## Contents

1 Getting Started  
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  
2.1 Using the Desktop .............................................. 3  
2.2 Using the Command Line ....................................... 3  

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

4 Recipes  
4.1 Button ............................................................ 7  
4.2 Joystick ............................................................ 9  
4.3 Slider ............................................................ 12  
4.4 Swiping ........................................................... 13  
4.5 Rotating .......................................................... 14  
4.6 Appearance ....................................................... 15  
4.7 Bluetooth ......................................................... 16  
4.8 Testing ............................................................ 16  

5 Blue Dot Android App  
5.1 Start .............................................................. 19  

6 Blue Dot Python App  
6.1 Start .............................................................. 21  
6.2 Options .......................................................... 22  

7 Blue Dot API  
7.1 BlueDot ........................................................... 25  
7.2 BlueDotPosition .................................................. 31  
7.3 BlueDotInteraction .............................................. 32  
7.4 BlueDotSwipe .................................................... 32  
7.5 BlueDotRotation ................................................ 33  

8 Bluetooth Comm API  
8.1 BluetoothServer ................................................ 35  
8.2 BluetoothClient ................................................ 37
In order to use Blue Dot you will need:

- A Raspberry Pi
  - with built-in Bluetooth (such as the Pi 3 or Pi 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 app² can be installed from the Google Play Store.

#### 1.1.2 Python Library

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

```bash
sudo apt install python3-dbus
sudo pip3 install bluedot
```

If you want to use Blue Dot with Python 2 (there really is no need though!):

```bash
sudo apt install python-dbus
sudo pip install bluedot
```

To upgrade to the latest version:

**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 (select *Menu → Programming → Python 3*)
2. Select *File → New File* to create a new program
3. Enter the following code:

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

4. Save your program (select *File → Save As*) and save as `mydot.py`
5. Run the program, (select *Run → Run Module* or press F5)

   **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 app³ on your Android phone or run the *Blue Dot Python App* (page 21) on your 2nd Raspberry Pi:

1. Select your Raspberry Pi from the list
2. Press the Blue Dot

**1.5 Where next**

Check out the *Recipes* (page 7) and the *Blue Dot API* (page 25) documentation for more ideas on using Blue Dot.

---


## Pair a Raspberry Pi and Android phone

### 2.1 Using the Desktop

On your Android phone:

1. Open Settings
2. Select Bluetooth
3. This will 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
5. Enter a PIN code

On your Android phone again:

1. Enter the same PIN code when prompted
2. Touch “OK”

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

### 2.2 Using the Command Line

On your Raspberry Pi:

1. Type `bluetoothctl` and press Enter to open Bluetooth control
2. At the `{bluetooth}`# prompt enter the following commands:
On your Android phone:
1. Open Settings
2. Select Bluetooth
3. Your Raspberry Pi will appear in the list; select it
4. Enter a PIN

On your Raspberry Pi:
1. Re-enter the PIN
2. Type `quit` and press Enter to return to the command line
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:
On the second Raspberry Pi:

1. Enter \texttt{bluetoothctl} to open Bluetooth control

2. At the \texttt{[bluetooth]#} prompt enter the following commands:

\begin{verbatim}
discoverable on
pairable on
agent on
default-agent
scan on
\end{verbatim}

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

\begin{verbatim}
\end{verbatim}

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

\begin{verbatim}
pair 12:23:34:45:56:67
\end{verbatim}

5. Enter a PIN

On the first Raspberry Pi again:

1. Enter the same PIN when prompted

2. Type \texttt{quit} and press Enter to return to the command line
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 25) 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 object then waiting for the Blue Dot app to connect and be pressed:

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

Alternatively you can also use `when_pressed` (page 30) to call a function:

```python
from blue import BlueDot
from signal import pause

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

bd = BlueDot()
bd.when_pressed = say_hello
pause()
```

`wait_for_release` (page 28) and `when_released` (page 30) also allow you to interact when the Blue Dot is released:
from bluedot import BlueDot
from signal import pause

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` (page 27) and `when_double_pressed` (page 29):

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` you can interact with electronic components, such as LEDs, connected to your Raspberry Pi.

When the Blue Dot 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` (page 30) and `whenReleased` (page 30):

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

bd = BlueDot()
led = LED(27)

(continues on next page)

4 https://gpiozero.readthedocs.io/en/latest/recipes.html#module-gpiozero
bd.when_pressed = led.on
bd.when_released = led.off
pause()

Alternatively use `source`\(^5\) and `values` (page 29):

```python
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`\(^6\) and `BlueDot` (page 25), you can really easily create a remote camera:

```python
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\(^7\) on a joystick:

```python
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:
    print("fire")

bd = BlueDot()
bd.when_pressed = dpad
pause()

At the moment the D-pad 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`, you can create a bluetooth controlled robot which moves when the dot is pressed and stops when it is released:

```python
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 31) 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:

```python
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)¹¹:

```python
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()

(continues on next page)

¹¹ https://github.com/bennuttall

4.2. Joystick

11
4.3 Slider

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

4.3.1 Centre Out

Using the `BlueDotPosition.distance` (page 31) property which is returned when the position is moved you can create a slider which goes from the centre out in any direction:

```python
from bluedot import BlueDot
from signal import pause

def show_percentage(pos):
    percentage = round(pos.distance * 100, 2)
    print("{:.1f}%".format(percentage))

bd = BlueDot()
bd.when_moved = show_percentage
pause()
```

4.3.2 Left to Right

The `BlueDotPosition.x` (page 31) 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:

```python
from bluedot import BlueDot
from signal import pause

def show_percentage(pos):
    horizontal = ((pos.x + 1) / 2)
    percentage = round(horizontal * 100, 2)
    print("{:.1f}%".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 31) instead.

4.3.3 Dimmer Switch

Using the `gpiozero.PWMLED`\(^{12}\) class and `BlueDot` (page 25) as a vertical slider you can create a wireless dimmer switch:

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

\(^{12}\) https://gpiozero.readthedocs.io/en/latest/api_output.html#gpiozero.PWMLED
```python
def set_brightness(pos):
    brightness = (pos.y + 1) / 2
    led.value = brightness

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

## 4.4 Swiping

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

### 4.4.1 Single

Detecting a single swipe is easy using `wait_for_swipe` (page 28):

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

Alternatively you can also use `when_swiped` (page 30) to call a function:

```python
from bluedot import BlueDot
from signal import pause

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

bd = BlueDot()
bd.when_swiped = swiped
pause()
```

### 4.4.2 Direction

You can tell what direction the Blue Dot is swiped by using the `BlueDotSwipe` (page 32) object passed to the function assigned to `when_swiped` (page 30):

```python
from bluedot import BlueDot
from signal import pause

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

(continues on next page)
bd = BlueDot()
bd.when_swiped = swiped
pause()

4.4.3 Speed, Angle, and Distance

BlueDotSwipe (page 32) 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:

```python
from bluedot import BlueDot
from signal import pause

def swiped(swipe):
    print("Swiped")
    print("speed=\n".format(swipe.speed))
    print("angle=\n".format(swipe.angle))
    print("distance=\n".format(swipe.distance))

bd = BlueDot()
bd.when_swiped = swiped
pause()
```

4.5 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.

4.5.1 Counter

Using the when_rotated (page 30) callback you can create a counter which increments / decrements when the Blue Dot is rotated either clockwise or anti-clockwise. A BlueDotRotation (page 33) object is passed to the callback. Its value (page 33) property will be -1 if rotated anti-clockwise and 1 if rotated clockwise:

```python
from bluedot import BlueDot
from signal import pause

count = 0

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

    print("{} {} {}\n".format(count,
                                rotation.clockwise,
                                rotation.anti_clockwise))

bd = BlueDot()
bd.when_rotated = rotated
pause()
```

The rotation speed can be modified using the BlueDot.rotation_segments (page 29) property which changes the number of segments the Blue Dot is split into:
4.6 Appearance

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

4.6.1 Colo(u)r

To change the color of the dot use the `color` (page 28): property:

```python
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:

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

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

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

4.6.2 Square

The dot can also be made square using the `square` (page 29): property:

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

4.6.3 Border

A border can also been added to the dot (or the square) by setting the `border` (page 28): property to `True`:

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

4.6.4 (In)visible

The dot can be hidden and shown using the `visible` (page 29): property:

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

You can interact with the Bluetooth adapter using `BlueDot` (page 25).

4.7.1 Pairing

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

```python
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):

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

bd = BlueDot()
button = Button(27)
button.when_pressed = bd.allow_pairing
pause()
```

4.7.2 Paired Devices

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

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

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

4.8 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 25) 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.
4.8.1 Mock App

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

```python
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()
```

4.8.2 Scripted Tests

Tests can also be scripted using `MockBlueDot`:

```python
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)
```
The Blue Dot app\textsuperscript{13} is available to download from the Google Play store. Please leave a rating and review if you find Blue Dot useful :) 

\begin{figure}
\centering
\includegraphics[width=\textwidth]{blue_dot_app.png}
\caption{The Blue Dot Android app}
\end{figure}

### 5.1 Start

1. Download the Blue Dot app\textsuperscript{14} from the Google Play store.
2. If you haven't already done so, pair your raspberry pi as described in the \textit{Getting Started} (page 1) guide.
3. Run the Blue Dot app

\textsuperscript{13} http://play.google.com/store/apps/details?id=com.stuffaboutcode.bluedot
\textsuperscript{14} 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
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 haven’t 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)

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

This class starts an instance of `btcomm.BluetoothServer` (page 35) which manages the connection with the Blue Dot client.

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

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

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

Parameters

- **device** *(str)* – 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)* – The Bluetooth port the server should use, the default is 1, and under normal use this should never need to change.

