bleak Documentation

Release 0.12.1

Henrik Blidh

Jul 07, 2021

Contents

Bleak is an acronym for Bluetooth Low Energy platform Agnostic Klient.

- Free software: MIT license
- Documentation: https://bleak.readthedocs.io.

Bleak is a GATT client software, capable of connecting to BLE devices acting as GATT servers. It is designed to provide a asynchronous, cross-platform Python API to connect and communicate with e.g. sensors.

CHAPTER 1

Features

- Supports Windows 10, version 16299 (Fall Creators Update) or greater
- Supports Linux distributions with BlueZ >= 5.43 (See *Linux backend* for more details)
- OS X/macOS support via Core Bluetooth API, from at least OS X version 10.11

Bleak supports reading, writing and getting notifications from GATT servers, as well as a function for discovering BLE devices.

Contents:

1.1 Installation

1.1.1 Stable release

To install bleak, run this command in your terminal:

```
$ pip install bleak
```

This is the preferred method to install bleak, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

1.1.2 From sources

The sources for bleak can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/hbldh/bleak
```

Or download the tarball:

```
$ curl  -OL https://github.com/hbldh/bleak/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

1.1.3 Building *BleakUWPBridge*

TBW.

1.2 Scan/Discover

1.2.1 BleakScanner

The `<BleakScanner> bleak.backends.scanner.BleakScanner` class is used to discover Bluetooth Low Energy devices by monitoring advertising data.

To discover Bluetooth devices that can be connected to:

```
import asyncio
from bleak import BleakScanner

async def run():
    devices = await BleakScanner.discover()
    for d in devices:
        print(d)

loop = asyncio.get_event_loop()
loop.run_until_complete(run())
```

This will scan for 5 seconds and then produce a printed list of detected devices:

```
24:71:89:CC:09:05: CC2650 SensorTag
4D:41:D5:8C:7A:0B: Apple, Inc. (b'\x10\x06\x11\x1a\xb2\x9b\x9c\xe3')
```

The first part, a Bluetooth address in Windows and Linux and a UUID in macOS, is what is used for connecting to a device using Bleak. The list of objects returned by the *discover* method are instances of *`bleak.backends.device.BLEDevice`* and has `name`, `address` and `rssi` attributes, as well as a `metadata` attribute, a dict with keys `uuids` and `manufacturer_data` which potentially contains a list of all service UUIDs on the device and a binary string of data from the manufacturer of the device respectively.

It can also be used as an object, either in an asynchronous context manager way:

```
import asyncio
from bleak import BleakScanner

async def run():
    async with BleakScanner() as scanner:
        await asyncio.sleep(5.0)
    for d in scanner.discovered_devices:
        print(d)

loop = asyncio.get_event_loop()
loop.run_until_complete(run())
```

or separately, calling `start` and `stop` methods on the scanner manually:

```
import asyncio
from bleak import BleakScanner

def detection_callback(device, advertisement_data):
    print(device.address, "RSSI:", device.rssi, advertisement_data)

async def run():
    scanner = BleakScanner()
    scanner.register_detection_callback(detection_callback)
    await scanner.start()
    await asyncio.sleep(5.0)
    await scanner.stop()

    for d in scanner.discovered_devices:
        print(d)

loop = asyncio.get_event_loop()
loop.run_until_complete(run())
```

In the manual mode, it is possible to add an own callback that you want to call upon each scanner detection, as can be seen above. There are also possibilities of adding scanning filters, which differ widely between OS backend implementations, so the instructions merit careful reading.

1.2.2 Scanning Filters

There are some scanning filters that can be applied, that will reduce your scanning results prior to them getting to bleak. These are quite backend specific, but they are generally used like this:

- On the *discover* method, send in keyword arguments according to what is described in the docstring of the method.
- On the backend's *BleakScanner* implementation, either send in keyword arguments according to what is described in the docstring of the class or use the `set_scanning_filter` method to set them after the instance has been created.

Scanning filters are currently implemented in Windows and BlueZ backends, but not yet in the macOS backend.

Scanning filter examples in .NET backend

To be written. In the meantime, check docstrings here and check out issue #230.

Scanning filter examples in BlueZ backend

To be written. In the meantime, check docstrings.

Scanning filter examples in Core Bluetooth backend

To be implemented. Exists in a draft in PR #209.

1.3 Usage

Note: A Bluetooth peripheral may have several characteristics with the same UUID, so the means of specifying characteristics by UUID or string representation of it might not always work in bleak version > 0.7.0. One can now also use the characteristic's handle or even the `BleakGATTCharacteristic` object itself in `read_gatt_char`, `write_gatt_char`, `start_notify`, and `stop_notify`.

One can use the `BleakClient` to connect to a Bluetooth device and read its model number via the asyncronous context manager like this:

```
import asyncio
from bleak import BleakClient

address = "24:71:89:cc:09:05"
MODEL_NBR_UUID = "00002a24-0000-1000-8000-00805f9b34fb"

async def run(address):
    async with BleakClient(address) as client:
        model_number = await client.read_gatt_char(MODEL_NBR_UUID)
        print("Model Number: {0}".format("".join(map(chr, model_number))))

loop = asyncio.get_event_loop()
loop.run_until_complete(run(address))
```

or one can do it without the context manager like this:

```
import asyncio
from bleak import BleakClient

address = "24:71:89:cc:09:05"
MODEL_NBR_UUID = "00002a24-0000-1000-8000-00805f9b34fb"

async def run(address):
    client = BleakClient(address)
    try:
        await client.connect()
        model_number = await client.read_gatt_char(MODEL_NBR_UUID)
        print("Model Number: {0}".format("".join(map(chr, model_number))))
    except Exception as e:
        print(e)
    finally:
        await client.disconnect()

loop = asyncio.get_event_loop()
loop.run_until_complete(run(address))
```

Make sure you always get to call the disconnect method for a client before discarding it; the Bluetooth stack on the OS might need to be cleared of residual data which is cached in the `BleakClient`.

See examples folder for more code, e.g. on how to keep a connection alive over a longer duration of time.

1.4 Bleak backends

Bleak supports the following operating systems:

- Windows 10, version 16299 (Fall Creators Update) and greater
- Linux distributions with BlueZ >= 5.43 (See *Linux backend* for more details)
- OS X/macOS support via Core Bluetooth API, from at least version 10.11

These pages document platform specific differences from the interface API.

Contents:

1.4.1 Windows backend

The Windows backend of bleak is written using the Python for .NET package. Combined with a thin bridge library (BleakUWPBridge) that is bundled with bleak, the .NET Bluetooth components can be used from Python.

The Windows backend implements a `BleakClient` in the module `bleak.backends.dotnet.client`, a `BleakScanner` method in the `bleak.backends.dotnet.scanner` module. There are also backend-specific implementations of the `BleakGATTService`, `BleakGATTCharacteristic` and `BleakGATTDescriptor` classes.

Finally, some .NET/`asyncio`-connectivity methods are available in the `bleak.backends.dotnet.utils` module.

Specific features for the Windows backend

Client

- The constructor keyword `address_type` which can have the values `"public"` or `"random"`. This value makes sure that the connection is made in a fashion that suits the peripheral.

1.4.2 Linux backend

The Linux backend of Bleak is written using the TxDBus package. It is written for Twisted, but by using the twisted.internet.asyncioreactor one can use it with *asyncio*.

Special handling for `write_gatt_char`

The `type` option to the `Characteristic.WriteValue` method was added to Bluez in 5.51 Before that commit, `Characteristic.WriteValue` was only "Write with response".

`Characteristic.AcquireWrite` was added in Bluez 5.46 which can be used to "Write without response", but for older versions of Bluez (5.43, 5.44, 5.45), it is not possible to "Write without response".

