.bluedot

CHAPTER 2

Pair a Raspberry Pi and Android phone

2.1 Using the Desktop

On your Android phone:

1. Open Settings
2. Select Bluetooth and make your phone "discoverable"

On your Raspberry Pi:

1. Click *Bluetooth* → *Turn On Bluetooth* (if it's off)
2. Click *Bluetooth* → *Make Discoverable*
3. Click *Bluetooth* → *Add Device*
4. Your phone will appear in the list, select it and click *Pair*

On your Android phone and Raspberry Pi.

1. Confirm the pairing code matches
2. Click OK

Note: You may receive errors relating to services not being able available or being unable to connect: these can be ignored, your phone and Raspberry Pi are now paired.

2.2 Using the Command Line

On your Android phone:

1. Open Settings
2. Select Bluetooth and make your phone "discoverable"

On your Raspberry Pi:

1. Type **`bluetoothctl`** and press Enter to open Bluetooth control
2. At the `[bluetooth]#` prompt enter the following commands:

```
discoverable on
pairable on
agent on
default-agent
scan on
```

3. Wait for a message to appear showing the Android phone has been found:

```
[NEW] Device 12:23:34:45:56:67 devicename
```

4. Type pair with the mac address of your Android phone:

```
pair 12:23:34:45:56:67
```

On your Android phone and Raspberry Pi.

1. Confirm the passcode.
2. Type **quit** and press Enter to return to the command line

CHAPTER 3

Pair 2 Raspberry Pis

The instructions below describe pairing a couple of Raspberry Pis which either have built-in Bluetooth (the Pi 3B or the Pi Zero W) or a USB Bluetooth dongle.

3.1 Using the Desktop

On the first Raspberry Pi:

1. Click *Bluetooth* → *Turn On Bluetooth* (if it's off)
2. Click *Bluetooth* → *Make Discoverable*

On the second Raspberry Pi:

1. Click *Bluetooth* → *Turn On Bluetooth* (if it's off)
2. Click *Bluetooth* → *Make Discoverable*
3. Click *Bluetooth* → *Add Device*
4. The first Pi will appear in the list: select it and click the *Pair* button

On the first Raspberry Pi again:

1. Accept the pairing request

Note: You may receive errors relating to services not being able available or being unable to connect: these can be ignored.

3.2 Using the Command Line

On the first Raspberry Pi:

1. Enter **`bluetoothctl`** to open Bluetooth control
2. At the `[bluetooth]#` prompt enter the following commands:

```
discoverable on
pairable on
agent on
default-agent
```

On the second Raspberry Pi:

1. Enter **bluetoothctl** to open Bluetooth control
2. At the [bluetooth]# prompt enter the following commands:

```
discoverable on
pairable on
agent on
default-agent
scan on
```

3. Wait for a message to appear showing the first Pi has been found:

```
[NEW] Device 12:23:34:45:56:67 devicename
```

4. Type pair with the mac address of the first Pi:

```
pair 12:23:34:45:56:67
```

On both Raspberry Pi's:

1. Confirm the passcode.
2. Type **quit** and press Enter to return to the command line

CHAPTER 4

Recipes

The recipes provide examples of how you can use Blue Dot. Don't be restricted by these ideas and be sure to have a look at the *Blue Dot API* (page 33) as there is more to be discovered.

4.1 Button

The simplest way to use the Blue Dot is as a wireless button.

4.1.1 Hello World

Let's say "Hello World" by creating the *BlueDot* (page 33) object then waiting for the Blue Dot app to connect and the button be pressed:

```
from bluedot import BlueDot
bd = BlueDot()
bd.wait_for_press()
print("Hello World")
```

Alternatively you can also use `when_pressed` to call a function:

```
from bluedot import BlueDot
from signal import pause

def say_hello():
    print("Hello World")

bd = BlueDot()
bd.when_pressed = say_hello

pause()
```

`wait_for_release` and `when_released` also allow you to interact when the button is released:

```
from bluedot import BlueDot
from signal import pause

```

(continues on next page)

(continued from previous page)

```
def say_hello():
    print("Hello World")

def say_goodbye():
    print("goodbye")

bd = BlueDot()
bd.when_pressed = say_hello
bd.when_released = say_goodbye

pause()
```

Double presses can also be used with `wait_for_double_press` and `when_double_pressed`:

```
from bluedot import BlueDot
from signal import pause

def shout_hello():
    print("HELLO")

bd = BlueDot()
bd.when_double_pressed = shout_hello

pause()
```

4.1.2 Flash an LED

Using Blue Dot in combination with `gpiozero`[4] you can interact with electronic components, such as LEDs, connected to your Raspberry Pi.

When a button is pressed, the LED connected to GPIO 27 will turn on; when released it will turn off:

```
import os
from bluedot import BlueDot
from gpiozero import LED

bd = BlueDot()
led = LED(27)

bd.wait_for_press()
led.on()

bd.wait_for_release()
led.off()
```

You could also use `when_pressed` and `when_released`:

```
from bluedot import BlueDot
from gpiozero import LED
from signal import pause

bd = BlueDot()
led = LED(27)

bd.when_pressed = led.on
bd.when_released = led.off

pause()
```

[4] https://gpiozero.readthedocs.io/en/latest/recipes.html#module-gpiozero

Alternatively use `source`[5] and `values`:

```
from bluedot import BlueDot
from gpiozero import LED
from signal import pause

bd = BlueDot()
led = LED(27)

led.source = bd.values

pause()
```

4.1.3 Remote Camera

Using a Raspberry Pi camera module, `picamera.PiCamera` and *BlueDot* (page 33), you can really easily create a remote camera:

```
from bluedot import BlueDot
from picamera import PiCamera
from signal import pause

bd = BlueDot()
cam = PiCamera()

def take_picture():
    cam.capture("pic.jpg")

bd.when_pressed = take_picture

pause()
```

4.2 Joystick

The Blue Dot can also be used as a joystick when the middle, top, bottom, left or right areas of the dot are touched.

4.2.1 D-pad

Using the position the Blue Dot was pressed you can work out whether it was pressed to go up, down, left, right like the D-pad[6] on a joystick:

```
from bluedot import BlueDot
from signal import pause

def dpad(pos):
    if pos.top:
        print("up")
    elif pos.bottom:
        print("down")
    elif pos.left:
        print("left")
    elif pos.right:
        print("right")
    elif pos.middle:
```

(continues on next page)

[5] https://gpiozero.readthedocs.io/en/latest/api_generic.html#gpiozero.SourceMixin.source
[6] https://en.wikipedia.org/wiki/D-pad

(continued from previous page)

```
        print("fire")

bd = BlueDot()
bd.when_pressed = dpad

pause()
```

At the moment the D-pad[7] only registers when it is pressed. To get it work when the position is moved you should add the following line above `pause()`:

```
bd.when_moved = dpad
```

4.2.2 Robot

These recipes assume your robot is constructed with a pair of H-bridges. The forward and backward pins for the H-bridge of the left wheel are 17 and 18 respectively, and the forward and backward pins for H-bridge of the right wheel are 22 and 23 respectively.

Using the Blue Dot and `gpiozero.Robot`[8], you can create a bluetooth controlled robot[9] which moves when the dot is pressed and stops when it is released:

```
from bluedot import BlueDot
from gpiozero import Robot
from signal import pause

bd = BlueDot()
robot = Robot(left=(17, 18), right=(22, 23))

def move(pos):
    if pos.top:
        robot.forward()
    elif pos.bottom:
        robot.backward()
    elif pos.left:
        robot.left()
    elif pos.right:
        robot.right()

def stop():
    robot.stop()

bd.when_pressed = move
bd.when_moved = move
bd.when_released = stop

pause()
```

4.2.3 Variable Speed Robot

You can change the robot to use variable speeds, so the further towards the edge you press the Blue Dot, the faster the robot will go.

The *`distance`* (page 39) attribute returns how far from the centre the Blue Dot was pressed, which can be passed to the robot's functions to change its speed:

[7] https://en.wikipedia.org/wiki/D-pad
[8] https://gpiozero.readthedocs.io/en/latest/api_boards.html#gpiozero.Robot
[9] https://youtu.be/eW9oEPySF58

```
from bluedot import BlueDot
from gpiozero import Robot
from signal import pause

bd = BlueDot()
robot = Robot(left=(lfpin, lbpin), right=(rfpin, rbpin))

def move(pos):
    if pos.top:
        robot.forward(pos.distance)
    elif pos.bottom:
        robot.backward(pos.distance)
    elif pos.left:
        robot.left(pos.distance)
    elif pos.right:
        robot.right(pos.distance)

def stop():
    robot.stop()

bd.when_pressed = move
bd.when_moved = move
bd.when_released = stop

pause()
```

Alternatively you can use a generator and yield (x, y) values to the `gpiozero.Robot.source` property (courtesy of Ben Nuttall[10]):

```
from gpiozero import Robot
from bluedot import BlueDot
from signal import pause

def pos_to_values(x, y):
    left = y if x > 0 else y + x
    right = y if x < 0 else y - x
    return (clamped(left), clamped(right))

def clamped(v):
    return max(-1, min(1, v))

def drive():
    while True:
        if bd.is_pressed:
            x, y = bd.position.x, bd.position.y
            yield pos_to_values(x, y)
        else:
            yield (0, 0)

robot = Robot(left=(lfpin, lbpin), right=(rfpin, rbpin))
bd = BlueDot()

robot.source = drive()

pause()
```

[10] https://github.com/bennuttall

4.3 Appearance

The button doesn't have to be blue or a dot, you can change how it looks, or make it completely invisible.

4.3.1 Colo(u)r

To change the color of the button use the *color* (page 35): property:

```
from bluedot import BlueDot
bd = BlueDot()
bd.color = "red"
```

A dictionary of available colors can be obtained from `bluedot.COLORS`.

The color can also be set using a hex value of *#rrggbb* or *#rrggbbaa* value:

```
bd.color = "#00ff00"
```

Or a tuple of 3 or 4 integers between *0* and *255* either (red, gree, blue) or (red, green, blue, alpha):

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

4.3.2 Square

The button can also be made square using the *square* (page 36): property:

```
from bluedot import BlueDot
bd = BlueDot()
bd.square = True
```

4.3.3 Border

A border can also been added to the button by setting the *border* (page 35): property to *True*:

```
from bluedot import BlueDot
bd = BlueDot()
bd.border = True
```

4.3.4 (In)visible

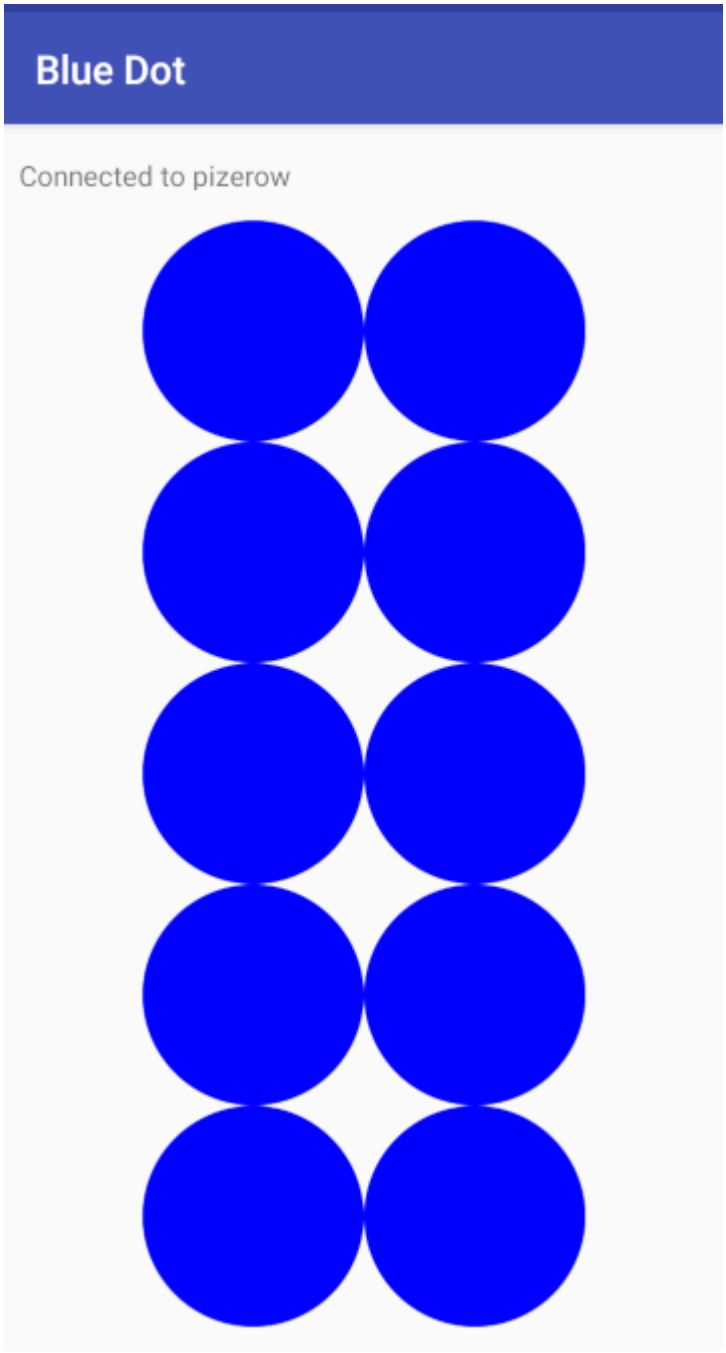
The button can be hidden and shown using the *visible* (page 36): property:

```
from bluedot import BlueDot
bd = BlueDot()
bd.visible = False
```