- **auto_start_server** *(bool)* – If True (the default), the Bluetooth server will be automatically started on initialisation; if False, the method `start()` (page 27) will need to be called before connections will be accepted.

- **power_up_device** *(bool)* – 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:

---

15 https://docs.python.org/3.5/library/stdtypes.html#str
16 https://docs.python.org/3.5/library/functions.html#int
17 https://docs.python.org/3.5/library/functions.html#bool
18 https://docs.python.org/3.5/library/functions.html#bool
`sudo rfkill unblock bluetooth`

- **`print_messages`** (bool) – If True (the default), server status messages will be printed stating when the server has started and when clients connect / disconnect.

**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) – The time in seconds the adapter will remain pairable. If set to None the device will be discoverable and pairable indefinitely.

**set_when_client_connects** (callback, background=False)
Sets the function which is called when a Blue Dot connects.

**Parameters**

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

**set_when_double_pressed** (callback, background=False)
Sets the function which is called when the Blue Dot is double pressed.

**Parameters**

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

**set_when_moved** (callback, background=False)
Sets the function which is called when the position the Blue Dot is pressed is moved.

**Parameters**

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

**set_when_pressed** (callback, background=False)
Sets the function which is called when the Blue Dot is pressed.

**Parameters**

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

---

19 https://docs.python.org/3.5/library/functions.html#bool
20 https://docs.python.org/3.5/library/functions.html#int
21 https://docs.python.org/3.5/library/functions.html#bool
22 https://docs.python.org/3.5/library/functions.html#bool
23 https://docs.python.org/3.5/library/functions.html#bool
24 https://docs.python.org/3.5/library/functions.html#bool
• **callback** (*function*) – The function to call, setting to *None* will stop the call-back.

• **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*.

**set_when_released** (*callback*, *background=False*)
Sets the function which is called when the Blue Dot is released.

**Parameters**

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

• **background** (*bool*\(^{26}\)) – If set to *True* the function will be run in a separate thread and it will return immediately. The default is *False*.

**set_when_rotated** (*callback*, *background=False*)
Sets the function which is called when the position the Blue Dot is rotated (like an iPod clock wheel).

**Parameters**

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

• **background** (*bool*\(^{27}\)) – If set to *True* the function will be run in a separate thread and it will return immediately. The default is *False*.

**set_when_swiped** (*callback*, *background=False*)
Sets the function which is called when the position the Blue Dot is swiped.

**Parameters**

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

• **background** (*bool*\(^{28}\)) – 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 35) 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*\(^{29}\)) – Number of seconds to wait for a wait connections, if *None* (the default), it will wait indefinitely for a connection from a Blue Dot client.

**wait_for_double_press** (*timeout=None*)
 Waits until a Blue Dot is double pressed. Returns *True* if the Blue Dot was double pressed.

**Parameters**

*timeout* (*float*\(^{30}\)) – Number of seconds to wait for a Blue Dot to be double pressed, if *None* (the default), it will wait indefinitely.

**wait_for_move** (*timeout=None*)
 Waits until the position where the Blue Dot is pressed is moved. Returns *True* if the position pressed on the Blue Dot was moved.

---

\(^{25}\) https://docs.python.org/3.5/library/functions.html#bool

\(^{26}\) https://docs.python.org/3.5/library/functions.html#bool

\(^{27}\) https://docs.python.org/3.5/library/functions.html#bool

\(^{28}\) https://docs.python.org/3.5/library/functions.html#bool

\(^{29}\) https://docs.python.org/3.5/library/functions.html#float

\(^{30}\) https://docs.python.org/3.5/library/functions.html#float
Parameters **timeout** (*float*) – Number of seconds to wait for the position that the Blue Dot is pressed to move, if `None` (the default), it will wait indefinitely.

**wait_for_press** *(timeout=None)*
Waits until a Blue Dot is pressed. Returns `True` if the Blue Dot was pressed.

Parameters **timeout** (*float*) – Number of seconds to wait for a Blue Dot to be pressed, if `None` (the default), it will wait indefinitely.

**wait_for_release** *(timeout=None)*
Waits until a Blue Dot is released. Returns `True` if the Blue Dot was released.

Parameters **timeout** (*float*) – Number of seconds to wait for a Blue Dot to be released, if `None` (the default), it will wait indefinitely.

**wait_for_swipe** *(timeout=None)*
Waits until the Blue Dot is swiped. Returns `True` if the Blue Dot was swiped.

Parameters **timeout** (*float*) – Number of seconds to wait for the Blue Dot to be swiped, if `None` (the default), it will wait indefinitely.

**adapter**
The `btcomm.BluetoothAdapter` (page 38) instance that is being used.

**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 `#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`.

**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.

**interaction**
Returns an instance of `BlueDotInteraction` (page 32) representing the current or last interaction with the Blue Dot.

