Bleson Documentation

TheCellule

Jun 14, 2022

Contents

CHAPTER 1

Installing

Bleson is currently only compatible with Python 3.

1.1 Linux

```
$ sudo pip3 install bleson
```

Note: To run scripts without having to run as root to acced the Bluetooth LE adapter you can use the *setcap* utility to give the Python3 binary necessary permission, for example:

```
sudo setcap cap_net_raw,cap_net_admin+eip $(eval readlink -f `which python3`)
```

Note: It's currently not necessary to stop the bluetoothd service for observing, but because Bleson uses the HCI socket interface directly, it is recommended.

For example, on a Raspberry Pi run:

```
sudo service bluetooth stop
```

1.2 macOS

```
$ pip3 install bleson
```

You may have to use sudo if you are not using *homebrew*.

Note: On Mac the system bundled Python is version 2, you will need Python 3 from Python.

Note: Currently only macOS 10.12+ is supported, soon to be resolved (and support macOS 10.9+) when PyObjC 4.1 is released.

1.3 Windows 10

```
$ pip install bleson
```

Note: Only Windows 10 Fall Creators Update (build 16299) has been tested and known to work.

Earlier version of Windows are not supported, and are unlikely to ever be.

Note: You must use pip v9.0 or newer to be able to install bleson's precompiled native components ('wheels') on Mac and Windows.*

Note: The bleson module depends on an native module, *blesonwin*, it will be automatically installed. The 'blesonwin' binary (wheel) pacakges have been pre-built for 32bit and 64bit architectures for Python 3.5 and Python 3.6

1.4 MicroPython

A port of MicroPython to the Apache MyNewt OS with Bluetooth LE support and a native module implementation of the Bleson API is under development for:

- micro:bit
- Adafruit Feather nRF52 Pro
- RuuviTag
- nRF51 dev kit
- nRF52 dev kit

Firmware images are downloadable from TheBubbleworks. The Advertiser and Beacon examples should work as is.

Note: It's very alpha, currently only supporting very basic Advertising (name only) or Physical Web beacons.

Note: Regarding the micro:bit release

There is currently 64k of flash space free, but there is no filesystem currently exposed to uPY (also the micro:bit libraries aren't currently included), so you have to send the script via the USB serial REPL after every power on, for now.

When you do paste if you use a terminal emulation program that can add inter character delays of around 8ms, e.g. CoolTerm available on all the major platforms, you will have more luck pasting text in without characters going missing.

CHAPTER 2

Examples

2.1 Observer example

Scan for local devices.

```
from time import sleep
from bleson import get_provider, Observer

def on_advertisement(advertisement):
   print(advertisement)

adapter = get_provider().get_adapter()

observer = Observer(adapter)
observer.on_advertising_data = on_advertisement

observer.start()
sleep(2)
observer.stop()
```

This will output one or more lines containing an AdvertisingReport for each Bluetooth LE device found, for example:

```
Advertisement(...)
```

2.2 Advertiser example

Advertise the host as a Bluetooth LE device with the name *bleson*

```
from time import sleep
```

(continues on next page)

(continued from previous page)

```
from bleson import get_provider, Advertiser, Advertisement

adapter = get_provider().get_adapter()

advertiser = Advertiser(adapter)
advertisement = Advertisement()
advertisement.name = "bleson"

advertiser.advertisement = advertisement

advertiser.start()
sleep(10)
advertiser.stop()
```

2.3 Peripheral example

Create a simple Peripheral. (This currently only works on the micro:bit port.)

```
from time import sleep
from bleson import get_default_adapter, Peripheral, Service, Characteristic,␣
↪CHAR_WRITE

adapter = get_default_adapter()

peripheral = Peripheral(adapter)

MICROBIT_LED_SERVICE = 'E95DD91D-251D-470A-A062-FA1922DFA9A8'
MICROBIT_LED_CHAR = 'E95D7B77-251D-470A-A062-FA1922DFA9A8'

def on_data_received(bytes):
    print(bytes)

led_service = Service(MICROBIT_LED_SERVICE)
led_write_char = Characteristic(MICROBIT_LED_CHAR, CHAR_WRITE)
led_write_char.on_data_received = on_data_received

led_service.add_characteristic(led_write_char)

peripheral.add_service(led_service)

peripheral.start()
sleep(2)
peripheral.stop()
```