1.4.3 macOS backend

The macOS backend of Bleak is written with pyobjc directives for interfacing with Foundation and CoreBluetooth APIs.

Specific features for the macOS backend

The most noticeable difference between the other backends of bleak and this backend, is that CoreBluetooth doesn't scan for other devices via Bluetooth address. Instead, UUIDs are utilized that are often unique between the device that is scanning and the device that is being scanned.

In the example files, this is handled in this fashion:

```
mac_addr = (
    "24:71:89:cc:09:05"
    if platform.system() != "Darwin"
    else "243E23AE-4A99-406C-B317-18F1BD7B4CBE"
)
```

As stated above, this will however only work the macOS machine that performed the scan and thus cached the device as `243E23AE-4A99-406C-B317-18F1BD7B4CBE`.

There is also no pairing functionality implemented in macOS right now, since it does not seem to be any explicit pairing methods in the COre Bluetooth.

1.5 Interfaces, exceptions and utils

1.5.1 Connection Clients

Interface

Base class for backend clients.

Created on 2018-04-23 by hbldh <henrik.blidh@nedomkull.com>

class bleak.backends.client.**BaseBleakClient**(*address_or_ble_device: Union[bleak.backends.device.BLEDevice, str], **kwargs*)

The Client Interface for Bleak Backend implementations to implement.

The documentation of this interface should thus be safe to use as a reference for your implementation.

Parameters **address_or_ble_device** (*BLEDevice* or str) – The Bluetooth address of the BLE peripheral to connect to or the *BLEDevice* object representing it.

Keyword Arguments

- **timeout** (*float*) – Timeout for required `discover` call. Defaults to 10.0.
- **disconnected_callback** (*callable*) – Callback that will be scheduled in the event loop when the client is disconnected. The callable must take one argument, which will be this client object.

connect(***kwargs*) → bool

Connect to the specified GATT server.

Returns Boolean representing connection status.

disconnect() → bool

Disconnect from the specified GATT server.

Returns Boolean representing connection status.

get_services(***kwargs*) → bleak.backends.service.BleakGATTServiceCollection

Get all services registered for this GATT server.

Returns A *bleak.backends.service.BleakGATTServiceCollection* with this device's services tree.

is_connected
Check connection status between this client and the server.

Returns Boolean representing connection status.

pair(**args*, ***kwargs*) → bool
Pair with the peripheral.

read_gatt_char(*char_specifier: Union[bleak.backends.characteristic.BleakGATTCharacteristic, int, str, uuid.UUID], **kwargs*) → bytearray
Perform read operation on the specified GATT characteristic.

Parameters **char_specifier** (*BleakGATTCharacteristic, int, str or UUID*) – The characteristic to read from, specified by either integer handle, UUID or directly by the BleakGATTCharacteristic object representing it.

Returns (bytearray) The read data.

read_gatt_descriptor(*handle: int*, ***kwargs*) → bytearray
Perform read operation on the specified GATT descriptor.

Parameters **handle** (*int*) – The handle of the descriptor to read from.

Returns (bytearray) The read data.

set_disconnected_callback(*callback: Optional[Callable[BaseBleakClient, None]], **kwargs*) → None
Set the disconnect callback. The callback will only be called on unsolicited disconnect event.

Callbacks must accept one input which is the client object itself.

Set the callback to `None` to remove any existing callback.

```
def callback(client):
    print("Client with address {} got disconnected!".format(client.address))

client.set_disconnected_callback(callback)
client.connect()
```

Parameters **callback** – callback to be called on disconnection.

start_notify(*char_specifier: Union[bleak.backends.characteristic.BleakGATTCharacteristic, int, str, uuid.UUID], callback: Callable[[int, bytearray], None], **kwargs*) → None
Activate notifications/indications on a characteristic.

Callbacks must accept two inputs. The first will be a integer handle of the characteristic generating the data and the second will be a `bytearray`.

```
def callback(sender: int, data: bytearray):
    print(f"{sender}: {data}")
client.start_notify(char_uuid, callback)
```

Parameters

- **char_specifier** (*BleakGATTCharacteristic, int, str or UUID*) – The characteristic to activate notifications/indications on a characteristic, specified by either integer handle, UUID or directly by the BleakGATTCharacteristic object representing it.

- **callback** (*function*) – The function to be called on notification.

stop_notify(*char_specifier: Union[bleak.backends.characteristic.BleakGATTCharacteristic, int, str, uuid.UUID]*) → None
Deactivate notification/indication on a specified characteristic.

Parameters **char_specifier** (BleakGATTCharacteristic, *int, str or UUID*) – The characteristic to deactivate notification/indication on, specified by either integer handle, UUID or directly by the BleakGATTCharacteristic object representing it.

unpair() → bool
Unpair with the peripheral.

write_gatt_char(*char_specifier: Union[bleak.backends.characteristic.BleakGATTCharacteristic, int, str, uuid.UUID], data: Union[bytes, bytearray, memoryview], response: bool = False*) → None
Perform a write operation on the specified GATT characteristic.

Parameters

- **char_specifier** (BleakGATTCharacteristic, *int, str or UUID*) – The characteristic to write to, specified by either integer handle, UUID or directly by the BleakGATTCharacteristic object representing it.
- **data** (*bytes or bytearray*) – The data to send.
- **response** (*bool*) – If write-with-response operation should be done. Defaults to *False*.

write_gatt_descriptor(*handle: int, data: Union[bytes, bytearray, memoryview]*) → None
Perform a write operation on the specified GATT descriptor.

Parameters

- **handle** (*int*) – The handle of the descriptor to read from.
- **data** (*bytes or bytearray*) – The data to send.

Windows

BLE Client for Windows 10 systems.

Created on 2017-12-05 by hbldh <henrik.blidh@nedomkull.com>

class bleak.backends.dotnet.client.**BleakClientDotNet**(*address_or_ble_device: Union[bleak.backends.device.BLEDevice, str], **kwargs*)
The native Windows Bleak Client.

Implemented using pythonnet, a package that provides an integration to the .NET Common Language Runtime (CLR). Therefore, much of the code below has a distinct C# feel.

Parameters **address_or_ble_device** (*BLEDevice* or str) – The Bluetooth address of the BLE peripheral to connect to or the *BLEDevice* object representing it.

Keyword Arguments

- **use_cached** (*bool*) – If set to *True*, then the OS level BLE cache is used for getting services, characteristics and descriptors. Defaults to `True`.
- **timeout** (*float*) – Timeout for required `BleakScanner.find_device_by_address` call. Defaults to 10.0.

connect(***kwargs*) → bool
Connect to the specified GATT server.

Keyword Arguments

- **timeout** (*float*) – Timeout for required `BleakScanner.find_device_by_address` call. Defaults to 10.0.
- **use_cached** (*bool*) – If set to *True*, then the OS level BLE cache is used for getting services, characteristics and descriptors. Defaults to `True`.

Returns Boolean representing connection status.

Raises

- `BleakError` – When device is not found.
- `TimeoutError` – When connecting to the device takes too long.

disconnect() → bool
Disconnect from the specified GATT server.

Returns Boolean representing if device is disconnected.

Raises `asyncio.TimeoutError` – If device did not disconnect with 10 seconds.

get_services(***kwargs*) → bleak.backends.service.BleakGATTServiceCollection
Get all services registered for this GATT server.

Keyword Arguments **use_cached** (*bool*) – If set to *True*, then the OS level BLE cache is used for getting services, characteristics and descriptors.

Returns A *bleak.backends.service.BleakGATTServiceCollection* with this device's services tree.

is_connected
Check connection status between this client and the server.

Returns Boolean representing connection status.

mtu_size
Get ATT MTU size for active connection

pair(*protection_level=None*, ***kwargs*) → bool
Attempts to pair with the device.