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

**is_pressed**
Returns `True` if the Blue Dot is pressed (or held).

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

---

31 https://docs.python.org/3.5/library/functions.html#float
32 https://docs.python.org/3.5/library/functions.html#float
33 https://docs.python.org/3.5/library/functions.html#float
34 https://docs.python.org/3.5/library/functions.html#float
```
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.

**position**
Returns an instance of `BlueDotPosition` (page 31) representing the current or last position the Blue Dot was pressed, held or released.

**Note:** If the Blue Dot is released (and inactive), `position` (page 29) will return the position where it was released, until it is pressed again. If the Blue Dot has never been pressed `position` (page 29) will return `None`.

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

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

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

**server**
The `btcomm.BluetoothServer` (page 35) instance that is being used to communicate with clients.

**square**
When set to `True` the ‘dot’ is made square. Default is `False`.

**value**
Returns a 1 if the Blue Dot is pressed, 0 if released.

**values**
Returns an infinite generator constantly yielding the current value.

**visible**
When set to `True` makes the dot invisible. Default is `False`.

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

**when_client_connects**
Sets or returns the function which is called when a Blue Dot 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)`

**when_double_pressed**
Sets or returns the function which is called when the Blue Dot is double pressed.

The function should accept 0 or 1 parameters, if the function accepts 1 parameter an instance of `BlueDotPosition` (page 31) will be returned representing where the Blue Dot was pressed the second time.
The function will be run in the same thread and block, to run in a separate thread use

\texttt{set\_when\_double\_pressed(function, background=True)}

\underline{Note:} The double press event is fired before the 2nd press event e.g. events would be appear in the order, pressed, released, double pressed, pressed.

\textbf{when\_moved}
Sets or returns the function which is called when the position the Blue Dot is pressed is moved.

The function should accept 0 or 1 parameters, if the function accepts 1 parameter an instance of \texttt{BlueDotPosition} (page 31) will be returned representing the new position of where the Blue Dot is held.

The function will be run in the same thread and block, to run in a separate thread use

\texttt{set\_when\_moved(function, background=True)}

\textbf{when\_pressed}
Sets or returns the function which is called when the Blue Dot is pressed.

The function should accept 0 or 1 parameters, if the function accepts 1 parameter an instance of \texttt{BlueDotPosition} (page 31) will be returned representing where the Blue Dot was pressed.

The following example will print a message to the screen when the button is pressed:

\begin{verbatim}
from bluedot import BlueDot
def dot_was_pressed():
    print("The Blue Dot was pressed")
bd = BlueDot()
bd.when_pressed = dot_was_pressed
\end{verbatim}

This example shows how the position of where the dot was pressed can be obtained:

\begin{verbatim}
from bluedot import BlueDot
def dot_was_pressed(pos):
    print("The Blue Dot was pressed at pos x=\{} y=\{}\".format(pos.x, pos.y))
bd = BlueDot()
bd.when_pressed = dot_was_pressed
\end{verbatim}

The function will be run in the same thread and block, to run in a separate thread use

\texttt{set\_when\_pressed(function, background=True)}

\textbf{when\_released}
Sets or returns the function which is called when the Blue Dot is released.

The function should accept 0 or 1 parameters, if the function accepts 1 parameter an instance of \texttt{BlueDotPosition} (page 31) will be returned representing where the Blue Dot was held when it was released.

The function will be run in the same thread and block, to run in a separate thread use

\texttt{set\_when\_released(function, background=True)}

\textbf{when\_rotated}
Sets or returns the function which is called when the Blue Dot is rotated (like an iPod clock wheel).

The function should accept 0 or 1 parameters, if the function accepts 1 parameter an instance of \texttt{BlueDotRotation} (page 33) will be returned representing how the Blue Dot was rotated.

The function will be run in the same thread and block, to run in a separate thread use

\texttt{set\_when\_rotated(function, background=True)}
when_swiped
    Sets or returns the function which is called when the Blue Dot is swiped.
    The function should accept 0 or 1 parameters, if the function accepts 1 parameter an instance of
    BlueDotSwipe (page 32) will be returned representing the how the Blue Dot was swiped.
    The function will be run in the same thread and block, to run in a separate thread use
    set_when_swiped(function, background=True)

7.2 BlueDotPosition

class bluedot.BlueDotPosition(x, y)
    Represents a position of where the blue dot is pressed, released or held.
    Parameters
    • x (float\(^{35}\)) – The x position of the Blue Dot, 0 being centre, -1 being far left and 1
      being far right.
    • y (float\(^{36}\)) – 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 degress 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.

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.

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.

\(^{35}\) https://docs.python.org/3.5/library/functions.html#float
\(^{36}\) https://docs.python.org/3.5/library/functions.html#float
### 7.3 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` 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*) – 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*) – 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` 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`.

### 7.4 BlueDotSwipe

**class** `bluedot.BlueDotSwipe([interaction])`

Represents a Blue Dot swipe interaction.

A `BlueDotSwipe` can be valid or invalid based on whether the Blue Dot interaction was a swipe or not.

**Parameters**
- `interaction` (*BlueDotInteraction*) – 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)

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 32) 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.

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.

7.5 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.

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.
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

```python
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:

```python
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)* – If `True` (the default), the Bluetooth server will be auto-

---

37 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`): 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`): The Bluetooth port the server should use, the default is 1.

- **encoding** (`str`): 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`): 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

```python
disconnect_client()
```

  Disconnects the client if connected. Returns `True` if a client was disconnected.