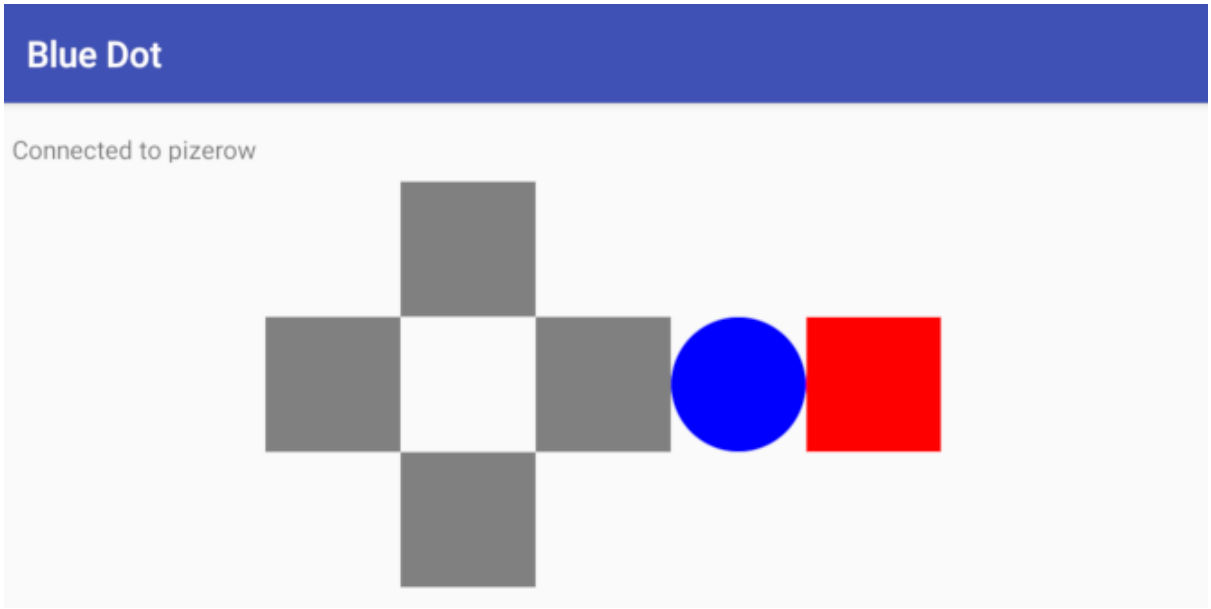
4.4 Layout

You can have as many buttons as you want.

The Buttons need to be in a grid of columns and rows.



By hiding specific buttons and being creative with the button's appearance you can create very sophisticated layouts for your controllers using Blue Dot.

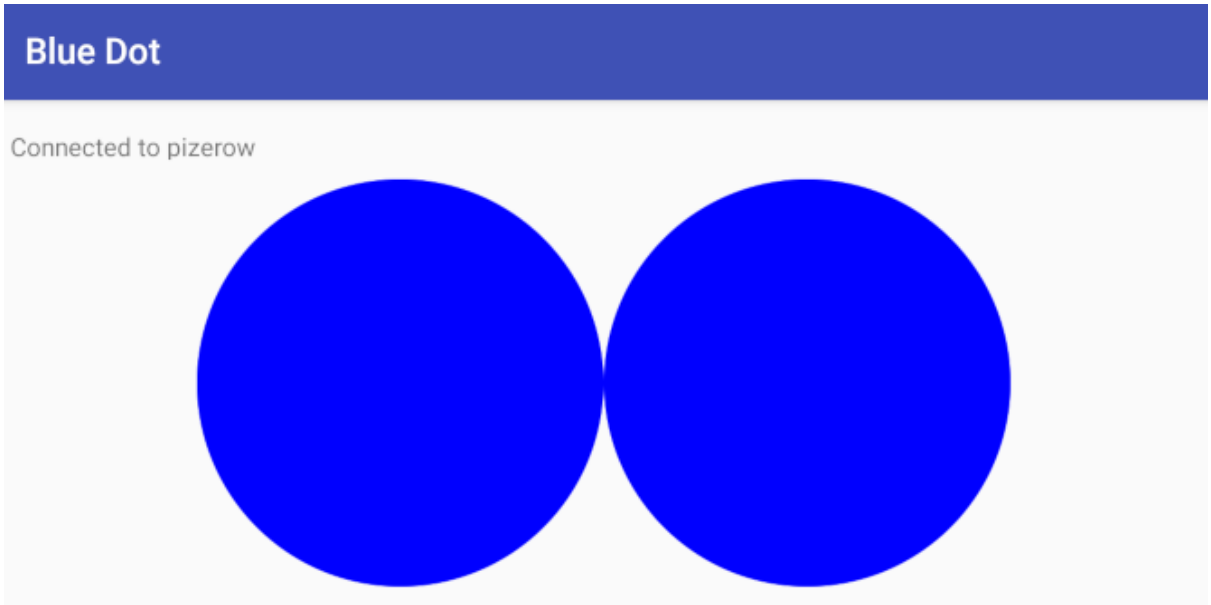


The Blue Dot android app supports multi touch allowing you to use multiple buttons simultaneously

Note: Currently only the Android client app supports multi buttons.

4.4.1 Two Buttons

Create 2 buttons side by side, by setting the number of *cols* to *2*:




```
from bluedot import BlueDot
from signal import pause

def pressed(pos):
    print("button {}.{} pressed".format(pos.col, pos.row))

bd = BlueDot(cols=2)
bd.when_pressed = pressed
```

(continues on next page)

(continued from previous page)

```
pause()
```

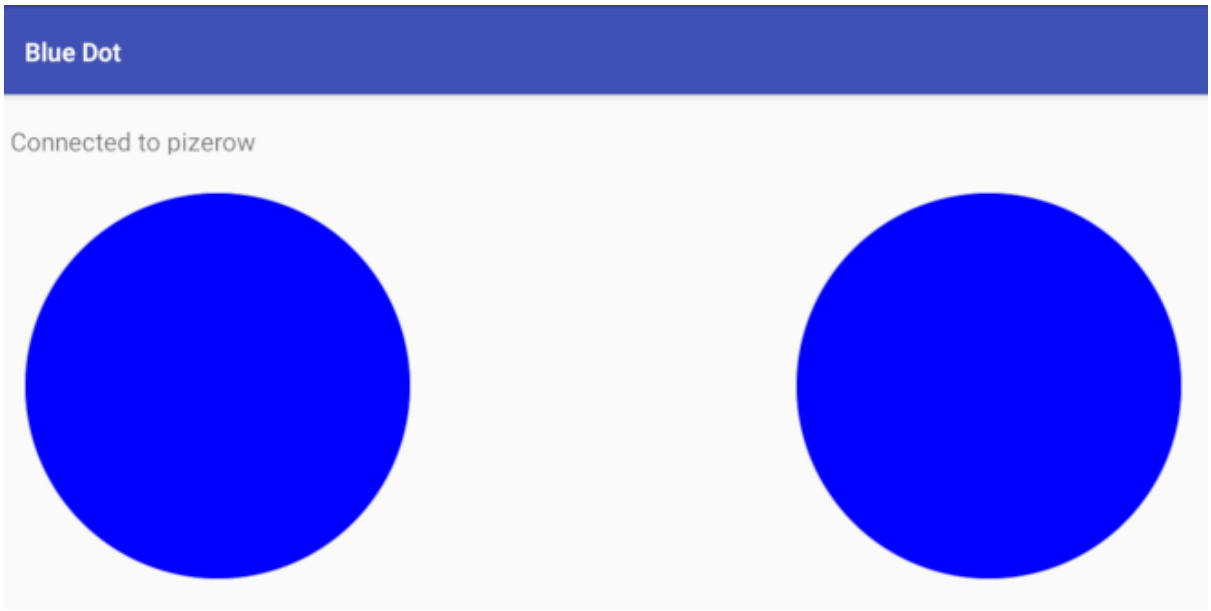
The buttons could be made verticle by setting the *rows* attribute:

```
bd = BlueDot(rows=2)
```

Each button can be set to call its own function by using the grid position:

```
from bluedot import BlueDot
from signal import pause

def pressed_1(pos):
    print("button 1 pressed")

def pressed_2(pos):
    print("button 2 pressed")

bd = BlueDot(cols=2, rows=1)

bd[0,0].when_pressed = pressed_1
bd[1,0].when_pressed = pressed_2

pause()
```

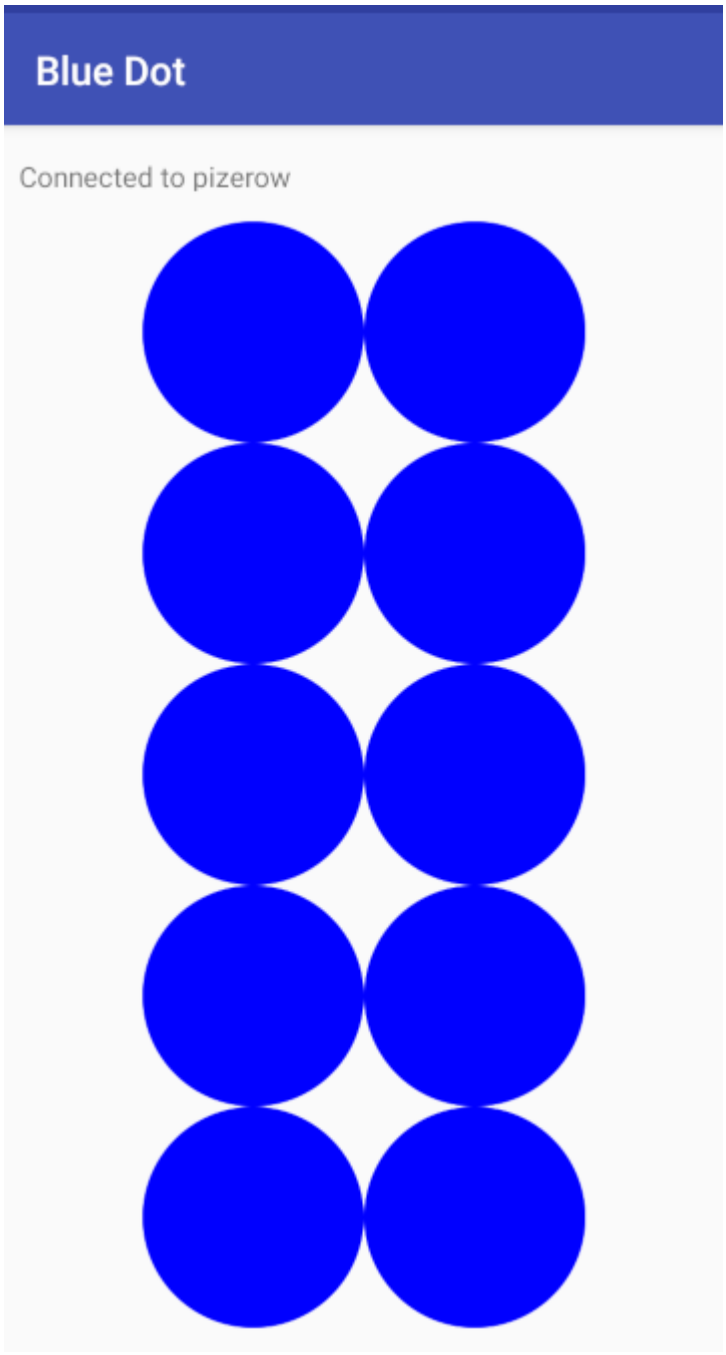
To create a gap in between the buttons you could create a row of 3 buttons and hide the middle button:




```
from bluedot import BlueDot
from signal import pause

def pressed(pos):
    print("button {}.{} pressed".format(pos.col, pos.row))

bd = BlueDot(cols=3, rows=1)
bd[1,0].visible = False
bd.when_pressed = pressed

pause()
```

