# bluedot Documentation

***Release 1.0.4***

**Martin O'Hanlon**

**Sep 13, 2017**

# Contents

Blue Dot allows you to control your Raspberry Pi projects wirelessly - its a bluetooth remote and zero boiler plate (super simple to use :) Python library.

Created by Martin O'Hanlon, @martinohanlon, stuffaboutco.de.

CHAPTER 1

# Start

Install and usage is really simple:

1. Get a Blue Dot app
2. Pair your Raspberry Pi
3. Install the Python library
4. Write some code:

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

5. Press the Blue Dot

See the getting started guide to 'get started'!

CHAPTER 2

# More

The Blue Dot is a joystick as well as button, you can tell if the dot was pressed in the middle, on the top, bottom, left or right. Perhaps create a BlueDot controlled Robot.

Why be restricted by such vague positions like top and bottom though, you can get the exact x, y position or even the angle and distance from centre where the dot was pressed.

Its not all about when the button was pressed either - pressed, released or moved they all work.

You can press it, slide it, swipe it - one blue circle can do a lot.

CHAPTER 3

# Even more

The online documentation describes how to use Blue Dot and the Python library including Recipes and ideas.

CHAPTER 4

# Status

Production - under active development. Be sure to raise an issue if you have a feature request or experience problems.

CHAPTER 5

Table of Contents

# Getting started

## What

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)

## Installation

These instructions assume your Raspberry Pi is running the latest version of Raspbian with Pixel.

### android app

If using an Android phone, the Blue Dot app can be installed from the Google Play Store.

### python library

Open a terminal, click `Menu > Accessories > Terminal`:

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

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

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

To upgrade to the latest version:

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

## Usage

### pair

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

### write code

1. Start up Python 3, click `Menu > Programming > Python 3`
2. Click `File > New File` to create a new program
3. Create your python program:

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

4. Save your program, click `File > Save As` and save as `mydot.py`
5. Run the program, click `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`.

### connect

Start-up the Blue Dot app on your Android phone or run the Blue Dot python app on your 2nd Raspberry Pi:

1. Select your Raspberry Pi from the list
2. Press the blue dot

## Where next

Check out the Recipes and the API documentation for more ideas on using bluedot.

# Recipes

The recipes provide examples of how you can use Blue Dot, dont be restricted by these ideas and be sure to have a look at the BlueDot API as there is more to be discovered.

## Button

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

### hello world

Let's say Hello World by creating the BlueDot object then waiting for the Blue Dot app to connect and 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 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()` 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()
```

### flash an led

Using Blue Dot in combination with gpiozero you can interact with electronic components, such as LED's, connected to your Raspberry Pi.

When the Blue Dot is pressed, the LED will turn on, when released it will turn off:

```
from bluedot import BlueDot
from gpiozero import LED

bd = BlueDot()
led = LED(pin)

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(pin)

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

pause()
```

Alternatively use `LED.source` and `BlueDot.values`:

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

bd = BlueDot()
led = LED(pin)

led.source = bd.values

pause()
```

### remote camera

Using a Raspberry Pi camera, picamera and Blue Dot 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()
```

## Joystick

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

### 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 dpad 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:
        print("fire")

bd = BlueDot()
bd.when_pressed = dpad

pause()
```

At the moment the dpad only registers when it is pressed, to get it work when the position is moved you should add:

```
bd.when_moved = dpad
```

### robot

Using the Blue Dot and gpiozero, you can create a bluetooth controlled robot 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=(lfpin, lbpin), right=(rfpin, rbpin))

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()
```

### 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.

`distance` returns how far from the centre the Blue Dot was pressed, which can be passed to the robot's functions to change its speed:

```
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 Robot's source property (courtesy of Ben Nuttall):

```
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()
```

## Slider

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

### center out

Using the `distance` property of the BlueDotPosition which is returned when the position is moved you can create a slide 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()
```

### left to right

The `x` property of the BlueDotPosition returns a value from -1 (far left) to 1 (far right), using this value you can create 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 would change the code above to use the `y` property instead of the `x`.

### dimmer switch

Using the PWMLED class from gpiozero and BlueDot 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

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

pause()
```

## Swiping

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

### 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()
```

### direction

You can tell what direction the Blue Dot is swiped by using the BlueDotSwipe object returned by `when_swiped`:

```
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")

bd = BlueDot()
bd.when_swiped = swiped

pause()
```

### speed, angle, distance

BlueDotSwipe returns more information other than the direction including the speed of the swipe (in Blue Dot radius per second), the angle of the swipe and the distance between the start and end position 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()
```

## Bluetooth

You can interact with the Bluetooth adapter using BlueDot.