Keyword Arguments **protection_level** –

Windows.Devices.Enumeration.DevicePairingProtectionLevel 1: None - Pair the device using no levels of protection. 2: Encryption - Pair the device using encryption. 3: EncryptionAndAuthentication - Pair the device using

encryption and authentication. (This will not work in Bleak...)

Returns Boolean regarding success of pairing.

read_gatt_char(*char_specifier: Union[bleak.backends.characteristic.BleakGATTCharacteristic, int, str, uuid.UUID]*, ***kwargs*) → bytearray
Perform read operation on the specified GATT characteristic.

Parameters **char_specifier** (*BleakGATTCharacteristic, int, str or UUID*) – The characteristic to read from, specified by either integer handle, UUID or directly by the BleakGATTCharacteristic object representing it.

Keyword Arguments **use_cached** (*bool*) – `False` forces Windows to read the value from the device again and not use its own cached value. Defaults to `False`.

Returns (bytearray) The read data.

read_gatt_descriptor(*handle: int*, ***kwargs*) → bytearray
Perform read operation on the specified GATT descriptor.

Parameters **handle** (*int*) – The handle of the descriptor to read from.

Keyword Arguments **use_cached** (*bool*) – `False` forces Windows to read the value from the device again and not use its own cached value. Defaults to `False`.

Returns (bytearray) The read data.

start_notify(*char_specifier: Union[bleak.backends.characteristic.BleakGATTCharacteristic, int, str, uuid.UUID], callback: Callable[[int, bytearray], None], **kwargs*) → None
Activate notifications/indications on a characteristic.

Callbacks must accept two inputs. The first will be a integer handle of the characteristic generating the data and the second will be a `bytearray` containing the data sent from the connected server.

```
def callback(sender: int, data: bytearray):
    print(f"{sender}: {data}")
client.start_notify(char_uuid, callback)
```

Parameters

- **char_specifier** (*BleakGATTCharacteristic, int, str or UUID*) – The characteristic to activate notifications/indications on a characteristic, specified by either integer handle, UUID or directly by the BleakGATTCharacteristic object representing it.
- **callback** (*function*) – The function to be called on notification.

stop_notify(*char_specifier: Union[bleak.backends.characteristic.BleakGATTCharacteristic, int, str, uuid.UUID]*) → None
Deactivate notification/indication on a specified characteristic.

Parameters **char_specifier** (*BleakGATTCharacteristic, int, str or UUID*) – The characteristic to deactivate notification/indication on, specified by either integer handle, UUID or directly by the BleakGATTCharacteristic object representing it.

unpair() → bool
Attempts to unpair from the device.

N.B. unpairing also leads to disconnection in the Windows backend.

Returns Boolean on whether the unparing was successful.

write_gatt_char(*char_specifier: Union[bleak.backends.characteristic.BleakGATTCharacteristic, int, str, uuid.UUID], data: Union[bytes, bytearray, memoryview], response: bool = False*) → None
Perform a write operation of the specified GATT characteristic.

Parameters

- **char_specifier** (*BleakGATTCharacteristic, int, str or UUID*) – The characteristic to write to, specified by either integer handle, UUID or directly by the BleakGATTCharacteristic object representing it.
- **data** (*bytes or bytearray*) – The data to send.
- **response** (*bool*) – If write-with-response operation should be done. Defaults to *False*.

write_gatt_descriptor(*handle: int, data: Union[bytes, bytearray, memoryview]*) → None
Perform a write operation on the specified GATT descriptor.

Parameters

- **handle** (*int*) – The handle of the descriptor to read from.
- **data** (*bytes or bytearray*) – The data to send.

macOS

BLE Client for CoreBluetooth on macOS

Created on 2019-06-26 by kevincar <kevincarrolldavis@gmail.com>

class bleak.backends.corebluetooth.client.**BleakClientCoreBluetooth**(*address_or_ble_device: Union[bleak.backends.device.BLE str], **kwargs*)

CoreBluetooth class interface for BleakClient

Parameters **address_or_ble_device** (*BLEDevice* or str) – The Bluetooth address of the BLE peripheral to connect to or the *BLEDevice* object representing it.

Keyword Arguments **timeout** (*float*) – Timeout for required `BleakScanner.find_device_by_address` call. Defaults to 10.0.

connect(***kwargs*) → bool
Connect to a specified Peripheral

Keyword Arguments **timeout** (*float*) – Timeout for required `BleakScanner.find_device_by_address` call. Defaults to 10.0.

Returns Boolean representing connection status.

disconnect() → bool
Disconnect from the peripheral device

get_rssi() → int
To get RSSI value in dBm of the connected Peripheral

get_services(***kwargs*) → bleak.backends.service.BleakGATTServiceCollection
Get all services registered for this GATT server.

Returns A *bleak.backends.service.BleakGATTServiceCollection* with this device's services tree.

is_connected
Checks for current active connection

mtu_size
Get ATT MTU size for active connection

pair(**args*, ***kwargs*) → bool
Attempt to pair with a peripheral.

Note: This is not available on macOS since there is not explicit method to do a pairing, Instead the docs state that it "auto-pairs" when trying to read a characteristic that requires encryption, something Bleak cannot do apparently.

Reference:

- Apple Docs

- Stack Overflow post #1
- Stack Overflow post #2

Returns Boolean regarding success of pairing.

read_gatt_char(*char_specifier: Union[bleak.backends.characteristic.BleakGATTCharacteristic, int, str, uuid.UUID], use_cached=False, **kwargs*) → bytearray

Perform read operation on the specified GATT characteristic.

Parameters

- **char_specifier** (*BleakGATTCharacteristic, int, str or UUID*) – The characteristic to read from, specified by either integer handle, UUID or directly by the BleakGATTCharacteristic object representing it.
- **use_cached** (*bool*) – *False* forces macOS to read the value from the device again and not use its own cached value. Defaults to *False*.

Returns (bytearray) The read data.

read_gatt_descriptor(*handle: int, use_cached=False, **kwargs*) → bytearray

Perform read operation on the specified GATT descriptor.

Parameters

- **handle** (*int*) – The handle of the descriptor to read from.
- **use_cached** (*bool*) – *False* forces Windows to read the value from the device again and not use its own cached value. Defaults to *False*.

Returns (bytearray) The read data.

start_notify(*char_specifier: Union[bleak.backends.characteristic.BleakGATTCharacteristic, int, str, uuid.UUID], callback: Callable[[int, bytearray], None], **kwargs*) → None

Activate notifications/indications on a characteristic.

Callbacks must accept two inputs. The first will be a integer handle of the characteristic generating the data and the second will be a `bytearray` containing the data sent from the connected server.

```
def callback(sender: int, data: bytearray):
    print(f"{sender}: {data}")
client.start_notify(char_uuid, callback)
```

Parameters

- **char_specifier** (*BleakGATTCharacteristic, int, str or UUID*) – The characteristic to activate notifications/indications on a characteristic, specified by either integer handle, UUID or directly by the BleakGATTCharacteristic object representing it.
- **callback** (*function*) – The function to be called on notification.

stop_notify(*char_specifier: Union[bleak.backends.characteristic.BleakGATTCharacteristic, int, str, uuid.UUID]*) → None

Deactivate notification/indication on a specified characteristic.

Parameters **char_specifier** (*BleakGATTCharacteristic, int, str or UUID*) – The characteristic to deactivate notification/indication on, specified by either integer handle, UUID or directly by the BleakGATTCharacteristic object representing it.

unpair() → bool
Returns:

write_gatt_char(*char_specifier: Union[bleak.backends.characteristic.BleakGATTCharacteristic, int, str, uuid.UUID], data: Union[bytes, bytearray, memoryview], response: bool = False*) → None
Perform a write operation of the specified GATT characteristic.