2.4 PhysicalWeb beacon example

Create a PhysicalWeb (Eddystone) with a URL

```
from time import sleep
from bleson import get_default_adapter, EddystoneBeacon
```

(continues on next page)

(continued from previous page)

```
adapter = get_default_adapter()

beacon = EddystoneBeacon(adapter)
beacon.url = 'https://www.bluetooth.com/'
beacon.start()
sleep(2)
beacon.stop()
```

2.5 Further Reading

Please see examples for more details.

- Examples prefixed with 'basic' shows basic Bleson API usage.
- Examples prefixed with 'context' shows Blesons context manger ('with' keyword) API usage.

CHAPTER 3

Command-Line tools

Bleson ships with built-in utilities that can be run directly from the command line.

3.1 Observer

Scan for nearby Bluetooth LE devices and print out any Advertising Reports received:

```
$ python3 -m bleson --observer
```

3.2 Beacon

Start a PhysicalWeb (Eddystone) beacon advertising a URL:

```
$ python3 -m bleson --beacon --url http://www.google.com/
```

CHAPTER 4

API - Core Data Types

Value objects representing Bluetooth Core Spec data types.

4.1 UUID16

class bleson.core.types.**UUID16**(*uuid*, *little_endian=True*)

16Bit UUID Type

Parameters

- **uuid** (*string, list, tuple, bytes or bytearray*) – The 16bit uuid source value
- **little_endian** (*bool*) – Byte order, default True

Example of initialisation with a 16bit integer:

```
print(UUID16(0xFFFF))
```

Output:

```
UUID16(0xffff)
```

Example of initialisation with a list:

```
print(UUID16([0x34, 0x12]))
```

Output:

```
UUID16(0x1234)
```

4.2 UUID128

class bleson.core.types.**UUID128**(*uuid*, *little_endian=True*)

128 Bit Type.

Parameters

- **uuid** (*string, list, tuple, bytes or bytearray*) – The 16bit uuid source value
- **little_endian** (*bool*) – Byte order, default True

Example of initialisation with a list:

```
print(UUID128([0x23, 0xD1, 0xBC, 0xEA, 0x5F, 0x78, 0x23, 0x15, 0xDE, 0xEF, 0x12,␣
↪0x12, 0x30, 0x15, 0x00, 0x00]))
```

Output:

```
UUID128('00001530-1212-efde-1523-785feabcd123')
```

Example of initialisation with a 16bit integer:

```
print(UUID128(0x1234))
```

Output:

```
UUID128('00001234-0000-1000-8000-00805f9b34fb')
```

4.3 BDAddress

class bleson.core.types.**BDAddress**(*address=None*)

Bluetooth Device Address.

Parameters **address** – The device bluetooth address

Example of initialisation with a list:

```
print(BDAddress([0xab, 0x90, 0x78, 0x56, 0x34, 0x12]))
```

Output:

```
BDAddress('12:34:56:78:90:AB')
```

Example of initialisation with a bytearray:

```
print(BDAddress(bytearray([0xab, 0x90, 0x78, 0x56, 0x34, 0x12])))
```

Output:

```
BDAddress('12:34:56:78:90:AB')
```

4.4 Device

class bleson.core.types.**Device**(*address: bleson.core.types.BDAddress = None, name=None, rssi=None*)

Bluetooth LE Device Info.