### 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:

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

bd = BlueDot()
button = Button(pin)

button.when_pressed = bd.allow_pairing

pause()
```

### paired devices

You can get the devices that your raspberry pi is paired too:

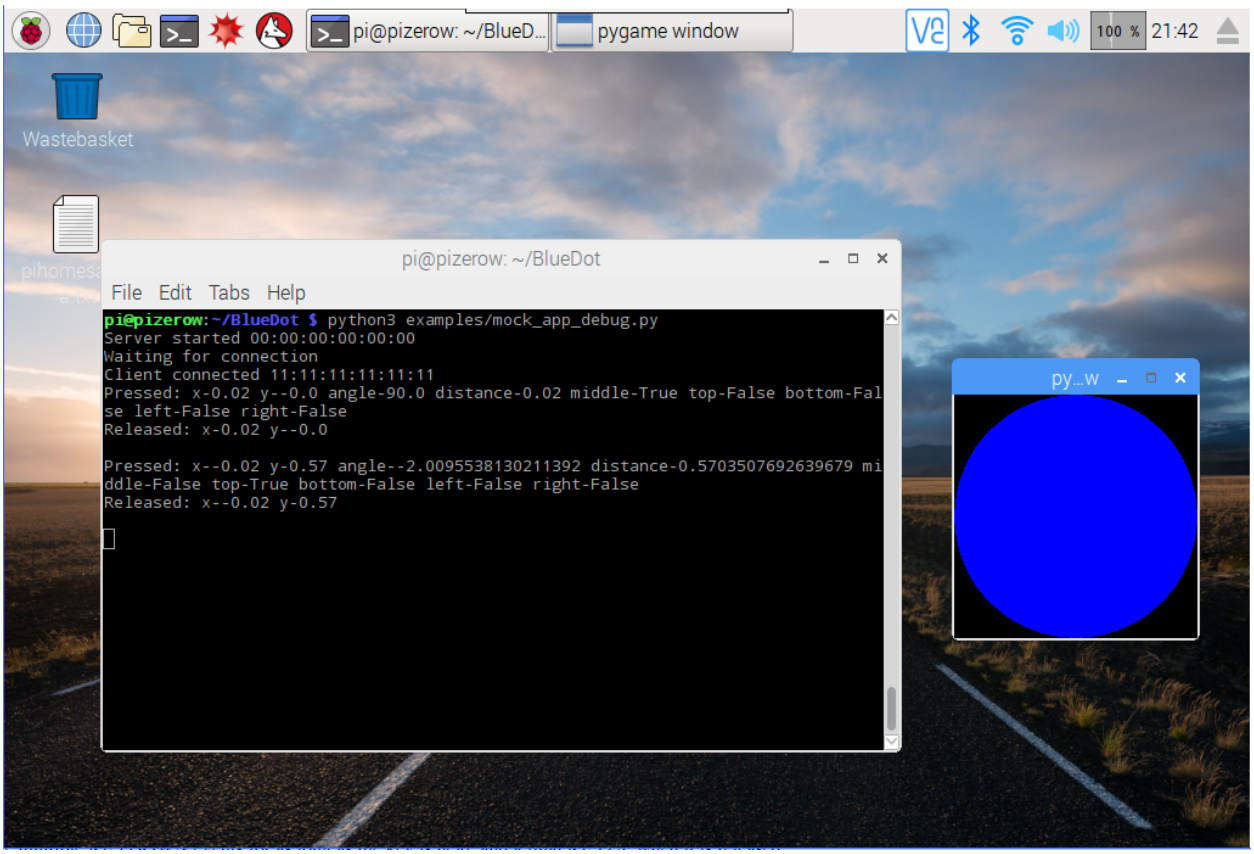
```
from bluedot import BlueDot
bd = BlueDot()

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

## Testing

bluedot 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 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.