4.4.2 Many Buttons

Create a grid of buttons by setting the *cols* and *rows* e.g. 10 buttons in a 2x5 grid:



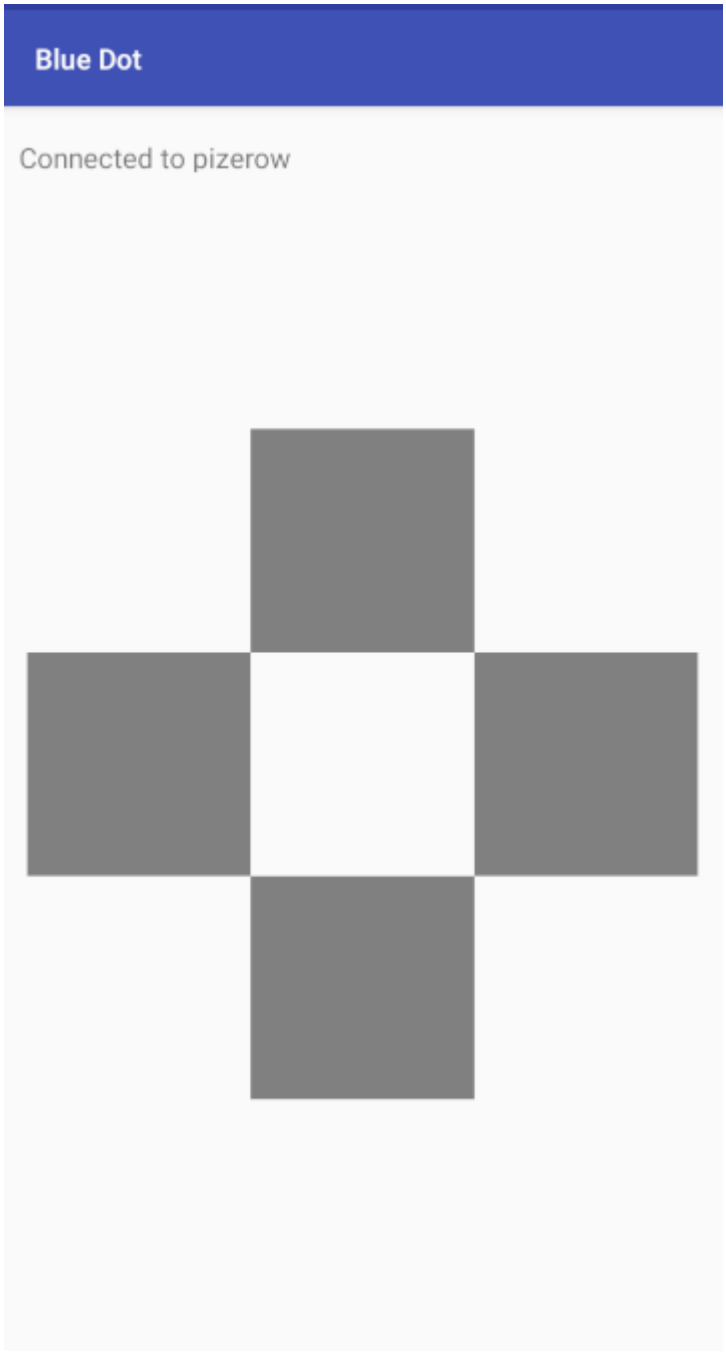

```
from bluedot import BlueDot
from signal import pause

def pressed(pos):
    print("button {}.{} pressed".format(pos.col, pos.row))

bd = BlueDot(cols=2, rows=5)
bd.when_pressed = pressed

pause()
```

You could assign all the buttons random colors:

```
from bluedot import BlueDot, COLORS
from random import choice
from signal import pause

def pressed(pos):
    print("button {}.{} pressed".format(pos.col, pos.row))

bd = BlueDot(cols=2, rows=5)
bd.when_pressed = pressed

for button in bd.buttons:
    button.color = choice(list(COLORS.values()))

pause()
```

4.4.3 D-pad

Create a traditional d-pad layout by using a 3x3 grid and hide the buttons at the corners and in the middle:



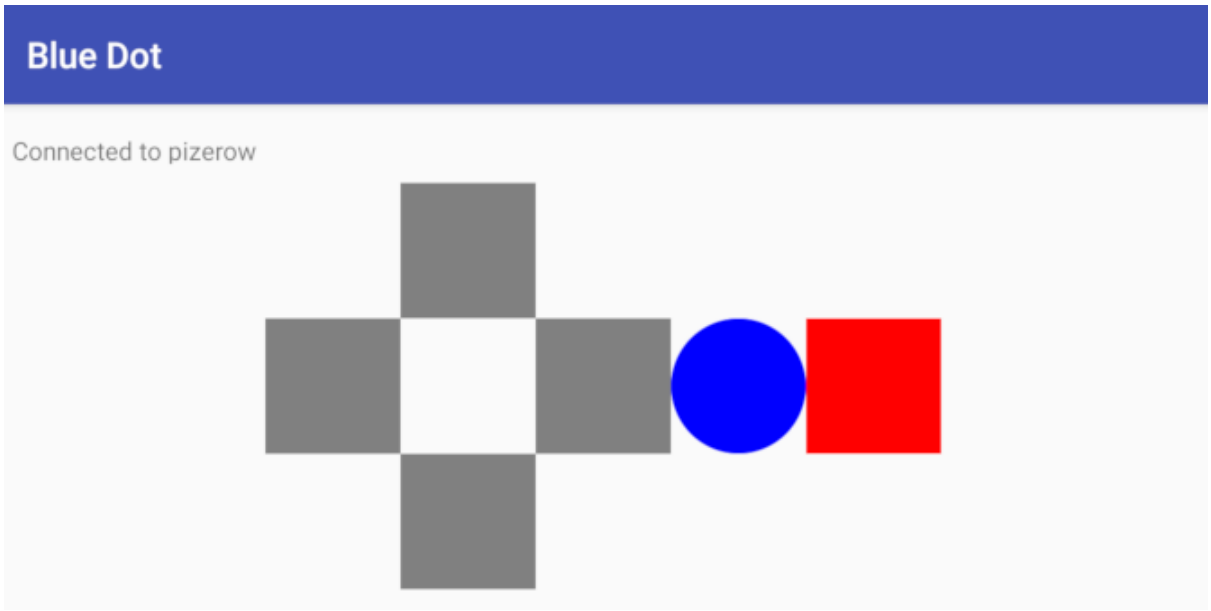

```
from bluedot import BlueDot
from signal import pause

def up():
    print("up")

def down():
    print("down")

def left():
    print("left")

def right():
    print("right")
```

(continues on next page)

(continued from previous page)

```
bd = BlueDot(cols=3, rows=3)
bd.color = "gray"
bd.square = True

bd[0,0].visible = False
bd[2,0].visible = False
bd[0,2].visible = False
bd[2,2].visible = False
bd[1,1].visible = False

bd[1,0].when_pressed = up
bd[1,2].when_pressed = down
bd[0,1].when_pressed = left
bd[2,1].when_pressed = right

pause()
```

Add 2 buttons on the right to create a joypad:




```
from bluedot import BlueDot
from signal import pause

def up():
    print("up")

def down():
    print("down")

def left():
    print("left")

def right():
    print("right")

bd = BlueDot(cols=3, rows=3)
bd.color = "gray"
bd.square = True

bd[0,0].visible = False
bd[2,0].visible = False
```

(continues on next page)

(continued from previous page)

```
bd[0,2].visible = False
bd[2,2].visible = False
bd[1,1].visible = False

bd[1,0].when_pressed = up
bd[1,2].when_pressed = down
bd[0,1].when_pressed = left
bd[2,1].when_pressed = right

pause()
```

4.5 Slider

By holding down a button and moving the position you can use it as an analogue slider.

4.5.1 Centre Out

Using the *BlueDotPosition.distance* (page 39) property which is returned when the position is moved you can create a slider which goes from the centre out in any direction:

```
from bluedot import BlueDot
from signal import pause

def show_percentage(pos):
    percentage = round(pos.distance * 100, 2)
    print("{}%".format(percentage))

bd = BlueDot()
bd.when_moved = show_percentage

pause()
```

4.5.2 Left to Right

The *BlueDotPosition.x* (page 39) property returns a value from -1 (far left) to 1 (far right). Using this value you can create a slider which goes horizontally through the middle:

```
from bluedot import BlueDot
from signal import pause

def show_percentage(pos):
    horizontal = ((pos.x + 1) / 2)
    percentage = round(horizontal * 100, 2)
    print("{}%".format(percentage))

bd = BlueDot()
bd.when_moved = show_percentage

pause()
```

To make a vertical slider you could change the code above to use *BlueDotPosition.y* (page 39) instead.

4.5.3 Dimmer Switch

Using the gpiozero.PWMLED[11] class and *BlueDot* (page 33) as a vertical slider you can create a wireless dimmer switch:

```
from bluedot import BlueDot
from gpiozero import PWMLED
from signal import pause

def set_brightness(pos):
    brightness = (pos.y + 1) / 2
    led.value = brightness

led = PWMLED(27)
bd = BlueDot()
bd.when_moved = set_brightness

pause()
```

4.6 Swiping

You can interact with the Blue Dot by swiping across it, like you would to move between pages in a mobile app.

4.6.1 Single

Detecting a single swipe is easy using `wait_for_swipe`:

```
from bluedot import BlueDot
bd = BlueDot()
bd.wait_for_swipe()
print("Blue Dot swiped")
```

Alternatively you can also use `when_swiped` to call a function:

```
from bluedot import BlueDot
from signal import pause

def swiped():
    print("Blue Dot swiped")

bd = BlueDot()
bd.when_swiped = swiped

pause()
```

4.6.2 Direction

You can tell what direction the Blue Dot is swiped by using the *BlueDotSwipe* (page 40) object passed to the function assigned to `when_swiped`:

```
from bluedot import BlueDot
from signal import pause

def swiped(swipe):
    if swipe.up:
```

(continues on next page)

[11] https://gpiozero.readthedocs.io/en/latest/api_output.html#gpiozero.PWMLED

(continued from previous page)

```
        print("up")
    elif swipe.down:
        print("down")
    elif swipe.left:
        print("left")
    elif swipe.right:
        print("right")

bd = BlueDot()
bd.when_swiped = swiped

pause()
```

4.6.3 Speed, Angle, and Distance

BlueDotSwipe (page 40) returns more than just the direction. It also includes the speed of the swipe (in Blue Dot radius per second), the angle, and the distance between the start and end positions of the swipe:

```
from bluedot import BlueDot
from signal import pause

def swiped(swipe):
    print("Swiped")
    print("speed={}".format(swipe.speed))
    print("angle={}".format(swipe.angle))
    print("distance={}".format(swipe.distance))

bd = BlueDot()
bd.when_swiped = swiped

pause()
```

4.7 Rotating

You can use Blue Dot like a rotary encoder or "iPod classic click wheel" - rotating around the outer edge of the Blue Dot will cause it to "tick". The Blue Dot is split into a number of virtual segments (the default is 8), when the position moves from one segment to another, it ticks.

4.7.1 Counter

Using the `when_rotated` callback you can create a counter which increments / decrements when the Blue Dot is rotated either clockwise or anti-clockwise. A *BlueDotRotation* (page 41) object is passed to the callback. Its *value* (page 41) property will be -1 if rotated anti-clockwise and 1 if rotated clockwise:

```
from bluedot import BlueDot
from signal import pause

count = 0

def rotated(rotation):
    global count
    count += rotation.value

    print("{} {} {}".format(count,
                            rotation.clockwise,
```

(continues on next page)

(continued from previous page)

```
                            rotation.anti_clockwise))

bd = BlueDot()
bd.when_rotated = rotated

pause()
```

The rotation speed can be modified using the *BlueDot.rotation_segments* (page 36) property which changes the number of segments the Blue Dot is split into:

```
bd.rotation_segments = 16
```

4.8 Multiple Blue Dot Apps

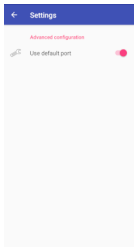
You can connect multiple Blue Dot clients (apps) to a single server (python program) by using different Bluetooth ports for each app.