Parameters

- **char_specifier** (BleakGATTCharacteristic, *int*, *str or UUID*) – The characteristic to write to, specified by either integer handle, UUID or directly by the BleakGATTCharacteristic object representing it.
- **data** (*bytes or bytearray*) – The data to send.
- **response** (*bool*) – If write-with-response operation should be done. Defaults to *False*.

write_gatt_descriptor(*handle: int, data: Union[bytes, bytearray, memoryview]*) → None
Perform a write operation on the specified GATT descriptor.

Parameters

- **handle** (*int*) – The handle of the descriptor to read from.
- **data** (*bytes or bytearray*) – The data to send.

Linux Distributions with BlueZ

BLE Client for BlueZ on Linux

class bleak.backends.bluezdbus.client.**BleakClientBlueZDBus**(*address_or_ble_device: Union[bleak.backends.device.BLEDevice, str], **kwargs*)

A native Linux Bleak Client

Implemented by using the BlueZ DBUS API.

Parameters **address_or_ble_device** (*BLEDevice* or str) – The Bluetooth address of the BLE peripheral to connect to or the *BLEDevice* object representing it.

Keyword Arguments

- **timeout** (*float*) – Timeout for required `BleakScanner.find_device_by_address` call. Defaults to 10.0.
- **disconnected_callback** (*callable*) – Callback that will be scheduled in the event loop when the client is disconnected. The callable must take one argument, which will be this client object.
- **adapter** (*str*) – Bluetooth adapter to use for discovery.

connect(***kwargs*) → bool
Connect to the specified GATT server.

Keyword Arguments **timeout** (*float*) – Timeout for required `BleakScanner.find_device_by_address` call. Defaults to 10.0.

Returns Boolean representing connection status.

Raises

- `BleakError` – If the device is already connected or if the device could not be found.
- `BleakDBusError` – If there was a D-Bus error

- `asyncio.TimeoutError` – If the connection timed out

disconnect() → bool
Disconnect from the specified GATT server.

Returns Boolean representing if device is disconnected.

Raises

- `BleakDBusError` – If there was a D-Bus error
- asyncio.TimeoutError if the device was not disconnected within 10 seconds

get_services(***kwargs*) → bleak.backends.service.BleakGATTServiceCollection
Get all services registered for this GATT server.

Returns A *`bleak.backends.service.BleakGATTServiceCollection`* with this device's services tree.

is_connected
Check connection status between this client and the server.

Returns Boolean representing connection status.

mtu_size
Get ATT MTU size for active connection

pair(**args*, ***kwargs*) → bool
Pair with the peripheral.

You can use ConnectDevice method if you already know the MAC address of the device. Else you need to StartDiscovery, Trust, Pair and Connect in sequence.

Returns Boolean regarding success of pairing.

read_gatt_char(*char_specifier: Union[bleak.backends.bluezdbus.characteristic.BleakGATTCharacteristicBlueZDBus, int, str, uuid.UUID], **kwargs*) → bytearray
Perform read operation on the specified GATT characteristic.

Parameters **char_specifier** (*`BleakGATTCharacteristicBlueZDBus, int, str or UUID`*) – The characteristic to read from, specified by either integer handle, UUID or directly by the BleakGATTCharacteristicBlueZDBus object representing it.

Returns (bytearray) The read data.

read_gatt_descriptor(*handle: int*, ***kwargs*) → bytearray
Perform read operation on the specified GATT descriptor.

Parameters **handle** (*`int`*) – The handle of the descriptor to read from.

Returns (bytearray) The read data.

start_notify(*char_specifier: Union[bleak.backends.bluezdbus.characteristic.BleakGATTCharacteristicBlueZDBus, int, str, uuid.UUID], callback: Callable[[int, bytearray], None], **kwargs*) → None
Activate notifications/indications on a characteristic.

Callbacks must accept two inputs. The first will be a integer handle of the characteristic generating the data and the second will be a `bytearray` containing the data sent from the connected server.

```
def callback(sender: int, data: bytearray):
    print(f"{sender}: {data}")
client.start_notify(char_uuid, callback)
```

Parameters

- **char_specifier** (*BleakGATTCharacteristicBlueZDBus, int, str or UUID*) – The characteristic to activate notifications/indications on a characteristic, specified by either integer handle, UUID or directly by the BleakGATTCharacteristicBlueZDBus object representing it.
- **callback** (*function*) – The function to be called on notification.

stop_notify(*char_specifier: Union[bleak.backends.bluezdbus.characteristic.BleakGATTCharacteristicBlueZDBus, int, str, uuid.UUID]*) → None
Deactivate notification/indication on a specified characteristic.

Parameters **char_specifier** (*BleakGATTCharacteristicBlueZDBus, int, str or UUID*) – The characteristic to deactivate notification/indication on, specified by either integer handle, UUID or directly by the BleakGATTCharacteristicBlueZDBus object representing it.

unpair() → bool
Unpair with the peripheral.

Returns Boolean regarding success of unpairing.

write_gatt_char(*char_specifier: Union[bleak.backends.bluezdbus.characteristic.BleakGATTCharacteristicBlueZDBus, int, str, uuid.UUID], data: Union[bytes, bytearray, memoryview], response: bool = False*) → None
Perform a write operation on the specified GATT characteristic.

Note: The version check below is for the "type" option to the "Characteristic.WriteValue" method that was added to Bluez in 5.51 Before that commit, `Characteristic.WriteValue` was only "Write with response". `Characteristic.AcquireWrite` was added in Bluez 5.46 which can be used to "Write without response", but for older versions of Bluez, it is not possible to "Write without response".

Parameters

- **char_specifier** (*BleakGATTCharacteristicBlueZDBus, int, str or UUID*) – The characteristic to write to, specified by either integer handle, UUID or directly by the BleakGATTCharacteristicBlueZDBus object representing it.
- **data** (*bytes or bytearray*) – The data to send.
- **response** (*bool*) – If write-with-response operation should be done. Defaults to *False*.

write_gatt_descriptor(*handle: int, data: Union[bytes, bytearray, memoryview]*) → None
Perform a write operation on the specified GATT descriptor.

Parameters

- **handle** (*int*) – The handle of the descriptor to read from.
- **data** (*bytes or bytearray*) – The data to send.

1.5.2 Scanning Clients

Interface

class bleak.backends.scanner.**AdvertisementData**(***kwargs*)
Wrapper around the advertisement data that each platform returns upon discovery

class bleak.backends.scanner.**BaseBleakScanner**(**args*, ***kwargs*)
Interface for Bleak Bluetooth LE Scanners

classmethod discover(*timeout=5.0*, ***kwargs*) → List[bleak.backends.device.BLEDevice]
Scan continuously for `timeout` seconds and return discovered devices.

Parameters **timeout** – Time to scan for.

Keyword Arguments ****kwargs** – Implementations might offer additional keyword arguments sent to the constructor of the BleakScanner class.

Returns:

discovered_devices
Gets the devices registered by the BleakScanner.

Returns A list of the devices that the scanner has discovered during the scanning.

classmethod find_device_by_address(*device_identifier: str*, *timeout: float = 10.0*, ***kwargs*) → Optional[bleak.backends.device.BLEDevice]
A convenience method for obtaining a `BLEDevice` object specified by Bluetooth address or (macOS) UUID address.

Parameters

- **device_identifier** (*str*) – The Bluetooth/UUID address of the Bluetooth peripheral sought.
- **timeout** (*float*) – Optional timeout to wait for detection of specified peripheral before giving up. Defaults to 10.0 seconds.

Keyword Arguments **adapter** (*str*) – Bluetooth adapter to use for discovery.