### 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()
```

### 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)
```

# Blue Dot Android App

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

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

## Start

1. Download the Blue Dot app from the Google Play store.
2. If you havent already done so, pair your raspberry pi as described in the getting started guide
3. Run the Blue Dot app
4. Select your Raspberry Pi from the paired devices list
5. Press the Dot

# Blue Dot Python app

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

## 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 guide
2. Run the Blue Dot app:

   ```
   python3 -m bluedot.app
   ```

4. Select your Raspberry Pi from the paired devices list
5. Press the Dot

## Options

To get help with the Blue Dot app options:

```
python3 -m bluedot.app --help
```

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

```
python3 -m bluedot.app --device hci1
```

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

```
python3 -m bluedot.app --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:

```
python3 -m bluedot.app --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:

```
python3 -m bluedot.app --fullscreen
```

# Blue Dot API

## 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 a bluetooth server (btcomm.BluetoothServer) 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:

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

**Parameters**

- **`device`** (*`string`*) – 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` will need to use 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:

  ```
  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 / disconect.

**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 indefinetly.

**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` representing the current or last interaction with the Blue Dot.

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

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

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

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

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

Note - if the Blue Dot is released (and inactive), `position` will return position when it was released, until it is pressed again. If the Blue Dot has never been pressed `position` will return `None`.

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

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

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

**stop**()
Stop the bluetooth server.

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

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

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

**Parameters** **timeout** (*float*) – 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.

**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*) – Number of seconds to wait for a Blue Dot to be double pressed, if None (the default), it will wait indefinetly.

**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.

**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 indefinetly.

**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 indefinetly.

**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 indefinetly.

**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 indefinetly.

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

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

**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 will be returned representing where the Blue Dot was pressed the second time.

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.

**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 BlueDotPosition will be returned representing the new position of where the Blue Dot is held.

**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 BlueDotPosition 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:

```
from bluedot import BlueDot

def dot_was_pressed():
    print("The Blue Dot was pressed")
```

```
bd = BlueDot()
bd.when_pressed = dot_was_pressed
```

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

```
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
```

**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 `BlueDotPosition` will be returned representing where the Blue Dot was held when it was released.

**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` will be returned representing the how the Blue Dot was swiped.

## BlueDotPosition

**class** `bluedot.`**BlueDotPosition**(*x*, *y*)
Represents a position of where the blue for is pressed, released or held.

**Parameters**

- **x** (*float*) – The x position of the Blue Dot, 0 being centre, `-1` being far left and `1` being far right.
- **y** (*float*) – 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, 0 > -180 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.

## 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.

**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.

**moved**(*moved_position*)
Adds an additional position to the interaction, called when the position the Blue Dot is pressed moves.

**positions**
A list of `BlueDotPositions` 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.

**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.

**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`.

## 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 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.

## MockBlueDot

**class** bluedot.**MockBlueDot**(*device='hci0'*, *port=1*, *auto_start_server=True*, *power_up_device=False*, *print_messages=True*)
MockBlueDot inherits from BlueDot but overrides ._create_server, to create a MockBluetoothServer 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 BlueDot without having to script mock functions.

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

**mock_blue_dot_moved**(*x*, *y*)
Simulates the blue dot being moved.

**Parameters**

- **x** (*int*) – The x position where the mock blue dot was moved too
- **y** (*int*) – The y position where the mock blue dot was moved too

**mock_blue_dot_pressed**(*x*, *y*)
Simulates the blue dot being pressed.

**Parameters**

- **x** (*int*) – The x position where the mock blue dot was pressed
- **y** (*int*) – The y position where the mock blue dot was pressed

**mock_blue_dot_released**(*x*, *y*)
Simulates the blue dot being released.

**Parameters**

- **x** (*int*) – The x position where the mock blue dot was released
- **y** (*int*) – The y position where the mock blue dot was released

**mock_client_connected**(*client_address='11:11:11:11:11:11'*)
Simulates a client connecting to the BlueDot.

**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 BlueDot.

# Bluetooth Comm API

Blue Dot also contains a useful API `bluedot.btcomm` 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.

## 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*) – If `True` (the default), the bluetooth server will be automatically started on initialisation, if `False`, the method `start` will need to be called before connections will be accepted.
- **device** (*string*) – 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** (*string*) – 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

**adapter**
A BluetoothAdapter object which represents the bluetooth device the server is using.

**client_address**
The mac address 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.

**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.

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

**Parameters** **data** (*string*) – The data to be sent.

**server_address**
The mac address of the device the server is using.

**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.

**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.

## 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** (*string*) – The server name ("raspberrypi") or server mac address ("11:11:11:11:11:11") to connect too.

  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*) – The bluetooth port the client should use, the default is `1`

- **device** (*string*) – 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** (*string*) – 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
```