Parameters

- **address** (BDAddress) – Bluetooth Device Address
- **name** (*str*) – device name
- **rssi** (*integer*) – device RSSI

Example of initialisation with list:

```
print(Device(address=BDAddress('12:34:56:78:90:AB'), name='bleson', rssi=-99))
```

Output:

```
Device(address=BDAddress('12:34:56:78:90:AB'), name='bleson', rssi=-99)
```

4.5 Advertisement

class bleson.core.types.**Advertisement**(*name=None, address=None, rssi=None, tx_power=None, data_format='simple', raw_data=None*)

Bluetooth LE AdvertisementReport

Parameters

- **address** (BDAddress) – Bluetooth Device Adress
- **name** (*str*) – device name
- **rssi** (*integer*) – device RSSI
- **tx_power** (*integer*) – device transmit power
- **raw_data** (*bytearray on Linux (HCI data) or a dictionary on macOS/Win*) – pre-rolled advertisement payload

Example of initialisation with list:

```
print(Advertisement(address=BDAddress('12:34:56:78:90:AB'), name='bleson', rssi=-
↪99, tx_power=0))
```

Output:

```
Advertisement(flags=0x06, name='bleson', txpower=0, uuid16s=[], uuid128s=[],␣
↪rssi=-99, mfg_data=None)
```

CHAPTER 5

API - Core Bluetooth LE Roles

5.1 Advertiser Role

class bleson.core.roles.**Advertiser**(*adapter: bleson.interfaces.adapter.Adapter*, *advertisement=None*, *scan_response=None*)

start()
Start the role.

stop()
Stop the role.

Note: macOS implementation note

You can only set the name and the services UUID's, and the visibility of your advertisement data depends on your app being a foreground or background process.

See the CoreBluetooth documentation.

5.2 Observer Role

class bleson.core.roles.**Observer**(*adapter: bleson.interfaces.adapter.Adapter*, *on_advertising_data=None*)

start()
Start the observer

stop()
Stop the role.

CHAPTER 6

API - Additional Bluetooth LE Roles

6.1 Beacons

class bleson.beacons.eddystone.**EddystoneBeacon**(*adapter*, *url=None*)

PhysicalWeb (Eddystone) Beacon Advertiser

Parameters

- **adapter** (*bleson.interfaces.adapter.Adapter*) – bluetooth adapter
- **url** (*str*) – URL to publish, maximum length of 17

Example of initialisation with a URL:

```
print(EddystoneBeacon('www.bluetooth.com'))
```

Output:

```
<bleson.beacons.eddystone.EddystoneBeacon object at ...>
```

CHAPTER 7

API - Other

7.1 Platform Provider

Obtain the Bluetooth LE provider for the current platform (Linux/macOS/Windows)

bleson.**get_provider**()

7.2 Logging

bleson.logger.**set_level**(*level*)
Set the Bleson module logging level.

CHAPTER 8

Contributing

Contributions to the library are welcome! Here are some guidelines to follow.

8.1 Suggestions

Please make suggestions for additional components or enhancements to the codebase by opening an issue explaining your reasoning clearly.

8.2 Bugs

Please submit bug reports by opening an issue explaining the problem clearly using code examples.

8.3 Documentation

The documentation source lives in the docs folder. Contributions to the documentation are welcome but should be easy to read and understand.

8.4 Commit messages and pull requests

Commit messages should be concise but descriptive, and in the form of a patch description, i.e. instructional not past tense ("Add beacon example" not "Added beacon example").

Commits which close (or intend to close) an issue should include the phrase "fix #123" or "close #123" where `#123` is the issue number, as well as include a short description, for example: "Add beacon example, close #123", and pull requests should aim to match or closely match the corresponding issue title.

8.5 Backwards compatibility

Not particularly releveant yet, but one aspiration is to keep the public API as unchanged as possible in the alpha and beta phases.

8.6 Python 2/3

The library is only compatible with Python 3.