Returns The `BLEDevice` sought or `None` if not detected.

classmethod find_device_by_filter(*filterfunc: Callable[[bleak.backends.device.BLEDevice, bleak.backends.scanner.AdvertisementData], bool]*, *timeout: float = 10.0*, ***kwargs*) → Optional[bleak.backends.device.BLEDevice]
A convenience method for obtaining a `BLEDevice` object specified by a filter function.

Parameters

- **filterfunc** (*AdvertisementDataFilter*) – A function that is called for every BLEDevice found. It should return True only for the wanted device.
- **timeout** (*float*) – Optional timeout to wait for detection of specified peripheral before giving up. Defaults to 10.0 seconds.

Keyword Arguments **adapter** (*str*) – Bluetooth adapter to use for discovery.

Returns The `BLEDevice` sought or `None` if not detected.

get_discovered_devices() → List[bleak.backends.device.BLEDevice]
Gets the devices registered by the BleakScanner.

Deprecated since version 0.11.0: This method will be removed in a future version of Bleak. Use the *discovered_devices* property instead.

Returns A list of the devices that the scanner has discovered during the scanning.

register_detection_callback(*callback: Optional[Callable[[bleak.backends.device.BLEDevice, bleak.backends.scanner.AdvertisementData], Optional[Awaitable[None]]]]*) → None
Register a callback that is called when a device is discovered or has a property changed.

If another callback has already been registered, it will be replaced with `callback`. `None` can be used to remove the current callback.

The `callback` is a function or coroutine that takes two arguments: `BLEDevice` and *`AdvertisementData`*.

Parameters **`callback`** – A function, coroutine or `None`.

`set_scanning_filter`(***kwargs*)
Set scanning filter for the BleakScanner.

Parameters **`**kwargs`** – The filter details. This will differ a lot between backend implementations.

`start`()
Start scanning for devices

`stop`()
Stop scanning for devices

Windows

`class` `bleak.backends.dotnet.scanner.`**`BleakScannerDotNet`**(***kwargs*)
The native Windows Bleak BLE Scanner.

Implemented using pythonnet, a package that provides an integration to the .NET Common Language Runtime (CLR). Therefore, much of the code below has a distinct C# feel.

Keyword Arguments

- **`mode`** (*`scanning`*) – Set to `Passive` to avoid the `Active` scanning mode.
- **`SignalStrengthFilter`** (`Windows.Devices.Bluetooth.BluetoothSignalStrengthFilter`) – A BluetoothSignalStrengthFilter object used for configuration of Bluetooth LE advertisement filtering that uses signal strength-based filtering.
- **`AdvertisementFilter`** (`Windows.Devices.Bluetooth.Advertisement.BluetoothLEAdvertisementFilter`) – A BluetoothLEAdvertisementFilter object used for configuration of Bluetooth LE advertisement filtering that uses payload section-based filtering.

`discovered_devices`
Gets the devices registered by the BleakScanner.

Returns A list of the devices that the scanner has discovered during the scanning.

`set_scanning_filter`(***kwargs*)
Set a scanning filter for the BleakScanner.

Keyword Arguments

- **`SignalStrengthFilter`** (`Windows.Devices.Bluetooth.BluetoothSignalStrengthFilter`) – A BluetoothSignalStrengthFilter object used for configuration of Bluetooth LE advertisement filtering that uses signal strength-based filtering.
- **`AdvertisementFilter`** (*`Windows.Devices.Bluetooth.Advertisement.BluetoothLEAdvertisementFilter`*) – A BluetoothLEAdvertisementFilter object used for configuration of Bluetooth LE advertisement filtering that uses payload section-based filtering.

start()
Start scanning for devices

status
Get status of the Watcher.

Returns

Aborted 4 An error occurred during transition or scanning that stopped the watcher due to an error.

Created 0 The initial status of the watcher.

Started 1 The watcher is started.

Stopped 3 The watcher is stopped.

Stopping 2 The watcher stop command was issued.

stop()
Stop scanning for devices

macOS

class bleak.backends.corebluetooth.scanner.**BleakScannerCoreBluetooth**(***kwargs*)
The native macOS Bleak BLE Scanner.

Documentation: https://developer.apple.com/documentation/corebluetooth/cbcentralmanager

CoreBluetooth doesn't explicitly use Bluetooth addresses to identify peripheral devices because private devices may obscure their Bluetooth addresses. To cope with this, CoreBluetooth utilizes UUIDs for each peripheral. Bleak uses this for the BLEDevice address on macOS.

Keyword Arguments **timeout** (*double*) – The scanning timeout to be used, in case of missing stopScan_ method.

discovered_devices
Gets the devices registered by the BleakScanner.

Returns A list of the devices that the scanner has discovered during the scanning.

set_scanning_filter(***kwargs*)
Set scanning filter for the scanner.

Note: This is not implemented for macOS yet.

Raises NotImplementedError

start()
Start scanning for devices

stop()
Stop scanning for devices

Linux Distributions with BlueZ

class bleak.backends.bluezdbus.scanner.**BleakScannerBlueZDBus**(***kwargs*)
The native Linux Bleak BLE Scanner.

For possible values for *filters*, see the parameters to the `SetDiscoveryFilter` method in the BlueZ docs

Keyword Arguments

- **`adapter`** (*`str`*) – Bluetooth adapter to use for discovery.
- **`filters`** (*`dict`*) – A dict of filters to be applied on discovery.

`discovered_devices`
Gets the devices registered by the BleakScanner.

Returns A list of the devices that the scanner has discovered during the scanning.

`set_scanning_filter`(***kwargs*)
Sets OS level scanning filters for the BleakScanner.

For possible values for *filters*, see the parameters to the `SetDiscoveryFilter` method in the BlueZ docs

See variant types here: <https://python-dbus-next.readthedocs.io/en/latest/type-system/>

Keyword Arguments **`filters`** (*`dict`*) – A dict of filters to be applied on discovery.

`start`()
Start scanning for devices

`stop`()
Stop scanning for devices

1.5.3 Class representing BLE devices

Generated by `bleak.discover()` and `bleak.backends.scanning.BaseBleakScanner`.

Wrapper class for Bluetooth LE servers returned from calling `bleak.discover()`.

Created on 2018-04-23 by hbldh <henrik.blidh@nedomkull.com>

`class` `bleak.backends.device.`**`BLEDevice`**(*address*, *name*, *details=None*, *rssi=0*, ***kwargs*)
A simple wrapper class representing a BLE server detected during a *discover* call.

- When using Windows backend, *details* attribute is a `Windows.Devices.Bluetooth.Advertisement.BluetoothLEAdvertisement` object, unless it is created with the Windows.Devices.Enumeration discovery method, then is is a `Windows.Devices.Enumeration.DeviceInformation`.
- When using Linux backend, `details` attribute is a dict with keys `path` which has the string path to the DBus device object and `props` which houses the properties dictionary of the D-Bus Device.
- When using macOS backend, `details` attribute will be a CBPeripheral object.

`address = None`
The Bluetooth address of the device on this machine.

`details = None`
The OS native details required for connecting to the device.

`metadata = None`
Device specific details. Contains a `uuids` key which is a list of service UUIDs and a `manufacturer_data` field with a bytes-object from the advertised data.

`name = None`
The advertised name of the device.

rssi = None
RSSI, if available

1.5.4 GATT objects

Gatt Service Collection class and interface class for the Bleak representation of a GATT Service.

Created on 2019-03-19 by hbldh <henrik.blidh@nedomkull.com>

class bleak.backends.service.**BleakGATTService**(*obj*)
Interface for the Bleak representation of a GATT Service.

add_characteristic(*characteristic: bleak.backends.characteristic.BleakGATTCharacteristic*)
Add a `BleakGATTCharacteristic` to the service.