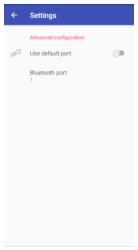
Create multiple *BlueDot* servers using specific ports:

```
from bluedot import BlueDot
from signal import pause

def bd1_pressed():
    print("BlueDot 1 pressed")

def bd2_pressed():
    print("BlueDot 2 pressed")

bd1 = BlueDot(port = 1)
bd2 = BlueDot(port = 2)

bd1.when_pressed = bd1_pressed
bd2.when_pressed = bd2_pressed

pause()
```

Change the BlueDot app to use the specific port by:

1. Opening settings from the menu
2. Turning *Use default port* off
3. Selecting the specific *Bluetooth port*





4.9 Bluetooth

You can interact with the Bluetooth adapter using *BlueDot* (page 33).

4.9.1 Pairing

You can put your Raspberry Pi into pairing mode which will allow pairing from other devices for 60 seconds:

```
from bluedot import BlueDot
from signal import pause

bd = BlueDot()
bd.allow_pairing()

pause()
```

Or connect up a physical button up to start the pairing (the button is assumed to be wired to GPIO 27):

```
from bluedot import BlueDot
from gpiozero import Button
from signal import pause

bd = BlueDot()
button = Button(27)

button.when_pressed = bd.allow_pairing

pause()
```

4.9.2 Paired Devices

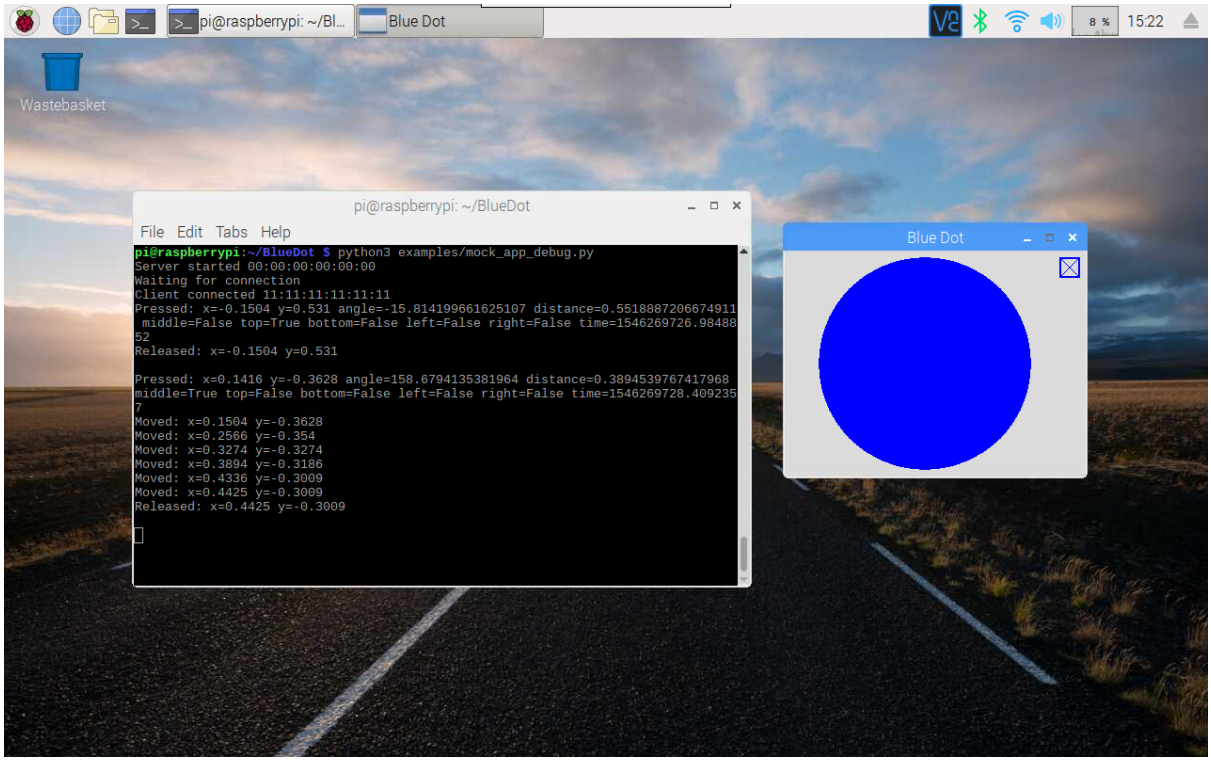
You can iterate over the devices that your Raspberry Pi is paired too:

```
from bluedot import BlueDot
bd = BlueDot()

devices = bd.paired_devices
for d in devices:
    device_address = d[0]
    device_name = d[1]
```

4.10 Testing

Blue Dot includes a `MockBlueDot` class to allow you to test and debug your program without having to use Bluetooth or a Blue Dot client.

`MockBlueDot` inherits from *BlueDot* (page 33) and is used in the same way, but you have the option of launching a mock app which you can click with a mouse or writing scripts to simulate the Blue Dot being used.



4.10.1 Mock App

Launch the mock Blue Dot app to test by clicking the on-screen dot with the mouse:

```
from bluedot import MockBlueDot
from signal import pause

def say_hello():
    print("Hello World")

bd = MockBlueDot()
bd.when_pressed = say_hello

bd.launch_mock_app()
pause()
```

4.10.2 Scripted Tests

Tests can also be scripted using `MockBlueDot`:

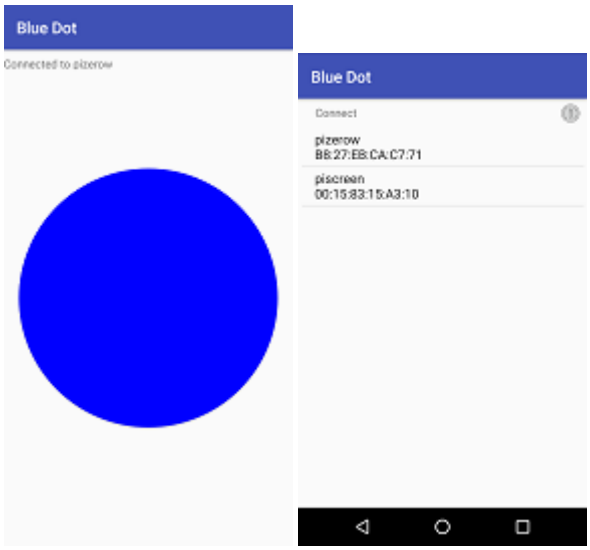
```
from bluedot import MockBlueDot

def say_hello():
    print("Hello World")

bd = MockBlueDot()
bd.when_pressed = say_hello

bd.mock_client_connected()
bd.mock_blue_dot_pressed(0,0)
```

CHAPTER 5

Blue Dot Android App

The Blue Dot app[12] is available to download from the Google Play store.

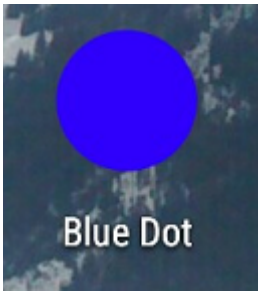
Please leave a rating and review if you find Blue Dot useful :)



5.1 Start

1. Download the Blue Dot app[13] from the Google Play store.
2. If you havent already done so, pair your raspberry pi as described in the *Getting Started* (page 1) guide.
3. Run the Blue Dot app

[12] http://play.google.com/store/apps/details?id=com.stuffaboutcode.bluedot
[13] http://play.google.com/store/apps/details?id=com.stuffaboutcode.bluedot



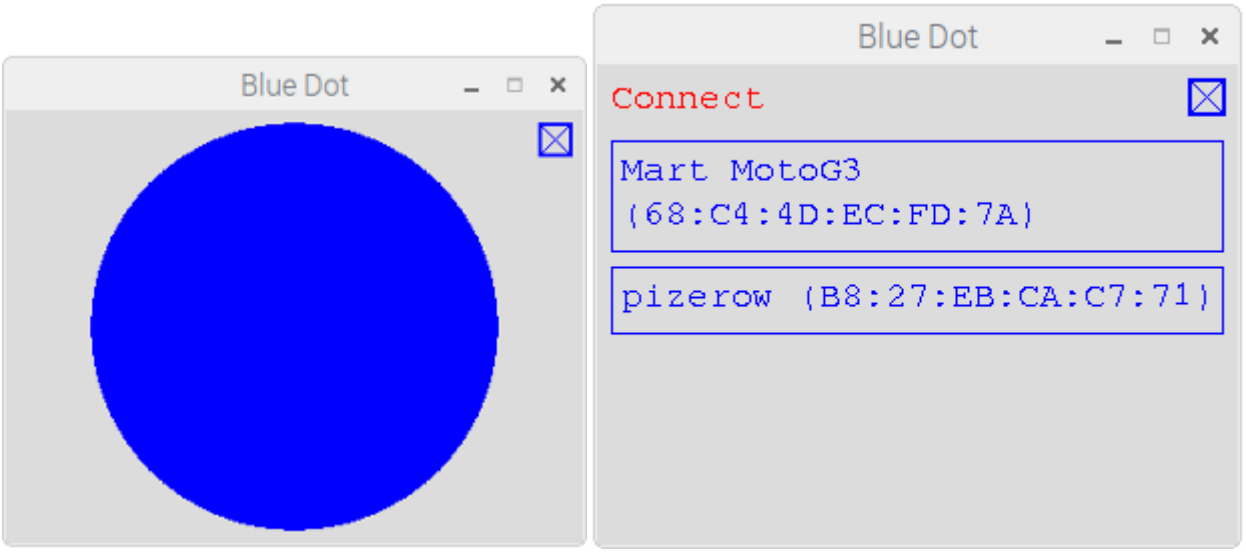
4. Select your Raspberry Pi from the paired devices list

5. Press the Dot

CHAPTER 6

Blue Dot Python App

Blue Dot Python app allows you to use another Raspberry Pi (or linux based computer) as the Blue Dot remote.



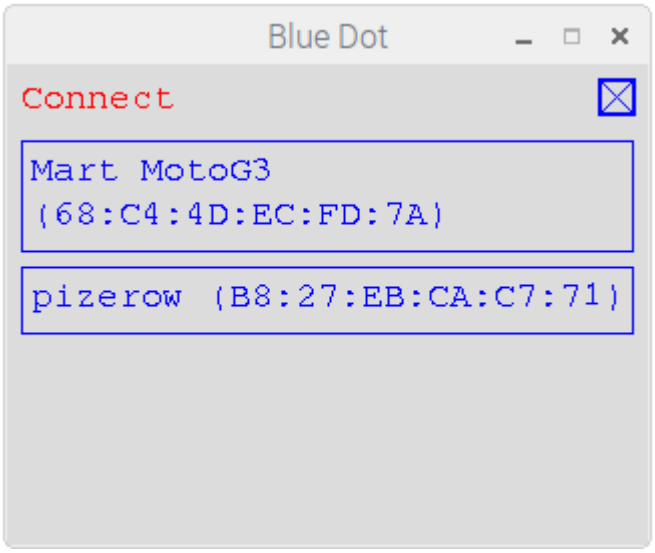
6.1 Start

The app is included in the bluedot Python library:

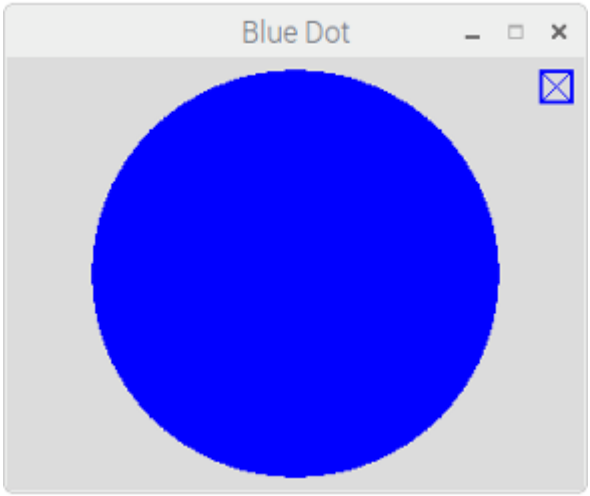
1. If you havent already done so, pair your raspberry pi and install the Python library as described in the *Getting Started* (page 1) guide
2. Run the Blue Dot app:

```
bluedotapp
```

3. Select your Raspberry Pi from the paired devices list



4. Press the Dot



6.2 Options

To get help with the Blue Dot app options:

```
bluedotapp --help
```

If you have more than 1 bluetooth device you can use `--device` to use a particular device:

```
bluedotapp --device hci1
```

You can specify the server to connect to at startup by using the `--server` option:

```
bluedotapp --server myraspberrypi
```

The screen size of the Blue Dot app can be changed using the `width` and `height` options and specifying the number of pixels:

```
bluedotapp --width 500 --height 500
```

The app can also be used full screen, if no `width` or `height` is given the screen will be sized to the current resolution of the screen:

```
bluedotapp --fullscreen
```

CHAPTER 7

Blue Dot API

7.1 BlueDot

class bluedot.**BlueDot**(*device='hci0'*, *port=1*, *auto_start_server=True*, *power_up_device=False*, *print_messages=True*, *cols=1*, *rows=1*)

Interacts with a Blue Dot client application, communicating when and where a button has been pressed, released or held.