```python
send(data)
```

  Send data to a connected Bluetooth client

  **Parameters**

  - **data** (`str`): The data to be sent.

```python
start()
```

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

```python
stop()
```

  Stops the Bluetooth server if its running.

```python
adapter
```

  A `BluetoothAdapter` (page 38) object which represents the Bluetooth device the server is using.

```python
client_address
```

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

```python
client_connected
```

  Returns `True` if a client is connected.

```python
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.

```python
device
```

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

38 https://docs.python.org/3.5/library/stdtypes.html#str
39 https://docs.python.org/3.5/library/stdtypes.html#int
40 https://docs.python.org/3.5/library/stdtypes.html#str
41 https://docs.python.org/3.5/library/functions.html#bool
42 https://docs.python.org/3.5/library/stdtypes.html#str
43 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\(^{44}\) 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 “raspberry”, send “helloworld” and print any data is receives:

```python
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\(^{45}\)) – 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\(^{46}\)) – The Bluetooth port the client should use, the default is 1.

- **device** (str\(^{47}\)) – 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\(^{48}\)) – 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.

---

\(^{44}\) https://en.wikipedia.org/wiki/MAC_address

\(^{45}\) https://docs.python.org/3.5/library/stdtypes.html#str

\(^{46}\) https://docs.python.org/3.5/library/functions.html#int

\(^{47}\) https://docs.python.org/3.5/library/stdtypes.html#str

\(^{48}\) https://docs.python.org/3.5/library/stdtypes.html#str
• **power_up_device** (*bool*) – 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:

```sh
sudo rfkill unblock bluetooth
```

• **auto_connect** (*bool*) – If True (the default), the Bluetooth client will automatically try to connect to the server at initialisation, if False, the `connect()` (page 38) method will need to be called.

```python
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:

```python
a = BluetoothAdapter()
print("Powered = {}".format(a.powered))
print(a.paired_devices)
```
Parameters **device** *(str)* – 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)* – The time in seconds the adapter will remain pairable. If set to *None* the device will be discoverable and pairable indefinitely.

**address**
The MAC address 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 adapter *(mac_address, name), (mac_address, name), ...*:

```python
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:

```bash
sudo rfkill unblock bluetooth
```
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

```python
class bluedot.mock.MockBlueDot (device='hci0', port=1, auto_start_server=True, power_up_device=False, print_messages=True)
```

MockBlueDot (page 41) inherits from BlueDot but overrides _create_server(), to create a MockBluetoothServer (page 42) which can be used for testing and debugging.

```python
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.

```python
mock_blue_dot_moved (x, y)
```

Simulates the Blue Dot being moved.

**Parameters**

- `x` (int\(^{56}\)) – The x position where the mock Blue Dot was moved too
- `y` (int\(^{57}\)) – The y position where the mock Blue Dot was moved too

```python
mock_blue_dot_pressed (x, y)
```

Simulates the Blue Dot being pressed.

**Parameters**

- `x` (int\(^{58}\)) – The x position where the mock Blue Dot was pressed

---

\(^{56}\) 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
• \(y\) (int\(^{59}\)) – The y position where the mock Blue Dot was pressed

\texttt{mock\_blue\_dot\_released}(x, y)
Simulates the Blue Dot being released.

\textbf{Parameters}

• \(x\) (int\(^{60}\)) – The x position where the mock Blue Dot was released
• \(y\) (int\(^{61}\)) – The y position where the mock Blue Dot was released

\texttt{mock\_client\_connected()}
Simulates a client connecting to the Blue Dot.

\textbf{Parameters} \texttt{client\_address (string)} – The mock client mac address, defaults to ‘11:11:11:11:11:11’

\texttt{mock\_client\_disconnected()}
Simulates a client disconnecting from the Blue Dot.

\section*{9.2 MockBluetoothServer}

\texttt{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)}

\text{MockBluetoothServer} (page 42) inherits from \texttt{BluetoothServer} (page 35) but overrides \texttt{__init__}, \texttt{start()} (page 42), \texttt{stop()} (page 42) and \texttt{send\_raw()} to create a \texttt{MockBluetoothServer} (page 42) which can be used for testing and debugging.

\texttt{mock\_client\_connected(mock\_client=None)}
Simulates a client connected to the \texttt{BluetoothServer} (page 35).