Should not be used by end user, but rather by *bleak* itself.

characteristics
List of characteristics for this service

description
String description for this service

get_characteristic(*uuid: Union[str, uuid.UUID]*) → Optional[bleak.backends.characteristic.BleakGATTCharacteristic]
Get a characteristic by UUID.

Parameters **uuid** – The UUID to match.

Returns The first characteristic matching `uuid` or `None` if no matching characteristic was found.

handle
The handle of this service

uuid
The UUID to this service

class bleak.backends.service.**BleakGATTServiceCollection**
Simple data container for storing the peripheral's service complement.

add_characteristic(*characteristic: bleak.backends.characteristic.BleakGATTCharacteristic*)
Add a `BleakGATTCharacteristic` to the service collection.

Should not be used by end user, but rather by *bleak* itself.

add_descriptor(*descriptor: bleak.backends.descriptor.BleakGATTDescriptor*)
Add a `BleakGATTDescriptor` to the service collection.

Should not be used by end user, but rather by *bleak* itself.

add_service(*service: bleak.backends.service.BleakGATTService*)
Add a *BleakGATTService* to the service collection.

Should not be used by end user, but rather by *bleak* itself.

characteristics
Returns dictionary of handles mapping to BleakGATTCharacteristic

descriptors
Returns a dictionary of integer handles mapping to BleakGATTDescriptor

get_characteristic(*specifier: Union[int, str, uuid.UUID]*) → bleak.backends.characteristic.BleakGATTCharacteristic
Get a characteristic by handle (int) or UUID (str or uuid.UUID)

get_descriptor(*handle: int*) → bleak.backends.descriptor.BleakGATTDescriptor
Get a descriptor by integer handle

get_service(*specifier: Union[int, str, uuid.UUID]*) → bleak.backends.service.BleakGATTService
Get a service by handle (int) or UUID (str or uuid.UUID)

services
Returns dictionary of handles mapping to BleakGATTService

Interface class for the Bleak representation of a GATT Characteristic

Created on 2019-03-19 by hbldh <henrik.blidh@nedomkull.com>

class bleak.backends.characteristic.**BleakGATTCharacteristic**(*obj: Any*)
Interface for the Bleak representation of a GATT Characteristic

add_descriptor(*descriptor: bleak.backends.descriptor.BleakGATTDescriptor*)
Add a `BleakGATTDescriptor` to the characteristic.

Should not be used by end user, but rather by *bleak* itself.

description
Description for this characteristic

descriptors
List of descriptors for this service

get_descriptor(*specifier: Union[int, str, uuid.UUID]*) → Optional[bleak.backends.descriptor.BleakGATTDescriptor]
Get a descriptor by handle (int) or UUID (str or uuid.UUID)

handle
The handle for this characteristic

properties
Properties of this characteristic

service_handle
The integer handle of the Service containing this characteristic

service_uuid
The UUID of the Service containing this characteristic

uuid
The UUID for this characteristic

class bleak.backends.characteristic.**GattCharacteristicsFlags**
An enumeration.

Interface class for the Bleak representation of a GATT Descriptor

Created on 2019-03-19 by hbldh <henrik.blidh@nedomkull.com>

class bleak.backends.descriptor.**BleakGATTDescriptor**(*obj: Any*)
Interface for the Bleak representation of a GATT Descriptor

characteristic_handle
handle for the characteristic that this descriptor belongs to

characteristic_uuid
UUID for the characteristic that this descriptor belongs to

description
A text description of what this descriptor represents

handle
Integer handle for this descriptor

uuid
UUID for this descriptor

1.5.5 Exceptions

exception bleak.exc.**BleakDBusError**(*dbus_error: str*, *error_body: list*)
Specialized exception type for D-Bus errors.

dbus_error
Gets the D-Bus error name, e.g. `org.freedesktop.DBus.Error.UnknownObject`.

dbus_error_details
Gets the optional D-Bus error details, e.g. 'Invalid UUID'.

exception bleak.exc.**BleakDotNetTaskError**
Wrapped exception that occurred in .NET async Task.

exception bleak.exc.**BleakError**
Base Exception for bleak.

1.5.6 Utilities

bleak.utils.**mac_int_2_str**(*mac*)
Convert integer Bluetooth address to colon separated hex string.

Parameters **mac** (*int*) – A positive integer.

Returns Bluetooth address as colon separated hex string.

bleak.utils.**mac_str_2_int**(*mac*)
Convert colon separated hex string Bluetooth address to integer.

Parameters **mac** (*str*) – A colon separated hex string Bluetooth address.

Returns Bluetooth address as integer.

1.6 Troubleshooting

When things don't seem to be working right, here are some things to try.

1.6.1 Enable Logging

The easiest way to enable logging is to set the `BLEAK_LOGGING` environment variable. Setting the variable depends on what type of terminal you are using.

Posix (Linux, macOS, Cygwin, etc.):

```
export BLEAK_LOGGING=1
```

Power Shell:

```
$env:BLEAK_LOGGING=1
```

Windows Command Prompt:

```
set BLEAK_LOGGING=1
```

Then run your Python script in the same terminal.

1.6.2 Capture Bluetooth Traffic

Sometimes it can be helpful to see what is actually going over the air between the OS and the Bluetooth device. There are tools available to capture HCI packets and decode them.

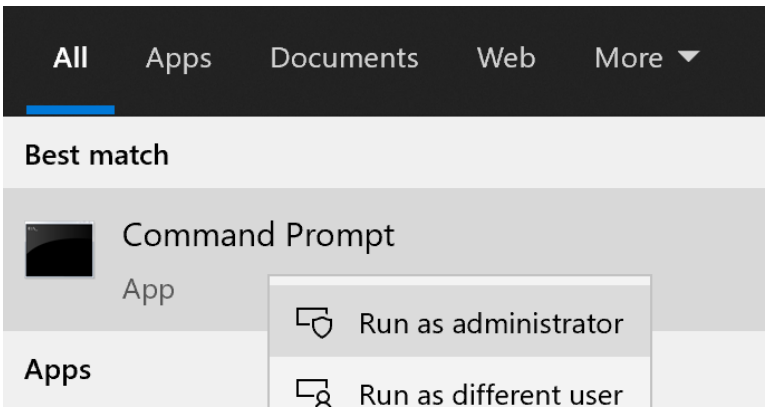
Windows 10

No special software is required on Windows to capture Bluetooth traffic, however special software is required to convert it to a useful format.

Capture

To capture Bluetooth traffic:

1. Open a Command Prompt as Administrator.
   - Search start menu for `cmd`.
   - Right-click *Command Prompt* and select *Run as Administrator*.



2. Run the following command in the Administrator Command Prompt:

```
logman create trace "bth_hci" -ow -o C:\bth_hci.etl -p {8a1f9517-3a8c-4a9e-a018-
↪4f17a200f277} 0xffffffffffffffff 0xff -nb 16 16 -bs 1024 -mode Circular -f␣
↪bincirc -max 4096 -ets
```

> **Tip:** `C:\bth_hci.etl` can be replaced with any file path you like.

3. Run your Python script in a different terminal (not as Administrator) to reproduce the problem.
4. In the Administrator Command Prompt run:

```
logman stop "bth_hci" -ets
```

Decode

Microsoft no longer has tools to directly view `.etl` files so in order to make use of the information, we need to convert it to a different file format. The Windows Driver Kit contains a tool to do this.

1. Download and install the Windows Driver Kit.

   Tip: The install may give warnings about additional software not being installed. These warnings can be ignored since we just need a standalone executable file from the installation.

2. Run the following command:

   ```
   "%ProgramFiles(x86)%\Windows Kits\10\Tools\x86\Bluetooth\BETLParse\btetlparse.exe
   ↪" c:\bth_hci.etl
   ```

   This will create a file with the same file name and a `.cfa` file extension (and an empty `.txt` file for some reason).