This class starts an instance of *btcomm.BluetoothServer* (page 43) which manages the connection with the Blue Dot client.

This class is intended for use with a Blue Dot client application.

The following example will print a message when the Blue Dot button is pressed:

```
from bluedot import BlueDot
bd = BlueDot()
bd.wait_for_press()
print("The button was pressed")
```

Multiple buttons can be created, by changing the number of columns and rows. Each button can be referenced using its [col, row]:

```
bd = BlueDot(cols=2, rows=2)
bd[0,0].wait_for_press()
print("Top left button pressed")
bd[1,1].wait_for_press()
print("Bottom right button pressed")
```

Parameters

- **device** (*str*[14]) – The Bluetooth device the server should use, the default is "hci0", if your device only has 1 Bluetooth adapter this shouldn't need to be changed.
- **port** (*int*[15]) – The Bluetooth port the server should use, the default is 1, and under normal use this should never need to change.

[14] https://docs.python.org/3.5/library/stdtypes.html#str
[15] https://docs.python.org/3.5/library/functions.html#int

- **auto_start_server** (*bool*[16]) – If `True` (the default), the Bluetooth server will be automatically started on initialisation; if `False`, the method *start()* (page 35) will need to be called before connections will be accepted.
- **power_up_device** (*bool*[17]) – If `True`, the Bluetooth device will be powered up (if required) when the server starts. The default is `False`.

  Depending on how Bluetooth has been powered down, you may need to use **rfkill** to unblock Bluetooth to give permission to bluez to power on Bluetooth:

```
sudo rfkill unblock bluetooth
```

- **print_messages** (*bool*[18]) – If `True` (the default), server status messages will be printed stating when the server has started and when clients connect / disconnect.
- **cols** (*int*[19]) – The number of columns in the grid of buttons. Defaults to `1`.
- **rows** (*int*[20]) – The number of rows in the grid of buttons. Defaults to `1`.

allow_pairing(*timeout=60*)

Allow a Bluetooth device to pair with your Raspberry Pi by putting the adapter into discoverable and pairable mode.

Parameters **timeout** (*int*[21]) – The time in seconds the adapter will remain pairable. If set to `None` the device will be discoverable and pairable indefinetly.

resize(*cols*, *rows*)

Resizes the grid of buttons.

Parameters

- **cols** (*int*[22]) – The number of columns in the grid of buttons.
- **rows** (*int*[23]) – The number of rows in the grid of buttons.

Note: Existing buttons will retain their state (color, border, etc) when resized. New buttons will be created with the default values set by the *BlueDot* (page 33).

---

set_when_client_connects(*callback*, *background=False*)

Sets the function which is called when a Blue Dot connects.

Parameters

- **callback** (*Callable*) – The function to call, setting to *None* will stop the callback.
- **background** (*bool*[24]) – If set to *True* the function will be run in a separate thread and it will return immediately. The default is *False*.

set_when_client_disconnects(*callback*, *background=False*)

Sets the function which is called when a Blue Dot disconnects.

Parameters

- **callback** (*Callable*) – The function to call, setting to *None* will stop the callback.

[16] https://docs.python.org/3.5/library/functions.html#bool
[17] https://docs.python.org/3.5/library/functions.html#bool
[18] https://docs.python.org/3.5/library/functions.html#bool
[19] https://docs.python.org/3.5/library/functions.html#int
[20] https://docs.python.org/3.5/library/functions.html#int
[21] https://docs.python.org/3.5/library/functions.html#int
[22] https://docs.python.org/3.5/library/functions.html#int
[23] https://docs.python.org/3.5/library/functions.html#int
[24] https://docs.python.org/3.5/library/functions.html#bool

- **background** (*bool*[25]) – If set to *True* the function will be run in a separate thread and it will return immediately. The default is *False*.

start()
Start the *btcomm.BluetoothServer* (page 43) if it is not already running. By default the server is started at initialisation.

stop()
Stop the Bluetooth server.

wait_for_connection(*timeout=None*)
Waits until a Blue Dot client connects. Returns `True` if a client connects.

Parameters **timeout** (*float*[26]) – Number of seconds to wait for a wait connections, if `None` (the default), it will wait indefinetly for a connection from a Blue Dot client.

adapter
The *btcomm.BluetoothAdapter* (page 46) instance that is being used.

border
When set to *True* adds a border to the dot. Default is *False*.

Note: If there are multiple buttons in the grid, the 'default' value will be returned and when set all buttons will be updated.

---

buttons
A list of *BlueDotButton* (page 37) objects in the "grid".

color
Sets or returns the color of the button. Defaults to BLUE.

An instance of `colors.Color` is returned.

Value can be set as a `colors.Color` object, a hex color value in the format *#rrggbb* or *#rrggbbaa*, a tuple of *(red, green, blue)* or *(red, green, blue, alpha)* values between *0* & *255* or a text description of the color, e.g. "red".

A dictionary of available colors can be obtained from *bluedot.COLORS*.

Note: If there are multiple buttons in the grid, the 'default' value will be returned and when set all buttons will be updated.

---

cols
Sets or returns the number of columns in the grid of buttons.

device
The Bluetooth device the server is using. This defaults to "hci0".

double_press_time
Sets or returns the time threshold in seconds for a double press. Defaults to 0.3.

Note: If there are multiple buttons in the grid, the 'default' value will be returned and when set all buttons will be updated.

---

interaction
Returns an instance of *BlueDotInteraction* (page 40) representing the current or last interaction with the Blue Dot.

[25] https://docs.python.org/3.5/library/functions.html#bool
[26] https://docs.python.org/3.5/library/functions.html#float

Note: If the Blue Dot is released (and inactive), *interaction* (page 35) will return the interaction when it was released, until it is pressed again. If the Blue Dot has never been pressed *interaction* (page 35) will return `None`.

If there are multiple buttons, the interaction will only be returned for button [0,0]

Deprecated since version 2.0.0.

is_connected
Returns `True` if a Blue Dot client is connected.

is_pressed
Returns `True` if the button is pressed (or held).

Note: If there are multiple buttons, if any button is pressed, *True* will be returned.

paired_devices
Returns a sequence of devices paired with this adapter `[(mac_address, name), (mac_address, name), ...]`:

```
bd = BlueDot()
devices = bd.paired_devices
for d in devices:
    device_address = d[0]
    device_name = d[1]
```

port
The port the server is using. This defaults to 1.

print_messages
When set to `True` messages relating to the status of the Bluetooth server will be printed.

rotation_segments
Sets or returns the number of virtual segments the button is split into for rotating. Defaults to 8.

Note: If there are multiple buttons in the grid, the 'default' value will be returned and when set all buttons will be updated.

rows
Sets or returns the number of rows in the grid of buttons.

running
Returns a `True` if the server is running.

server
The *btcomm.BluetoothServer* (page 43) instance that is being used to communicate with clients.

square
When set to *True* the 'dot' is made square. Default is *False*.

Note: If there are multiple buttons in the grid, the 'default' value will be returned and when set all buttons will be updated.

visible
When set to *False* the dot will be hidden. Default is *True*.

Note: Events (press, release, moved) are still sent from the dot when it is not visible.

If there are multiple buttons in the grid, the 'default' value will be returned and when set all buttons will be updated.

---

when_client_connects
Sets or returns the function which is called when a Blue Dot application connects.

The function will be run in the same thread and block, to run in a separate thread use *set_when_client_connects(function, background=True)*

when_client_disconnects
Sets or returns the function which is called when a Blue Dot disconnects.

The function will be run in the same thread and block, to run in a separate thread use *set_when_client_disconnects(function, background=True)*

7.2 BlueDotButton

class bluedot.**BlueDotButton**(*bd*, *col*, *row*, *color*, *square*, *border*, *visible*)
Represents a single button on the button client applications. It keeps tracks of when and where the button has been pressed and processes any events.

This class is intended for use via *BlueDot* (page 33) and should not be instantiated "manually".

A button can be interacted with individually via *BlueDot* (page 33) by stating its position in the grid e.g.

```
from bluedot import BlueDot
bd = BlueDot()

first_button = bd[0,0].wait_for_press

first_button.wait_for_press()
print("The first button was pressed")
```

Parameters

- **bd** (BlueDot (page 33)) – The BlueDot object this button belongs too.
- **col** (*int*[27]) – The column position for this button in the grid.
- **col** – The row position for this button in the grid.

:param string color The color of the button.

Can be set as a colors.Color object, a hex color value in the format *#rrggbb* or *#rrggbbaa*, a tuple of *(red, green, blue)* or *(red, green, blue, alpha)* values between *0* & *255* or a text description of the color, e.g. "red".

A dictionary of available colors can be obtained from *bluedot.COLORS*.

Parameters

- **square** (*bool*[28]) – When set to *True* the button is made square.
- **border** (*bool*[29]) – When set to *True* adds a border to the button.
- **visible** (*bool*[30]) – When set to *False* the button will be hidden.

[27] https://docs.python.org/3.5/library/functions.html#int
[28] https://docs.python.org/3.5/library/functions.html#bool
[29] https://docs.python.org/3.5/library/functions.html#bool
[30] https://docs.python.org/3.5/library/functions.html#bool

get_rotation()
Returns an instance of *BlueDotRotation* (page 41) if the last interaction with the button was a rotation. Returns *None* if the button was not rotated.

get_swipe()
Returns an instance of *BlueDotSwipe* (page 40) if the last interaction with the button was a swipe. Returns *None* if the button was not swiped.

is_double_press(*position*)
Returns True if the position passed represents a double press.

i.e. The last interaction was the button was to release it, and the time to press is less than the double_press_time.

Parameters **position** (BlueDotPosition (page 39)) – The BlueDotPosition where the Dot was pressed.

move(*position*)
Processes any "released" events associated with this button.

Parameters **position** (BlueDotPosition (page 39)) – The BlueDotPosition where the Dot was pressed.

press(*position*)
Processes any "pressed" events associated with this button.

Parameters **position** (BlueDotPosition (page 39)) – The BlueDotPosition where the dot was pressed.

release(*position*)
Processes any "released" events associated with this button.

Parameters **position** (BlueDotPosition (page 39)) – The BlueDotPosition where the Dot was pressed.

border
When set to *True* adds a border to the dot. Default is *False*.

color
Sets or returns the color of the dot. Defaults to BLUE.

An instance of `colors.Color` is returned.

Value can be set as a `colors.Color` object, a hex color value in the format *#rrggbb* or *#rrggbbaa*, a tuple of *(red, green, blue)* or *(red, green, blue, alpha)* values between *0* & *255* or a text description of the color, e.g. "red".

A dictionary of available colors can be obtained from *bluedot.COLORS*.

interaction
Returns an instance of *BlueDotInteraction* (page 40) representing the current or last interaction with the button.

Note: If the button is released (and inactive), *interaction* (page 38) will return the interaction when it was released, until it is pressed again. If the button has never been pressed *interaction* (page 38) will return `None`.

---

modified
Returns *True* if the button's appearance has been modified [is different] from the default.

square
When set to *True* the 'dot' is made square. Default is *False*.

visible
When set to *False* the dot will be hidden. Default is *True*.

Note: Events (press, release, moved) are still sent from the dot when it is not visible.

7.3 BlueDotPosition

class bluedot.**BlueDotPosition**(*col*, *row*, *x*, *y*)

Represents a position of where the blue dot is pressed, released or held.

Parameters

- **x** (*float*[31]) – The x position of the Blue Dot, 0 being centre, -1 being far left and 1 being far right.
- **y** (*float*[32]) – The y position of the Blue Dot, 0 being centre, -1 being at the bottom and 1 being at the top.

angle
The angle from centre of where the Blue Dot is pressed, held or released. 0 degrees is up, 0..180 degrees clockwise, -180..0 degrees anti-clockwise.

bottom
Returns `True` if the Blue Dot is pressed, held or released at the bottom.

col
The column.

distance
The distance from centre of where the Blue Dot is pressed, held or released. The radius of the Blue Dot is 1.

left
Returns `True` if the Blue Dot is pressed, held or released on the left.

middle
Returns `True` if the Blue Dot is pressed, held or released in the middle.

right
Returns `True` if the Blue Dot is pressed, held or released on the right.

row
The row.

time
The time the blue dot was at this position.