\textbf{Parameters} \texttt{mock\_client (MockBluetoothClient (page 42))} – The mock client to interact with, defaults to \texttt{None}. If \texttt{None}, client address is set to ‘99:99:99:99:99:99’

\texttt{mock\_client\_disconnected()}
Simulates a client disconnecting from the \texttt{BluetoothServer} (page 35).

\texttt{mock\_client\_sending\_data(data)}
Simulates a client sending data to the \texttt{BluetoothServer} (page 35).

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

\texttt{stop()}
 Stops the Bluetooth server if its running.

\section*{9.3 MockBluetoothClient}

\texttt{class bluedot.mock.MockBluetoothClient(server, data\_received\_callback, port=1, device='mock1', encoding='utf-8', power\_up\_device=False, auto\_connect=True)}

\text{MockBluetoothClient} (page 42) inherits from \texttt{BluetoothClient} (page 37) but overrides \texttt{__init__}, \texttt{connect()} (page 43) and \texttt{send\_raw()} to create a \texttt{MockBluetoothServer} (page 42) which can be used for testing and debugging.

\(^{59}\) 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
Note - the server parameter should be an instance of MockBluetoothServer (page 42).

**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 37).
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, on port 1, 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],[param1],[param2],[*]
```

Messages are sent when:

Blue Dot is released, pressed or moved - [0,1,2],[x],[y]

- operation:
  0. Blue Dot released.
  1. Blue Dot pressed.
  2. Blue Dot pressed position moved.

- x & y specify the position on the Blue Dot that was pressed, released, and/or moved:
  - Positions are values between -1 and +1, with 0 being the centre and 1 being the radius of the Blue Dot.
  - x is the horizontal position where +1 is far right.
– y is the vertical position where +1 is the top.

At connection the client sends a handshake - [3],[protocol version],[client name]

• client to server.
• operation 3.
• protocol version is sent and corresponds to the version of protocol the client supports.
• client name is a string value used in exceptions to report what client has connected.

When the setup (or appearance) of the Blue Dot changes - [4],[color],[square],[border],[visible]:

• server to client.
• operation 4.
• color is a hex value in the format #rrggbbaa representing red, green, blue, alpha values.
• square is 0 or 1, 1 if the dot should be a square.
• border is 0 or 1, 1 if the dot should have a border.
• visible is 0 or 1, 1 if the dot should be visible.

\r represents the ASCII new-line character (ASCII character 10).

10.3 Example

When the Android client connects using protocol version 1:

```
3,1,Android Blue Dot app\n
```

If the blue dot is pressed at the top, the following message will be sent:

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

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

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

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

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

The color of the dot is changed to “red” the server sends to the client:

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

10.4 Protocol versions

• 0 - initial version
• 1 - introduction of operation 3, 4
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
```

Clone repo and install for dev:

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

### 11.2 Test

Install pytest:

```
sudo pip3 install -U pytest
```

Run tests:

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

### 11.3 Deploy

Build for deployment:

[^62]: [https://doc.pytest.org/](https://doc.pytest.org/)

Deploy to PyPi:\footnote{https://pypi.python.org/pypi}{\footnote{https://pypi.python.org/pypi}}:

```
twine upload dist/* --skip-existing
```
12.1 Bluedot Python library

12.1.1 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.

12.1.2 1.3.1 - 2019-01-01

• minor bug fix to launch_mock_app

12.1.3 1.3.0 - 2018-12-30

• added ability to change the color, border, shape and visibility of the dot (color (page 28), border (page 28), square (page 29), visible (page 29))
  • 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

12.1.4 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)
12.1.5 1.2.2 - 2017-12-30

- bluetooth comms tests and minor bug fix in BluetoothClient (page 37)

12.1.6 1.2.1 - 2017-12-18

- massive code and docs tidy up by Dave Jones

12.1.7 1.2.0 - 2017-12-10

- added when_rotated
- threaded swipe callbacks
- exposed new BlueDot (page 25) properties (adapter (page 28), running (page 29), paired_devices (page 28))
- fixed active bug in interaction
- automated tests

12.1.8 1.1.0 - 2017-11-05

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

12.1.9 1.0.4 - 2017-09-10

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

12.1.10 1.0.3 - 2017-07-28

- python 2 bug fix

12.1.11 1.0.2 - 2017-07-23

- bug fix

12.1.12 1.0.1 - 2017-06-19

- bug fixes

12.1.13 1.0.0 - 2017-06-04

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

---

64 https://github.com/waveform80
12.1.14 0.4.0 - 2017-05-05

- added swipes and interactions
- doc updates
- bug fix in `BlueDot.when_moved` (page 30)

12.1.15 0.3.0 - 2017-05-01

- Python Blue Dot app
- minor bug fix in `BluetoothClient` (page 37)