CHAPTER 9

Development

The main GitHub repository for the project can be found at:

https://github.com/TheCellule/python-bleson

It's strongly recommended to update your tools:

```
pip3 install --upgrade pip setuptools wheel twine
```

9.1 Building the docs

Build the documentation:

```
python3 setup.py doc
```

Build the documentation and run the doctests:

```
python3 setup.py doctest
```

9.2 Test suite

```
python3 setup.py test
```

9.3 Release to PyPi

A CI integration takes care of testing and then publishing to PyPi, all you need todo is

- merge the changes into master,

- bump *bleson/VERSION*,
- *git add*
- *git commit*
- run:

```
python3 setup.py tag
```

CHAPTER 10

Internal API

10.1 Abstract Interfaces

class bleson.interfaces.provider.**Provider**

get_adapter(*adapter_id=None*)
Return an Adapter instance, default to first one available

class bleson.interfaces.adapter.**Adapter**
Adapter interface

off()
Power off the adapter hardware.

on()
Power on the adapter hardware.

open()
Initialise the adapter.

start_advertising(*advertisement*, *scan_response=None*)
Start BLE advertising.

start_scanning()
Start BLE scanning.

stop_advertising()
Stop BLE advertising.

stop_scanning()
Stop BLE scanning.

class bleson.interfaces.role.**Role**

start()
Start the role.

stop()
Stop the role.

class bleson.core.types.**ValueObject**

__bytes__()
Return bytes in little endian byte order.

__len__()

10.2 Platform Implementations

10.2.1 Linux

Bleson uses the Linux kernel's HCI Sockets interface, is not dependent on the userland BlueZ service or binaries, it's pure Python sockets.

class bleson.providers.linux.linux_provider.**LinuxProvider**

get_adapter(*adapter_id=0*)
Return an Adapter instance, default to first one available

class bleson.providers.linux.linux_adapter.**BluetoothHCIAdapter**(*device_id=0*)

off()
Power off the adapter hardware.

on()
Power on the adapter hardware.

open()
Initialise the adapter.

start_advertising(*advertisement*, *scan_response=None*)
Start BLE advertising.

start_scanning()
Start BLE scanning.

stop_advertising()
Stop BLE advertising.

stop_scanning()
Stop BLE scanning.

10.2.2 macOS

The CoreBluetooth dispatch queue is run in a background thread, requires the use of features added in PyObjC 4.1 for macOS < 10.12.

class bleson.providers.macos.macos_provider.**MacOSProvider**

get_adapter(*adapter_id=0*)
Return an Adapter instance, default to first one available

class bleson.providers.macos.macos_adapter.**CoreBluetoothAdapter**(*device_id=0*)

off()
Power off the adapter hardware.

on()
Power on the adapter hardware.

open()
Initialise the adapter.

start_advertising(*advertisement*, *scan_response=None*)
Start BLE advertising.

start_scanning()
Start BLE scanning.

stop_advertising()
Stop BLE advertising.

stop_scanning()
Stop BLE scanning.

10.2.3 Windows

On Windows there is an additional module used called 'blesonwin', it provides a Python native module to access the WinRT BLE API's. It's not recommended to use this module directly in user scripts.

class bleson.providers.win32.win32_provider.**Win32Provider**

get_adapter(*adapter_id=0*)
Return an Adapter instance, default to first one available

class bleson.providers.win32.win32_adapter.**BluetoothAdapter**(*device_id=0*)

off()
Power off the adapter hardware.

on()
Power on the adapter hardware.

open()
Initialise the adapter.

start_advertising(*advertisement*, *scan_response=None*)
Start BLE advertising.

start_scanning()
Start BLE scanning.

stop_advertising()
Stop BLE advertising.

stop_scanning()
Stop BLE scanning.

CHAPTER 11

Indices and tables

- genindex
- modindex
- search

Python Module Index

b

Index

Symbols

A

B

C

D

E

G

L

M

O

P

R

S

U

V

W