Note: This is the time the message was received from the Blue Dot app, not the time it was sent.

top
Returns `True` if the Blue Dot is pressed, held or released at the top.

x
The x position of the Blue Dot, 0 being centre, -1 being far left and 1 being far right.

y
The y position of the Blue Dot, 0 being centre, -1 being at the bottom and 1 being at the top.

[31] https://docs.python.org/3.5/library/functions.html#float
[32] https://docs.python.org/3.5/library/functions.html#float

7.4 BlueDotInteraction

class bluedot.**BlueDotInteraction**(*pressed_position*)

Represents an interaction with the Blue Dot, from when it was pressed to when it was released.

A *BlueDotInteraction* (page 40) can be active or inactive, i.e. it is active because the Blue Dot has not been released, or inactive because the Blue Dot was released and the interaction finished.

Parameters **pressed_position** (BlueDotPosition (page 39)) – The BlueDotPosition when the Blue Dot was pressed.

moved(*moved_position*)

Adds an additional position to the interaction, called when the position the Blue Dot is pressed moves.

released(*released_position*)

Called when the Blue Dot is released and completes a Blue Dot interaction

Parameters **released_position** (BlueDotPosition (page 39)) – The BlueDotPosition when the Blue Dot was released.

active

Returns True if the interaction is still active, i.e. the Blue Dot hasnt been released.

current_position

Returns the current position for the interaction.

If the interaction is inactive, it will return the position when the Blue Dot was released.

distance

Returns the total distance of the Blue Dot interaction

duration

Returns the duration in seconds of the interaction, i.e. the amount time between when the Blue Dot was pressed and now or when it was released.

positions

A sequence of *BlueDotPosition* (page 39) instances for all the positions which make up this interaction.

The first position is where the Blue Dot was pressed, the last is where the Blue Dot was released, all position in between are where the position Blue Dot changed (i.e. moved) when it was held down.

pressed_position

Returns the position when the Blue Dot was pressed i.e. where the interaction started.

previous_position

Returns the previous position for the interaction.

If the interaction contains only 1 position, None will be returned.

released_position

Returns the position when the Blue Dot was released i.e. where the interaction ended.

If the interaction is still active it returns None.

7.5 BlueDotSwipe

class bluedot.**BlueDotSwipe**(*interaction*)

Represents a Blue Dot swipe interaction.

A *BlueDotSwipe* (page 40) can be valid or invalid based on whether the Blue Dot interaction was a swipe or not.

Parameters **interaction** (BlueDotInteraction (page 40)) – The BlueDotInteraction object to be used to determine whether the interaction was a swipe.

angle
Returns the angle of the swipe (i.e. the angle between the pressed and released positions)

col
The column.

direction
Returns the direction ("up", "down", "left", "right") of the swipe. If the swipe is not valid *None* is returned.

distance
Returns the distance of the swipe (i.e. the distance between the pressed and released positions)

down
Returns `True` if the Blue Dot was swiped down.

interaction
The *BlueDotInteraction* (page 40) object relating to this swipe.

left
Returns `True` if the Blue Dot was swiped left.

right
Returns `True` if the Blue Dot was swiped right.

row
The row.

speed
Returns the speed of the swipe in Blue Dot radius / second.

up
Returns `True` if the Blue Dot was swiped up.

valid
Returns `True` if the Blue Dot interaction is a swipe.

7.6 BlueDotRotation

class bluedot.**BlueDotRotation**(*interaction*, *no_of_segments*)

anti_clockwise
Returns `True` if the Blue Dot was rotated anti-clockwise.

clockwise
Returns `True` if the Blue Dot was rotated clockwise.

col
The column.

interaction
The *BlueDotInteraction* (page 40) object relating to this rotation.

row
The row.

valid
Returns `True` if the Blue Dot was rotated.

value
Returns 0 if the Blue Dot wasn't rotated, -1 if rotated anti-clockwise and 1 if rotated clockwise.

CHAPTER 8

Bluetooth Comm API

Blue Dot also contains a useful `btcomm` API for sending and receiving data over Bluetooth.

For normal use of Blue Dot, this API doesn't need to be used, but its included in the documentation for info and for those who might need a simple Bluetooth communication library.

8.1 BluetoothServer

class bluedot.btcomm.**BluetoothServer**(*data_received_callback*, *auto_start=True*, *device='hci0'*, *port=1*, *encoding='utf-8'*, *power_up_device=False*, *when_client_connects=None*, *when_client_disconnects=None*)

Creates a Bluetooth server which will allow connections and accept incoming RFCOMM serial data.

When data is received by the server it is passed to a callback function which must be specified at initiation.

The following example will create a Bluetooth server which will wait for a connection and print any data it receives and send it back to the client:

```
from bluedot.btcomm import BluetoothServer
from signal import pause

def data_received(data):
    print(data)
    s.send(data)

s = BluetoothServer(data_received)
pause()
```

Parameters

- **data_received_callback** – A function reference should be passed, this function will be called when data is received by the server. The function should accept a single parameter which when called will hold the data received. Set to `None` if received data is not required.
- **auto_start** (*bool*[33]) – If `True` (the default), the Bluetooth server will be auto-

[33] https://docs.python.org/3.5/library/functions.html#bool

matically started on initialisation, if `False`, the method `start` will need to be called before connections will be accepted.

- **device** (*str*[34]) – The Bluetooth device the server should use, the default is "hci0", if your device only has 1 Bluetooth adapter this shouldn't need to be changed.
- **port** (*int*[35]) – The Bluetooth port the server should use, the default is 1.
- **encoding** (*str*[36]) – The encoding standard to be used when sending and receiving byte data. The default is "utf-8". If set to `None` no encoding is done and byte data types should be used.
- **power_up_device** (*bool*[37]) – If `True`, the Bluetooth device will be powered up (if required) when the server starts. The default is `False`.

  Depending on how Bluetooth has been powered down, you may need to use **rfkill** to unblock Bluetooth to give permission to bluez to power on Bluetooth:

```
sudo rfkill unblock bluetooth
```

- **when_client_connects** – A function reference which will be called when a client connects. If `None` (the default), no notification will be given when a client connects
- **when_client_disconnects** – A function reference which will be called when a client disconnects. If `None` (the default), no notification will be given when a client disconnects

disconnect_client()
Disconnects the client if connected. Returns *True* if a client was disconnected.

send(*data*)
Send data to a connected Bluetooth client

Parameters **data** (*str*[38]) – The data to be sent.

start()
Starts the Bluetooth server if its not already running. The server needs to be started before connections can be made.

stop()
Stops the Bluetooth server if its running.

adapter
A *BluetoothAdapter* (page 46) object which represents the Bluetooth device the server is using.

client_address
The MAC address[39] of the client connected to the server. Returns `None` if no client is connected.

client_connected
Returns `True` if a client is connected.

data_received_callback
Sets or returns the function which is called when data is received by the server.

The function should accept a single parameter which when called will hold the data received. Set to `None` if received data is not required.

device
The Bluetooth device the server is using. This defaults to "hci0".

[34] https://docs.python.org/3.5/library/stdtypes.html#str
[35] https://docs.python.org/3.5/library/functions.html#int
[36] https://docs.python.org/3.5/library/stdtypes.html#str
[37] https://docs.python.org/3.5/library/functions.html#bool
[38] https://docs.python.org/3.5/library/stdtypes.html#str
[39] https://en.wikipedia.org/wiki/MAC_address

encoding
The encoding standard the server is using. This defaults to "utf-8".

port
The port the server is using. This defaults to 1.

running
Returns a `True` if the server is running.

server_address
The MAC address[40] of the device the server is using.

when_client_connects
Sets or returns the function which is called when a client connects.

when_client_disconnects
Sets or returns the function which is called when a client disconnects.

8.2 BluetoothClient

class bluedot.btcomm.**BluetoothClient**(*server*, *data_received_callback*, *port=1*, *device='hci0'*, *encoding='utf-8'*, *power_up_device=False*, *auto_connect=True*)

Creates a Bluetooth client which can send data to a server using RFCOMM Serial Data.

The following example will create a Bluetooth client which will connect to a paired device called "raspberrypi", send "helloworld" and print any data is receives:

```
from bluedot.btcomm import BluetoothClient
from signal import pause

def data_received(data):
    print(data)

c = BluetoothClient("raspberrypi", data_received)
c.send("helloworld")

pause()
```

Parameters

- **server** (*str*[41]) – The server name ("raspberrypi") or server MAC address ("11:11:11:11:11:11") to connect to. The server must be a paired device.
- **data_received_callback** – A function reference should be passed, this function will be called when data is received by the client. The function should accept a single parameter which when called will hold the data received. Set to `None` if data received is not required.
- **port** (*int*[42]) – The Bluetooth port the client should use, the default is 1.
- **device** (*str*[43]) – The Bluetooth device to be used, the default is "hci0", if your device only has 1 Bluetooth adapter this shouldn't need to be changed.
- **encoding** (*str*[44]) – The encoding standard to be used when sending and receiving byte data. The default is "utf-8". If set to `None` no encoding is done and byte data types should be used.

[40] https://en.wikipedia.org/wiki/MAC_address
[41] https://docs.python.org/3.5/library/stdtypes.html#str
[42] https://docs.python.org/3.5/library/functions.html#int
[43] https://docs.python.org/3.5/library/stdtypes.html#str
[44] https://docs.python.org/3.5/library/stdtypes.html#str

- **power_up_device** (*bool*[45]) – If `True`, the Bluetooth device will be powered up (if required) when the server starts. The default is `False`.

  Depending on how Bluetooth has been powered down, you may need to use **rfkill** to unblock Bluetooth to give permission to Bluez to power on Bluetooth:

  ```
  sudo rfkill unblock bluetooth
  ```

- **auto_connect** (*bool*[46]) – If `True` (the default), the Bluetooth client will automatically try to connect to the server at initialisation, if `False`, the *connect()* (page 46) method will need to be called.

connect()
Connect to a Bluetooth server.

disconnect()
Disconnect from a Bluetooth server.

send(*data*)
Send data to a Bluetooth server.

Parameters **data** (*str*[47]) – The data to be sent.

adapter
A *BluetoothAdapter* (page 46) object which represents the Bluetooth device the client is using.

client_address
The MAC address of the device being used.

connected
Returns `True` when connected.

data_received_callback
Sets or returns the function which is called when data is received by the client.

The function should accept a single parameter which when called will hold the data received. Set to `None` if data received is not required.

device
The Bluetooth device the client is using. This defaults to "hci0".

encoding
The encoding standard the client is using. The default is "utf-8".

port
The port the client is using. This defaults to 1.

server
The server name ("raspberrypi") or server MAC address[48] ("11:11:11:11:11:11") to connect to.

8.3 BluetoothAdapter

class bluedot.btcomm.**BluetoothAdapter**(*device='hci0'*)
Represents and allows interaction with a Bluetooth Adapter.

The following example will get the Bluetooth adapter, print its powered status and any paired devices:

```
a = BluetoothAdapter()
print("Powered = {}".format(a.powered))
print(a.paired_devices)
```

[45] https://docs.python.org/3.5/library/functions.html#bool
[46] https://docs.python.org/3.5/library/functions.html#bool
[47] https://docs.python.org/3.5/library/stdtypes.html#str
[48] https://en.wikipedia.org/wiki/MAC_address

Parameters **device** (*str*[49]) – The Bluetooth device to be used, the default is "hci0", if your device only has 1 Bluetooth adapter this shouldn't need to be changed.

allow_pairing(*timeout=60*)
Put the adapter into discoverable and pairable mode.

Parameters **timeout** (*int*[50]) – The time in seconds the adapter will remain pairable. If set to `None` the device will be discoverable and pairable indefinetly.

address
The MAC address[51] of the Bluetooth adapter.

device
The Bluetooth device name. This defaults to "hci0".

discoverable
Set to `True` to make the Bluetooth adapter discoverable.

pairable
Set to `True` to make the Bluetooth adapter pairable.

paired_devices
Returns a sequence of devices paired with this adapater `[(mac_address, name), (mac_address, name), ...]`:

```
a = BluetoothAdapter()
devices = a.paired_devices
for d in devices:
    device_address = d[0]
    device_name = d[1]
```

powered
Set to `True` to power on the Bluetooth adapter.

Depending on how Bluetooth has been powered down, you may need to use **rfkill** to unblock Bluetooth to give permission to bluez to power on Bluetooth:

```
sudo rfkill unblock bluetooth
```

[49] https://docs.python.org/3.5/library/stdtypes.html#str
[50] https://docs.python.org/3.5/library/functions.html#int
[51] https://en.wikipedia.org/wiki/MAC_address

CHAPTER 9

Mock API

Blue Dot also contains a useful `mock` API for simulating Blue Dot and bluetooth comms. This is useful for testing and allows for prototyping without having to use a Blue Dot client.