12.1.16 0.2.1 - 2017-04-23

- bug fix in `MockBlueDot`
- doc fixes

12.1.17 0.2.0 - 2017-04-23

- added `when_client_connects` (page 29), `when_client_disconnects` (page 29)
- added `allow_pairing()` (page 26) functions
- refactored Bluetooth comms
- added `BluetoothAdapter` (page 38)

12.1.18 0.1.2 - 2017-04-14

- mock blue dot improvements
- doc fixes

12.1.19 0.1.1 - 2017-04-08

- clamped distance in `BlueDotPosition` (page 31)

12.1.20 0.1.0 - 2017-04-07

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

12.1.21 0.0.6 - 2017-04-05

- Added `MockBlueDot` for testing and debugging
- more docs
12.1.22 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

12.1.23 0.0.2 - 2017-03-29

Alpha - initial testing

12.2 Android app

12.2.1 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

12.2.2 3 (1.1.1) - 2018-09-21
  • Android SDK version uplift (due to google play store minimum requirements change)

12.2.3 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

12.2.4 1 (0.0.2) - 2017-04-05
  • icon transparency
  • connection monitor
  • added info icon to https://bluedot.readthedocs.io
12.2.5  0 (0.0.1) - 2017-03-29

- alpha - initial testing
Python Module Index

b

bluedot, 25
bluedot.btcomm, 35
bluedot.mock, 41
A
active (bluedot.BlueDotInteraction attribute), 32
adapter (bluedot.BlueDot attribute), 28
adapter (bluedot.btcomm.BluetoothClient attribute), 38
adapter (bluedot.btcomm.BluetoothServer attribute), 36
address (bluedot.btcomm.BluetoothAdapter attribute), 39
allow_pairing() (bluedot.BlueDot method), 26
allow_pairing() (bluedot.btcomm.BluetoothAdapter method), 39
angle (bluedot.BlueDotPosition attribute), 31
angle (bluedot.BlueDotSwipe attribute), 32
anti_clockwise (bluedot.BlueDotRotation attribute), 33

B
BlueDot (class in bluedot), 25
bluedot (module), 25
bluedot.btcomm (module), 35
bluedot.mock (module), 41
BlueDotInteraction (class in bluedot), 32
BlueDotPosition (class in bluedot), 31
BlueDotRotation (class in bluedot), 33
BlueDotSwipe (class in bluedot), 32
BluetoothAdapter (class in bluedot.btcomm), 38
BluetoothClient (class in bluedot.btcomm), 37
BluetoothServer (class in bluedot.btcomm), 35
border (bluedot.BlueDot attribute), 28
bottom (bluedot.BlueDotPosition attribute), 31

C
client_address (bluedot.btcomm.BluetoothClient attribute), 38
client_address (bluedot.btcomm.BluetoothServer attribute), 36
client_connected (bluedot.btcomm.BluetoothServer attribute), 36
clockwise (bluedot.BlueDotRotation attribute), 33
color (bluedot.BlueDot attribute), 28
connect () (bluedot.btcomm.BluetoothClient method), 38
connect () (bluedot.mock.MockBluetoothClient method), 43
connected (bluedot.btcomm.BluetoothClient attribute), 38
current_position (bluedot.BlueDotInteraction attribute), 32

data_received_callback (bluedot.btcomm.BluetoothClient attribute), 38
data_received_callback (bluedot.btcomm.BluetoothServer attribute), 36
device (bluedot.BlueDot attribute), 28
device (bluedot.btcomm.BluetoothClient attribute), 39
device (bluedot.btcomm.BluetoothClient attribute), 38
device (bluedot.btcomm.BluetoothServer attribute), 36
disconnect () (bluedot.btcomm.BluetoothClient method), 38
disconnect () (bluedot.mock.MockBluetoothClient method), 43
disconnect_client () (bluedot.btcomm.BluetoothServer method), 36
discordable (bluedot.btcomm.BluetoothAdapter attribute), 39
distance (bluedot.BlueDotInteraction attribute), 32
distance (bluedot.BlueDotPosition attribute), 31
distance (bluedot.BlueDotSwipe attribute), 33
double_press_time (bluedot.BlueDot attribute), 28
down (bluedot.BlueDotSwipe attribute), 33
duration (bluedot.BlueDotSwipe attribute), 32
encoding (bluedot.btcomm.BluetoothClient attribute), 38

E
Index
encoding (bluedot.btcomm.BluetoothServer attribute), 36

I
interaction (bluedot.BlueDot attribute), 28
interaction (bluedot.BlueDotSwipe attribute), 33
is_connected (bluedot.BlueDot attribute), 33
is_pressed (bluedot.BlueDot attribute), 28

L
launch_mock_app() (bluedot.mock.MockBlueDot method), 41
left (bluedot.BlueDotPosition attribute), 31
left (bluedot.BlueDotSwipe attribute), 33