3. Download and install Wireshark.

4. Open the `.cfa` file in Wireshark to view the captured Bluetooth traffic.

macOS

On macOS, special software is required to capture and view Bluetooth traffic. You will need to sign up for an Apple Developer account to obtain this software.

1. Go to https://developer.apple.com/download/more/ and download *Additional Tools for Xcode ...* where ... is the Xcode version corresponding to your macOS version (e.g. 12 for Big Sur, 11 for Mojave, etc.).
2. Open the disk image and in the *Hardware* folder, double-click the *PacketLogger.app* to run it.
3. Click the *Clear* button in the toolbar to clear the old data.
4. Run your Python script to reproduce the problem.
5. Click the *Stop* button in the toolbar to stop the capture.

Tip: The Bluetooth traffic can be viewed in the *PacketLogger.app* or it can be saved to a file and viewed in Wireshark.

Linux

On Linux, Wireshark can be used to capture and view Bluetooth traffic.

1. Install Wireshark. Most distributions include a `wireshark` package. For example, on Debian/Ubuntu based distributions:

   ```
   sudo apt update && sudo apt install wireshark
   ```

2. Start Wireshark and select your Bluetooth adapter, then start a capture.

   Tip: Visit the Wireshark Wiki for help with configuring permissions and making sure proper drivers are installed.

3. Run your Python script to reproduce the problem.
4. Click the stop button in Wireshark to stop the capture.

1.6.3 Handling OS Caching of BLE Device Services

If you develop your own BLE peripherals, and frequently change services, characteristics and/or descriptors, then Bleak might report outdated versions of your peripheral's services due to OS level caching. The caching is done to speed up the connections with peripherals where services do not change and is enabled by default on most operating systems and thus also in Bleak.

There are ways to avoid this on different backends though, and if you experience these kinds of problems, the steps below might help you to circumvent the caches.

Windows 10

The Windows .NET backend has the most straightforward means of handling the os caches. When creating a BleakClient, one can use the keyword argument *use_cached*:

```
async with BleakClient(address, use_cached=False) as client:
    print(f"Connected: {client.is_connected}")
    // Do whatever it is you want to do.
```

The keyword argument is also present in the `bleak.backends.client.BleakClient.connect()` method to use if you don't want to use the async context manager:

```
client = BleakClient(address)
await client.connect(use_cached=True)
print(f"Connected: {client.is_connected}")
// Do whatever it is you want to do.
await client.disconnect()
```

macOS

The OS level caching handling on macOS has not been explored yet.

Linux

When you change the structure of services/characteristics on a device, you have to remove the device from BlueZ so that it will read everything again. Otherwise BlueZ gives the cached values from the first time the device was connected. You can use the `bluetoothctl` command line tool to do this:

```
bluetoothctl -- remove [mac_address]
```

1.7 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

1.7.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/hbldh/bleak/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

Write Documentation

bleak could always use more documentation, whether as part of the official bleak docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/hbldh/bleak/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

1.7.2 Get Started!

Ready to contribute? Here's how to set up *bleak* for local development.

1. Fork the *bleak* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/bleak.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv bleak
$ cd bleak/
$ python setup.py develop
```

4. Create a branch for local development, originating from the *develop* branch:

```
$ git checkout -b name-of-your-bugfix-or-feature develop
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 bleak tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

1.7.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. If the pull request adds functionality, the docs should be updated.
2. Modify the `CHANGELOG.rst`, describing your changes as is specified by the guidelines in that document.
3. **The pull request should work for Python 3.6+ on the following platforms:**
   - Windows 10, version 16299 (Fall Creators Update) and greater
   - Linux distributions with BlueZ >= 5.43
   - OS X / macOS >= 10.11
4. Squash all your commits on your PR branch, if the commits are not solving different problems and you are committing them in the same PR. In that case, consider making several PRs instead.
5. Feel free to add your name as a contributor to the `AUTHORS.rst` file!

1.8 Credits

1.8.1 Development Lead

- Henrik Blidh <henrik.blidh@nedomkull.com>

1.8.2 Development Team / Collaborators

- David Lechner <david@pybricks.com>

1.8.3 Contributors

- Chad Spensky <chad@allthenticate.net>
- Bernie Conrad <bernie@allthenticate.net>
- Jonathan Soto <jsotogaviard@alum.mit.edu>

1.9 Changelog

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

1.9.1 0.12.1 (2021-07-07)

Changed

- Changed minimum `winrt` package version to 1.0.21033.1. Fixes #589.

Fixed

- Fixed unawaited future when writing without response on CoreBluetooth backend. Fixes #586.

1.9.2 0.12.0 (2021-06-19)

Added

- Added `mtu_size` property for clients.
- Added WinRT backend.
- Added `BleakScanner.discovered_devices` property.
- Added an event to await when stopping scanners in WinRT and pythonnet backends. Fixes #556.
- Added `BleakScanner.find_device_by_filter` static method.
- Added `scanner_byname.py` example.
- Added optional command line argument to specify device to all applicable examples.

Changed

- Added `Programming Language :: Python :: 3.9` classifier in `setup.py`.
- Deprecated `BleakScanner.get_discovered_devices()` async method.
- Added capability to handle async functions as detection callbacks in `BleakScanner`.

- Added error description in addition to error name when `BleakDBusError` is converted to string.
- Change typing of data parameter in write methods to `Union[bytes, bytearray, memoryview]`.
- Improved type hints in CoreBluetooth backend.
- Use delegate callbacks for `get_rssi()` on CoreBluetooth backend.
- Use `@objc.python_method` where possible in `PeripheralDelegate` class.
- Using ObjC key-value observer to wait for `BleakScanner.start()` and `stop()` in CoreBluetooth backend.

Fixed

- Fixed `KeyError` when trying to connect to `BLEDevice` from advertising data callback on macOS. Fixes #448.
- Handling of undetected devices in `connect_by_bledevice.py` example. Fixes #487.
- Added `Optional` typehint for `BleakScanner.find_device_by_address`.
- Fixed `linux_autodoc_mock_import` in `docs/conf.py`.
- Minor fix for disconnection event handling in BlueZ backend. Fixes #491.
- Corrections for the Philips Hue lamp example. Merged #505.
- Fixed `BleakClientBlueZDBus.pair()` method always returning `True`. Fixes #503.
- Fixed waiting for notification start/stop to complete in CoreBluetooth backend.
- Fixed write without response on BlueZ < 5.51.
- Fixed error propagation for CoreBluetooth events.
- Fixed failed import on CI server when BlueZ is not installed.
- Fixed notification `value` should be `bytearray` on CoreBluetooth. Fixes #560.
- Fixed crash when cancelling connection when Python runtime shuts down on CoreBluetooth backend. Fixes #538.
- Fixed connecting to multiple devices using a single `BleakScanner` on CoreBluetooth backend.
- Fixed deadlock in CoreBluetooth backend when device disconnects while callbacks are pending. Fixes #535.
- Fixed deadlock when using more than one service, characteristic or descriptor with the same UUID on CoreBluetooth backend.
- Fixed exception raised when calling `BleakScanner.stop()` when already stopped in CoreBluetooth backend.

1.9.3 0.11.0 (2021-03-17)

Added

- Updated `dotnet.client.BleakClientDotNet` connect method docstring.
- Added `AdvertisementServiceData` in BLEDevice in macOS devices
- Protection levels (encryption) in Windows backend pairing. Solves #405.
- Philips Hue lamp example script. Relates to #405.