9.1 MockBlueDot

class bluedot.mock.**MockBlueDot**(*device='hci0'*, *port=1*, *auto_start_server=True*, *power_up_device=False*, *print_messages=True*, *cols=1*, *rows=1*)

MockBlueDot (page 49) inherits from `BlueDot` but overrides `_create_server()`, to create a *MockBluetoothServer* (page 50) which can be used for testing and debugging.

launch_mock_app()

Launches a mock Blue Dot app.

The mock app reacts to mouse clicks and movement and calls the mock blue dot methods to simulates presses.

This is useful for testing, allowing you to interact with Blue Dot without having to script mock functions.

The mock app uses pygame which will need to be installed.

mock_blue_dot_moved(*col*, *row*, *x*, *y*)

Simulates the Blue Dot being moved.

Parameters

- **col** (*int*[52]) – The column position of the button
- **row** (*int*[53]) – The row position of the button
- **x** (*int*[54]) – The x position where the button was moved too
- **y** (*int*[55]) – The y position where the button was moved too

mock_blue_dot_pressed(*col*, *row*, *x*, *y*)

Simulates the Blue Dot being pressed.

[52] https://docs.python.org/3.5/library/functions.html#int
[53] https://docs.python.org/3.5/library/functions.html#int
[54] https://docs.python.org/3.5/library/functions.html#int
[55] https://docs.python.org/3.5/library/functions.html#int

Parameters

- **col** (*int*[56]) – The column position of the button
- **row** (*int*[57]) – The row position of the button
- **x** (*int*[58]) – The x position where the button was pressed
- **y** (*int*[59]) – The y position where the button was pressed

mock_blue_dot_released(*col*, *row*, *x*, *y*)
Simulates the Blue Dot being released.

Parameters

- **col** (*int*[60]) – The column position of the button
- **row** (*int*[61]) – The row position of the button
- **x** (*int*[62]) – The x position where the button was released
- **y** (*int*[63]) – The y position where the button was released

mock_client_connected()
Simulates a client connecting to the Blue Dot.

Parameters **client_address** (*string*) – The mock client mac address, defaults to '11:11:11:11:11:11'

mock_client_disconnected()
Simulates a client disconnecting from the Blue Dot.

9.2 MockBluetoothServer

class bluedot.mock.**MockBluetoothServer**(*data_received_callback*, *auto_start=True*, *device='mock0'*, *port=1*, *encoding='utf-8'*, *power_up_device=False*, *when_client_connects=None*, *when_client_disconnects=None*)
MockBluetoothServer (page 50) inherits from *BluetoothServer* (page 43) but overrides __init__, *start()* (page 50) , *stop()* (page 50) and send_raw() to create a *MockBluetoothServer* (page 50) which can be used for testing and debugging.

mock_client_connected(*mock_client=None*)
Simulates a client connected to the *BluetoothServer* (page 43).

Parameters **mock_client** (MockBluetoothClient (page 51)) – The mock client to interact with, defaults to *None*. If *None*, client address is set to '99:99:99:99:99:99'

mock_client_disconnected()
Simulates a client disconnecting from the *BluetoothServer* (page 43).

mock_client_sending_data(*data*)
Simulates a client sending data to the *BluetoothServer* (page 43).

start()
Starts the Bluetooth server if its not already running. The server needs to be started before connections can be made.

[56] https://docs.python.org/3.5/library/functions.html#int
[57] https://docs.python.org/3.5/library/functions.html#int
[58] https://docs.python.org/3.5/library/functions.html#int
[59] https://docs.python.org/3.5/library/functions.html#int
[60] https://docs.python.org/3.5/library/functions.html#int
[61] https://docs.python.org/3.5/library/functions.html#int
[62] https://docs.python.org/3.5/library/functions.html#int
[63] https://docs.python.org/3.5/library/functions.html#int

stop()
Stops the Bluetooth server if its running.

9.3 MockBluetoothClient

class bluedot.mock.**MockBluetoothClient**(*server*, *data_received_callback*, *port=1*, *device='mock1'*, *encoding='utf-8'*, *power_up_device=False*, *auto_connect=True*)
MockBluetoothClient (page 51) inherits from *BluetoothClient* (page 45) but overrides __init__, *connect()* (page 51) and send_raw() to create a *MockBluetoothServer* (page 50) which can be used for testing and debugging.

Note - the *server* parameter should be an instance of *MockBluetoothServer* (page 50).

connect()
Connect to a Bluetooth server.

disconnect()
Disconnect from a Bluetooth server.

mock_server_sending_data(*data*)
Simulates a server sending data to the *BluetoothClient* (page 45).

CHAPTER 10

Protocol

Blue Dot uses a client/server model. The `BlueDot` class starts a Bluetooth server, the Blue Dot application connects as a client.

The detail below can be used to create new applications (clients); if you do please send a pull request :)

10.1 Bluetooth

Communication over Bluetooth is made using a RFCOMM serial port profile using UUID "00001101-0000-1000-8000-00805f9b34fb".

10.2 Specification

The transmission of data from client to server or server to client is a simple stream no acknowledgements or data is sent in response to commands.

All messages between conform to the same format:

```
[operation],[params],[*]\n
```

Messages are sent as utf-8 encoded strings.

\n represents the new-line character.

The following operations are used to communicate between client and server.

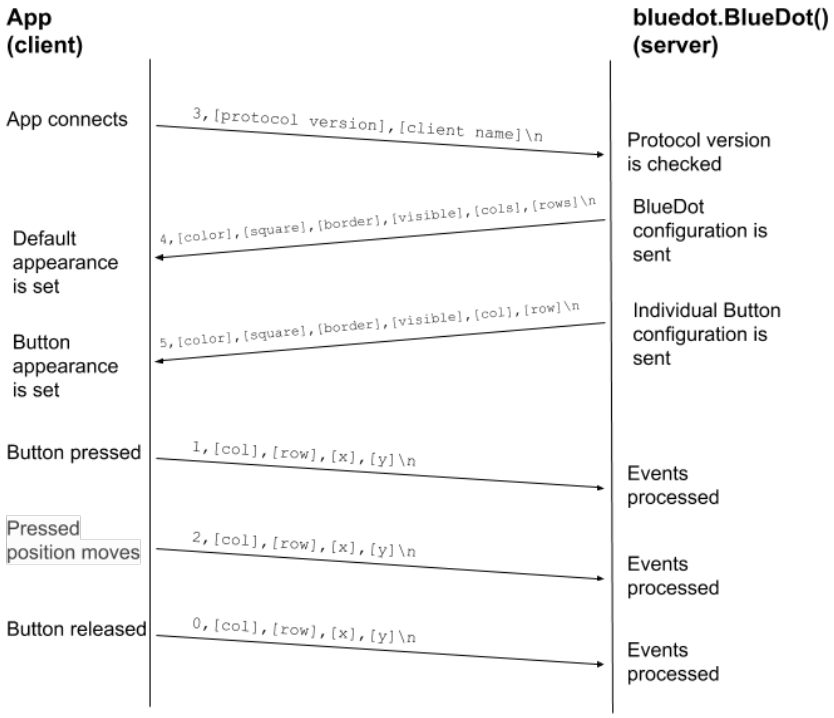
Operations	Message format	Direction
Button released	`0,[col],[row],[x],[y]\n`	Client > Server
Button pressed	`1,[col],[row],[x],[y]\n`	Client > Server
Button moved	`2,[col],[row],[x],[y]\n`	Client > Server
Protocol check	`3,[protocol version],[client name]\n`	Client > Server
Set config	`4,[color],[square],[border],[visible],[cols],[rows]\n`	Server > Client
Set button config	`5,[color],[square],[border],[visible],[col],[row]\n`	Server > Client

Messages are constructed using the following parameters.

Parameter	Description
cols	The number of columns in the matrix of buttons
rows	The number of rows in the matrix of buttons
col	The column position of the button (0 is top)
row	The row position of the button (0 is left)
x	Horizontal position between -1 and +1, with 0 being the centre and +1 being the right radius of the button.
y	Vertical position between -1 and +1, with 0 being the centre and +1 being the top radius of the button.
protocol version	The version of protocol the client supports.
client name	The name of the client e.g. “Android Blue Dot App”
color	A hex value in the format `#rrggbbaa` representing red, green, blue, alpha values.
square	0 or 1, 1 if the dot should be a square.
border	0 or 1, 1 if the dot should have a border.
visible	0 or 1, 1 if the dot should be visible.

Messages are sent when:

1. A client connects
2. When the setup (or appearance) of a button changes
3. A button is released, pressed or moved



10.3 Example

When the Android client connects using protocol version 2:

```
3,2,Android Blue Dot app\n
```

The setup of the Blue Dot is sent to the client:

```
4,#0000ffff,0,0,1,1,2\n
```

If any buttons are different to the default, the configuration is sent:

```
5,#00ff0000,0,0,1,0,1\n
```

If the "first" button at position [0,0] is pressed at the top, the following message will be sent:

```
1,0,0,0.0,1.0\n
```

While the button is pressed (held down), the user moves their finger to the far right causing the following message to be sent:

```
2,0,0,1.0,0.0\n
```

The button is then released, resulting in the following message:

```
0,0,0,1.0,0.0\n
```

The color of the button is changed to "red" the server sends to the client:

```
5,#ff0000ff,0,0,1,0,0\n
```

10.4 Versions

- 0 - initial version
- 1 - introduction of operation 3, 4
- 2 - Blue Dot version 2, introduction of col, row for multiple buttons and operation 5

CHAPTER 11

Build

These are instructions for how to develop, build and deploy Blue Dot.

11.1 Develop

Install / upgrade tools:

```
sudo python3 -m pip install --upgrade pip setuptools wheel twine virtualenv
```

Create a virtual environment (recommended):

```
virtualenv --system-site-packages bluedot-dev
cd bluedot-dev
source bin/activate
```

Clone repo and install for dev:

```
git clone https://github.com/martinohanlon/BlueDot
cd BlueDot
git checkout dev
python3 setup.py develop
```

11.2 Test

Install pytest[64]:

```
pip3 install -U pytest
```

Run tests:

```
cd BlueDot/tests
pytest -v
```

[64] https://doc.pytest.org/

11.3 Deploy

Build for deployment:

```
python3 setup.py sdist
python3 setup.py bdist_wheel
```

Deploy to PyPI[65]:

```
twine upload dist/* --skip-existing
```

[65] https://pypi.python.org/pypi

CHAPTER 12

Change log

12.1 Bluedot Python library

12.1.1 2.0.0 - 2020-11-01

- implementation of multiple buttons in a matrix
- refactor of significant portions of the code base
- improvement to btcomm to manage large messages
- update to MockBlueDot
- deprecated BlueDot.interaction
- added warnings when invalid data is received
- support for protocol version 2
- removed support for Python 2, 3.3 & 3.4

12.1.2 1.3.2 - 2019-04-22

- change to how callbacks are called
- added *set_when_pressed*, *set_when_released*, etc to allow callbacks to be called in their own threads.