M
middle (bluedot.BlueDotPosition attribute), 31
mock_blue_dot_moved() (bluedot.mock.MockBlueDot method), 41
mock_blue_dot_pressed() (bluedot.mock.MockBlueDot method), 41
mock_blue_dot_released() (bluedot.mock.MockBlueDot method), 42
mock_client_connected() (bluedot.mock.MockBlueDot method), 42
mock_client_connected() (bluedot.mock.MockBluetoothServer method), 42
mock_client_disconnected() (bluedot.mock.MockBlueDot method), 42
mock_client_disconnected() (bluedot.mock.MockBluetoothServer method), 42
mock_client_sending_data() (bluedot.mock.MockBluetoothServer method), 42
mock_server_sending_data() (bluedot.mock.MockBluetoothClient method), 43
MockBlueDot (class in bluedot.mock), 41
MockBluetoothClient (class in bluedot.mock), 42
MockBluetoothServer (class in bluedot.mock), 42
moved() (bluedot.BlueDotInteraction method), 32

P
pairable (bluedot.btcomm.BluetoothAdapter attribute), 39
paired_devices (bluedot.BlueDot attribute), 28
paired_devices (bluedot.btcomm.BluetoothAdapter attribute), 39
port (bluedot.BlueDot attribute), 29
port (bluedot.btcomm.BluetoothClient attribute), 38
port (bluedot.btcomm.BluetoothServer attribute), 37
position (bluedot.BlueDot attribute), 29
positions (bluedot.BlueDotInteraction attribute), 32
powered (bluedot.btcomm.BluetoothAdapter attribute), 39
pressed_position (bluedot.BlueDotInteraction attribute), 32
previous_position (bluedot.BlueDotInteraction attribute), 32
print_messages (bluedot.BlueDot attribute), 29

R
released() (bluedot.BlueDotInteraction method), 32
released_position (bluedot.BlueDotInteraction attribute), 32
right (bluedot.BlueDotPosition attribute), 31
right (bluedot.BlueDotSwipe attribute), 33
rotation_segments (bluedot.BlueDot attribute), 29
running (bluedot.BlueDot attribute), 29
running (bluedot.btcomm.BluetoothServer attribute), 37

S
send() (bluedot.btcomm.BluetoothClient method), 38
send() (bluedot.btcomm.BluetoothServer method), 36
server (bluedot.BlueDot attribute), 29
server (bluedot.btcomm.BluetoothClient attribute), 38
server_address (bluedot.btcomm.BluetoothServer attribute), 37
set_when_client_connects() (bluedot.BlueDot method), 26
set_when_client_disconnects() (bluedot.BlueDot method), 26
set_when_double_pressed() (bluedot.BlueDot method), 26
set_when_moved() (bluedot.BlueDot method), 26
set_when_pressed() (bluedot.BlueDot method), 26
set_when_released() (bluedot.BlueDot method), 27
set_when_rotated() (bluedot.BlueDot method), 27
set_when_swiped() (bluedot.BlueDot method), 27
speed (bluedot.BlueDotSwipe attribute), 33
square (bluedot.BlueDot attribute), 29
start() (bluedot.BlueDot method), 27
start() (bluedot.btcomm.BluetoothServer method), 36
start() (bluedot.mock.MockBluetoothServer method), 42
stop() (bluedot.BlueDot method), 27
stop() (bluedot.btcomm.BluetoothServer method), 36
stop() (bluedot.mock.MockBluetoothServer method), 42

t (bluedot.BlueDotPosition attribute), 31
top (bluedot.BlueDotPosition attribute), 31

U
up (bluedot.BlueDotSwipe attribute), 33

V
valid (bluedot.BlueDotRotation attribute), 33
valid (bluedot.BlueDotSwipe attribute), 33
value (bluedot.BlueDot attribute), 29
value (bluedot.BlueDotRotation attribute), 33
values (bluedot.BlueDotRotation attribute), 33
visible (bluedot.BlueDot attribute), 29

W
wait_for_connection() (bluedot.BlueDot method), 27
wait_for_double_press() (bluedot.BlueDot method), 27
wait_for_move() (bluedot.BlueDot method), 27
wait_for_press() (bluedot.BlueDot method), 28
wait_for_release() (bluedot.BlueDot method), 28
wait_for_swipe() (bluedot.BlueDot method), 28
when_client_connects (bluedot.BlueDot attribute), 29
when_client_connects (bluedot.btcomm.BluetoothServer attribute), 37
when_client_disconnects (bluedot.BlueDot attribute), 29
when_client_disconnects (bluedot.btcomm.BluetoothServer attribute), 37
when_double_pressed (bluedot.BlueDot attribute), 29
when_moved (bluedot.BlueDot attribute), 30
when_pressed (bluedot.BlueDot attribute), 30
when_released (bluedot.BlueDot attribute), 30
when_rotated (bluedot.BlueDot attribute), 30
when_swiped (bluedot.BlueDot attribute), 30

X
x (bluedot.BlueDotPosition attribute), 31

Y
y (bluedot.BlueDotPosition attribute), 31