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

**adapter**
A BluetoothAdapter object which represents the bluetooth device the client is using.

**client_address**
The mac address of the device being used.

**connect**()
Connect to a bluetooth server.

**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.

**disconnect**()
Disconnect from a bluetooth server.

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

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

**send**(*data*)
Send data to a bluetooth server

**Parameters** **data** (*string*) – The data to be sent.

**server**
The server name ("raspberrypi") or server mac address ("11:11:11:11:11:11") to connect too.

## 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)
```

**Parameters** **device** (*string*) – 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.

**address**
The mac address of the bluetooth adapter.

**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 indefinetly.

**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 list of devices paired with this adapater `((device_mac_address, device_name), (device_mac_address, device_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
```

# Protocol

Blue Dot uses a client/server model, the bluedot Python library 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 :)

## Bluetooth

Communication over Bluetooth is made using a RFCOMM serial port profile, on port 1, using UUID "00001101-0000-1000-8000-00805f9b34fb".

## Protocol

The transmission is a 1 way stream between client and server, the server sends no acknowledgements or data to the client.

All messages between client and server conforms to the same format:

```
[operation],[x],[y]\n
```

`operation` is either 0, 1 or 2:

0. Blue Dot released.
1. Blue Dot pressed.
2. Blue Dot pressed position moved.

`x` & `y` is the position on the Blue Dot where button was pressed, released, moved.

Positions are values between -1 and +1, with 0 being the centre, 1 being the radius of the Blue Dot.

`x` is the horizontal position, +1 is far right.

`y` is the vertical position, +1 is the top.

e.g.

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

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

while the blue dot is pressed (held down), the position it is pressed moves to the far right:

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

when then button is released:

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

If positions cannot be sent, `x` and `y` should still be sent but be defaulted to `0`.

Messages should always be terminated with `\n`.

# Changelog

## Bluedot python library

### 1.0.4 - 2017-09-10

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

### 1.0.3 - 2017-07-28

- python 2 bug fix

### 1.0.2 - 2017-07-23

- bug fix

### 1.0.1 - 2017-06-19

- bug fixes

### 1.0.0 - 2017-06-04

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

### 0.4.0 - 2017-05-05

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

### 0.3.0 - 2017-05-01

- Python Blue Dot app
- minor bug fix in BluetoothClient

### 0.2.1 - 2017-04-23

- bug fix in MockBlueDot
- doc fixes

### 0.2.0 - 2017-04-23

- added when_client_connects, when_client_disconnects
- added allow_pairing functions
- refactored Bluetooth comms
- added BluetoothAdapter

### 0.1.2 - 2017-04-14

- mock blue dot improvements
- doc fixes

### 0.1.1 - 2017-04-08

- clamped distance in BlueDotPosition

#### 0.1.0 - 2017-04-07

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

#### 0.0.6 - 2017-04-05

- Added MockBlueDot for testing and debugging
- more docs

#### 0.0.4 - 2017-03-31

Updates after alpha feedback

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

#### 0.0.2 - 2017-03-29

Alpha - initial testing

### android app

#### 1.0 (0.0.2) - 2017-04-05

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

#### 0.0 (0.0.1) - 2017-03-29

Alpha - initial testing

# Python Module Index

## b

# Index

## P



## R



## S



## T



## U



## V



## W



## X



## Y