12.1.3 1.3.1 - 2019-01-01

- minor bug fix to launch_mock_app

12.1.4 1.3.0 - 2018-12-30

- added ability to change the color, border, shape and visibility of the dot (`color` (page 35), `border` (page 35), `square` (page 36), `visible` (page 36))
- added protocol version checking
- minor threading changes in btcomm

- updates to the Blue Dot Python app
- rewrite of the mock app
- support for protocol version 1

12.1.5 1.2.3 - 2018-02-22

- fix to *wait_for_press* and *wait_for_release*
- when_client_connects and when_client_disconnects callbacks are now threaded
- The python blue dot app can now be started with the command *bluedotapp*
- new tests for *wait_for_(events)*

12.1.6 1.2.2 - 2017-12-30

- bluetooth comms tests and minor bug fix in `BluetoothClient` (page 45)

12.1.7 1.2.1 - 2017-12-18

- massive code and docs tidy up by Dave Jones[66]

12.1.8 1.2.0 - 2017-12-10

- added when_rotated
- threaded swipe callbacks
- exposed new `BlueDot` (page 33) properties (`adapter` (page 35), `running` (page 36), `paired_devices` (page 36))
- fixed active bug in interaction
- automated tests

12.1.9 1.1.0 - 2017-11-05

- threaded callbacks
- python app rounded x,y performance improvements

12.1.10 1.0.4 - 2017-09-10

- serial port profile port fix
- launching multiple blue dots fix

12.1.11 1.0.3 - 2017-07-28

- python 2 bug fix

[66] https://github.com/waveform80

12.1.12 1.0.2 - 2017-07-23

- bug fix

12.1.13 1.0.1 - 2017-06-19

- bug fixes

12.1.14 1.0.0 - 2017-06-04

- production release!
- added double click
- doc updates
- minor changes

12.1.15 0.4.0 - 2017-05-05

- added swipes and interactions
- doc updates
- bug fix in `BlueDot.when_moved`

12.1.16 0.3.0 - 2017-05-01

- Python Blue Dot app
- minor bug fix in *BluetoothClient* (page 45)

12.1.17 0.2.1 - 2017-04-23

- bug fix in `MockBlueDot`
- doc fixes

12.1.18 0.2.0 - 2017-04-23

- added *when_client_connects* (page 37), *when_client_disconnects* (page 37)
- added *allow_pairing()* (page 34) functions
- refactored Bluetooth comms
- added *BluetoothAdapter* (page 46)

12.1.19 0.1.2 - 2017-04-14

- mock blue dot improvements
- doc fixes

12.1.20 0.1.1 - 2017-04-08

- clamped distance in *BlueDotPosition* (page 39)

12.1.21 0.1.0 - 2017-04-07

- Check Bluetooth adapter is powered
- Handle client connection timeouts
- Docs & image updates

12.1.22 0.0.6 - 2017-04-05

- Added `MockBlueDot` for testing and debugging
- more docs

12.1.23 0.0.4 - 2017-03-31

Updates after alpha feedback

- Python 2 compatibility
- `.dot_position` to `.position`
- `.values` added
- clamped `x`, `y` to 1
- loads of doc updates

12.1.24 0.0.2 - 2017-03-29

Alpha - initial testing

12.2 Android app

12.2.1 10 (2.2.1) - 2022-01-03

- Android 12+ fixes

12.2.2 9 (2.2) - 2022-12-23

- Android SDK and API version uplift (due to google play store minimum requirements change)

12.2.3 8 (2.1) - 2020-12-28

- removed “auto port discovery” after the introduction of pulseaudio to Raspberry Pi OS broke it
- introduced the “default port” setting as an alternative

12.2.4 7 (2.0) - 2020-11-01

- implementation of multiple buttons in a matrix
- support for protocol version 2

12.2.5 6 (1.3.1) - 2019-12-30

- Minor bug fix

12.2.6 5 (1.3) - 2019-12-29

- Added settings menu so a specific bluetooth port can be selected
- Using specific bluetooth ports, multiple apps can now connect to a single BT devices
- Minor bugs fixes

12.2.7 4 (1.2) - 2018-12-30

- Rewrite of the Button view
- Rewrite of the Bluetooth comms layer
- Support for colours, square and border
- Landscape (and portrait) views
- added protocol version checking
- support for protocol version 1

12.2.8 3 (1.1.1) - 2018-09-21

- Android SDK version uplift (due to google play store minimum requirements change)

12.2.9 2 (1.1) - 2017-11-05

- better responsive layout
- fixed issues with small screen devices
- rounded x,y values increasing performance
- new help icon
- link to https://bluedot.readthedocs.io not http

12.2.10 1 (0.0.2) - 2017-04-05

- icon transparency
- connection monitor
- added info icon to https://bluedot.readthedocs.io

12.2.11 0 (0.0.1) - 2017-03-29

- alpha - initial testing

Python Module Index

b

bluedot, 33
bluedot.btcomm, 43
bluedot.mock, 49

Index

A

active (*bluedot.BlueDotInteraction attribute*), 40
adapter (*bluedot.BlueDot attribute*), 35
adapter (*bluedot.btcomm.BluetoothClient attribute*), 46
adapter (*bluedot.btcomm.BluetoothServer attribute*), 44
address (*bluedot.btcomm.BluetoothAdapter attribute*), 47
allow_pairing() (*bluedot.BlueDot method*), 34
allow_pairing() (*bluedot.btcomm.BluetoothAdapter method*), 47
angle (*bluedot.BlueDotPosition attribute*), 39
angle (*bluedot.BlueDotSwipe attribute*), 40
anti_clockwise (*bluedot.BlueDotRotation attribute*), 41

B

BlueDot (*class in bluedot*), 33
bluedot (*module*), 33
bluedot.btcomm (*module*), 43
bluedot.mock (*module*), 49
BlueDotButton (*class in bluedot*), 37
BlueDotInteraction (*class in bluedot*), 40
BlueDotPosition (*class in bluedot*), 39
BlueDotRotation (*class in bluedot*), 41
BlueDotSwipe (*class in bluedot*), 40
BluetoothAdapter (*class in bluedot.btcomm*), 46
BluetoothClient (*class in bluedot.btcomm*), 45
BluetoothServer (*class in bluedot.btcomm*), 43
border (*bluedot.BlueDot attribute*), 35
border (*bluedot.BlueDotButton attribute*), 38
bottom (*bluedot.BlueDotPosition attribute*), 39
buttons (*bluedot.BlueDot attribute*), 35

C

client_address (*bluedot.btcomm.BluetoothClient attribute*), 46
client_address (*bluedot.btcomm.BluetoothServer attribute*), 44
client_connected (*bluedot.btcomm.BluetoothServer attribute*), 44
clockwise (*bluedot.BlueDotRotation attribute*), 41
col (*bluedot.BlueDotPosition attribute*), 39
col (*bluedot.BlueDotRotation attribute*), 41
col (*bluedot.BlueDotSwipe attribute*), 41
color (*bluedot.BlueDot attribute*), 35
color (*bluedot.BlueDotButton attribute*), 38
cols (*bluedot.BlueDot attribute*), 35
connect() (*bluedot.btcomm.BluetoothClient method*), 46
connect() (*bluedot.mock.MockBluetoothClient method*), 51
connected (*bluedot.btcomm.BluetoothClient attribute*), 46
current_position (*bluedot.BlueDotInteraction attribute*), 40

D

data_received_callback (*bluedot.btcomm.BluetoothClient attribute*), 46
data_received_callback (*bluedot.btcomm.BluetoothServer attribute*), 44
device (*bluedot.BlueDot attribute*), 35
device (*bluedot.btcomm.BluetoothAdapter attribute*), 47
device (*bluedot.btcomm.BluetoothClient attribute*), 46
device (*bluedot.btcomm.BluetoothServer attribute*), 44
direction (*bluedot.BlueDotSwipe attribute*), 41
disconnect() (*bluedot.btcomm.BluetoothClient method*), 46
disconnect() (*bluedot.mock.MockBluetoothClient method*), 51
disconnect_client() (*bluedot.btcomm.BluetoothServer method*), 44
discoverable (*bluedot.btcomm.BluetoothAdapter attribute*), 47
distance (*bluedot.BlueDotInteraction attribute*), 40
distance (*bluedot.BlueDotPosition attribute*), 39
distance (*bluedot.BlueDotSwipe attribute*), 41

double_press_time (*bluedot.BlueDot attribute*), 35
down (*bluedot.BlueDotSwipe attribute*), 41
duration (*bluedot.BlueDotInteraction attribute*), 40

E

encoding (*bluedot.btcomm.BluetoothClient attribute*), 46
encoding (*bluedot.btcomm.BluetoothServer attribute*), 44

G

get_rotation() (*bluedot.BlueDotButton method*), 37
get_swipe() (*bluedot.BlueDotButton method*), 38

I

interaction (*bluedot.BlueDot attribute*), 35
interaction (*bluedot.BlueDotButton attribute*), 38
interaction (*bluedot.BlueDotRotation attribute*), 41
interaction (*bluedot.BlueDotSwipe attribute*), 41
is_connected (*bluedot.BlueDot attribute*), 36
is_double_press() (*bluedot.BlueDotButton method*), 38
is_pressed (*bluedot.BlueDot attribute*), 36

L

launch_mock_app() (*bluedot.mock.MockBlueDot method*), 49
left (*bluedot.BlueDotPosition attribute*), 39
left (*bluedot.BlueDotSwipe attribute*), 41

M

middle (*bluedot.BlueDotPosition attribute*), 39
mock_blue_dot_moved() (*bluedot.mock.MockBlueDot method*), 49
mock_blue_dot_pressed() (*bluedot.mock.MockBlueDot method*), 49
mock_blue_dot_released() (*bluedot.mock.MockBlueDot method*), 50
mock_client_connected() (*bluedot.mock.MockBlueDot method*), 50
mock_client_connected() (*bluedot.mock.MockBluetoothServer method*), 50
mock_client_disconnected() (*bluedot.mock.MockBlueDot method*), 50
mock_client_disconnected() (*bluedot.mock.MockBluetoothServer method*), 50
mock_client_sending_data() (*bluedot.mock.MockBluetoothServer method*), 50
mock_server_sending_data() (*bluedot.mock.MockBluetoothClient method*), 51
MockBlueDot (*class in bluedot.mock*), 49
MockBluetoothClient (*class in bluedot.mock*), 51
MockBluetoothServer (*class in bluedot.mock*), 50
modified (*bluedot.BlueDotButton attribute*), 38
move() (*bluedot.BlueDotButton method*), 38
moved() (*bluedot.BlueDotInteraction method*), 40

P

pairable (*bluedot.btcomm.BluetoothAdapter attribute*), 47
paired_devices (*bluedot.BlueDot attribute*), 36
paired_devices (*bluedot.btcomm.BluetoothAdapter attribute*), 47
port (*bluedot.BlueDot attribute*), 36
port (*bluedot.btcomm.BluetoothClient attribute*), 46
port (*bluedot.btcomm.BluetoothServer attribute*), 45
positions (*bluedot.BlueDotInteraction attribute*), 40
powered (*bluedot.btcomm.BluetoothAdapter attribute*), 47
press() (*bluedot.BlueDotButton method*), 38
pressed_position (*bluedot.BlueDotInteraction attribute*), 40
previous_position (*bluedot.BlueDotInteraction attribute*), 40
print_messages (*bluedot.BlueDot attribute*), 36

R

release() (*bluedot.BlueDotButton method*), 38
released() (*bluedot.BlueDotInteraction method*), 40
released_position (*bluedot.BlueDotInteraction attribute*), 40
resize() (*bluedot.BlueDot method*), 34
right (*bluedot.BlueDotPosition attribute*), 39
right (*bluedot.BlueDotSwipe attribute*), 41
rotation_segments (*bluedot.BlueDot attribute*), 36
row (*bluedot.BlueDotPosition attribute*), 39
row (*bluedot.BlueDotRotation attribute*), 41
row (*bluedot.BlueDotSwipe attribute*), 41
rows (*bluedot.BlueDot attribute*), 36
running (*bluedot.BlueDot attribute*), 36
running (*bluedot.btcomm.BluetoothServer attribute*), 45

S

send() (*bluedot.btcomm.BluetoothClient method*), 46
send() (*bluedot.btcomm.BluetoothServer method*), 44
server (*bluedot.BlueDot attribute*), 36
server (*bluedot.btcomm.BluetoothClient attribute*), 46
server_address (*bluedot.btcomm.BluetoothServer attribute*), 45

set_when_client_connects() (*bluedot.BlueDot method*), 34
set_when_client_disconnects() (*bluedot.BlueDot method*), 34
speed (*bluedot.BlueDotSwipe attribute*), 41
square (*bluedot.BlueDot attribute*), 36
square (*bluedot.BlueDotButton attribute*), 38
start() (*bluedot.BlueDot method*), 35
start() (*bluedot.btcomm.BluetoothServer method*), 44
start() (*bluedot.mock.MockBluetoothServer method*), 50
stop() (*bluedot.BlueDot method*), 35
stop() (*bluedot.btcomm.BluetoothServer method*), 44
stop() (*bluedot.mock.MockBluetoothServer method*), 50

T

time (*bluedot.BlueDotPosition attribute*), 39
top (*bluedot.BlueDotPosition attribute*), 39

U

up (*bluedot.BlueDotSwipe attribute*), 41

V

valid (*bluedot.BlueDotRotation attribute*), 41
valid (*bluedot.BlueDotSwipe attribute*), 41
value (*bluedot.BlueDotRotation attribute*), 41
visible (*bluedot.BlueDot attribute*), 36
visible (*bluedot.BlueDotButton attribute*), 38

W

wait_for_connection() (*bluedot.BlueDot method*), 35
when_client_connects (*bluedot.BlueDot attribute*), 37
when_client_connects (*bluedot.btcomm.BluetoothServer attribute*), 45
when_client_disconnects (*bluedot.BlueDot attribute*), 37
when_client_disconnects (*bluedot.btcomm.BluetoothServer attribute*), 45

X

x (*bluedot.BlueDotPosition attribute*), 39

Y

y (*bluedot.BlueDotPosition attribute*), 39