- Keyword arguments to `get_services` method on `BleakClient`.
- Keyword argument `use_cached` on .NET backend, to enable uncached reading of services, characteristics and descriptors in Windows.
- Documentation on troubleshooting OS level caches for services.
- New example added: Async callbacks with a queue and external consumer
- `handle` property on `BleakGATTService` objects
- `service_handle` property on `BleakGATTCharacteristic` objects
- Added more specific type hints for `BleakGATTServiceCollection` properties.
- Added `asyncio` task to disconnect devices on event loop crash in BlueZ backend.
- Added filtering on advertisement data callbacks on BlueZ backend so that callbacks only occur when advertising data changes like on macOS backend.
- Added fallback to try `org.bluez.Adapter1.ConnectDevice` when trying to connect a device in BlueZ backend.
- Added UART service example.

Fixed

- Fixed wrong OS write method called in `write_gatt_descriptor()` in Windows backend. Merged #403.
- Fixed `BaseBleakClient.services_resolved` not reset on disconnect on BlueZ backend. Merged #401.
- Fixed RSSI missing in discovered devices on macOS backend. Merged #400.
- Fixed scan result shows 'Unknown' name of the `BLEDevice`. Fixes #371.
- Fixed a broken check for the correct adapter in `BleakClientBlueZDBus`.
- Fixed #445 and #362 for Windows.

Changed

- Using handles to identify the services. Added *handle* abstract property to *BleakGATTService* and storing the services by handle instead of UUID.
- Changed `BleakScanner.set_scanning_filter()` from async method to normal method.
- Changed BlueZ backend to use `dbus-next` instead of `txdbus`.
- Changed `BleakClient.is_connected` from async method to property.
- Consolidated D-Bus signal debug messages in BlueZ backend.

Removed

- Removed all `__str__` methods from backend service, characteristic and descriptor implementations in favour of those in the abstract base classes.

1.9.4 0.10.0 (2020-12-11)

Added

- Added `AdvertisementData` class used with detection callbacks across all supported platforms. Merged #334.
- Added `BleakError` raised during import on unsupported platforms.
- Added `rssi` parameter to `BLEDevice` constructor.
- Added `detection_callback` kwarg to `BleakScanner` constructor.

Changed

- Updated minimum PyObjC version to 7.0.1.
- Consolidated implementation of `BleakScanner.register_detection_callback()`. All platforms now take callback with `BLEDevice` and `AdvertisementData` arguments.
- Consolidated `BleakScanner.find_device_by_address()` implementations.
- Renamed "device" kwarg to "adapter" in BleakClient and BleakScanner. Fixes #381.

Fixed

- Fixed use of bare exceptions.
- Fixed `BleakClientBlueZDBus.start_notify()` misses initial notifications with fast Bluetooth devices. Fixed #374.
- Fix event callbacks on Windows not running in asyncio event loop thread.
- Fixed `BleakScanner.discover()` on older versions of macOS. Fixes #331.
- Fixed disconnect callback on BlueZ backend.
- Fixed calling `BleakClient.is_connected()` on Mac before connection.
- Fixed kwargs ignored in `BleakScanner.find_device_by_address()` in BlueZ backend. Fixes #360.

Removed

- Removed duplicate definition of `BLEDevice` in BlueZ backend.
- Removed unused imports.
- Removed separate implementation of global `discover` method.

1.9.5 0.9.1 (2020-10-22)

Added

- Added new attribute `_device_info` on `BleakClientBlueZDBus`. Merges #347.
- Added Pull Request Template.

Changed

- Updated instructions on how to contribute, file issues and make PRs.
- Updated `AUTHORS.rst` file with development team.

Fixed

- Fix well-known services not converted to UUIDs in `BLEDevice.metadata` in CoreBluetooth backend. Fixes #342.
- Fix advertising data replaced instead of merged in scanner in CoreBluetooth backend. Merged #343.
- Fix CBCentralManager not properly waited for during initialization in some cases.
- Fix AttributeError in CoreBluetooth when using BLEDeviceCoreBluetooth object.

1.9.6 0.9.0 (2020-10-20)

Added

- Timeout for BlueZ backend connect call to avoid potential infinite hanging. Merged #306.
- Added Interfaces API docs again.
- Troubleshooting documentation.
- noqa flags added to `BleakBridge` imports.
- Adding a timeout on OSX so that the connect cannot hang forever. Merge #336.

Changed

- `BleakCharacteristic.description()` on .NET now returns the same value as other platforms.
- Changed all adding and removal of .NET event handler from `+=/-=` syntax to calling `add_` and `remove_` methods instead. This allows for proper removal of event handlers in .NET backend.
- All code dependence on the `BleakBridge` is now removed. It is only imported to allow for access to UWP namespaces.
- Removing internal method `_start_notify` in the .NET backend.
- `GattSession` object now manages lifetime of .NET `BleakClient` connection.
- `BleakClient` in .NET backend will reuse previous device information when reconnecting so that it doesn't have to scan/discover again.

Fixed

- UUID property bug fixed in BlueZ backend. Merged #307.
- Fix for broken RTD documentation.
- Fix UUID string arguments should not be case sensitive.
- Fix `BleakGATTService.get_characteristic()` method overridden with `NotImplementedError` in BlueZ backend.

- Fix `AttributeError` when trying to connect using CoreBluetooth backend. Merged #323.
- Fix disconnect callback called multiple times in .NET backend. Fixes #312.
- Fix `BleakClient.disconnect()` method failing when called multiple times in .NET backend. Fixes #313.
- Fix `BleakClient.disconnect()` method failing when called multiple times in Core Bluetooth backend. Merge #333.
- Catch RemoteError in `is_connected` in BlueZ backend. Fixes #310,
- Prevent overwriting address in constructor of `BleakClient` in BlueZ backend. Merge #311.
- Fix nordic uart UUID. Merge #339.

1.9.7 0.8.0 (2020-09-22)

Added

- Implemented `set_disconnected_callback` in the .NET backend `BleakClient` implementation.
- Added `find_device_by_address` method to the `BleakScanner` interface, for stopping scanning when a desired address is found.
- Implemented `find_device_by_address` in the .NET backend `BleakScanner` implementation and switched its `BleakClient` implementation to use that method in `connect`.
- Implemented `find_device_by_address` in the BlueZ backend `BleakScanner` implementation and switched its `BleakClient` implementation to use that method in `connect`.
- Implemented `find_device_by_address` in the Core Bluetooth backend `BleakScanner` implementation and switched its `BleakClient` implementation to use that method in `connect`.
- Added text representations of Protocol Errors that are visible in the .NET backend. Added these texts to errors raised.
- Added pairing method in `BleakClient` interface.
- Implemented pairing method in .NET backend.
- Implemented pairing method in the BlueZ backend.
- Added stumps and `NotImplementedError` on pairing in macOS backend.
- Added the possibility to connect using `BLEDevice` instead of a string address. This allows for skipping the discovery call when connecting.

Removed

- Support for Python 3.5.

Changed

- **BREAKING CHANGE** All notifications now have the characteristic's integer **handle** instead of its UUID as a string as the first argument `sender` sent to notification callbacks. This provides the uniqueness of sender in notifications as well.
- Renamed `BleakClient` argument `address` to `address_or_ble_device`.

- Version 0.5.0 of BleakUWPBridge, with some modified methods and implementing `IDisposable`.
- Merged #224. All storing and passing of event loops in bleak is removed.
- Removed Objective C delegate compliance checks. Merged #253.
- Made context managers for .NET `DataReader` and `DataWriter`.

Fixed

- .NET backend loop handling bug entered by #224 fixed.
- Removed default `DEBUG` level set to bleak logger. Fixes #251.
- More coherency in logger uses over all backends. Fixes #258
- Attempted fix of #255 and #133: cleanups, disposing of objects and creating new `BleakBridge` instances each disconnect.
- Fixed some type hints and docstrings.
- Modified the `connected_peripheral_delegate` handling in macOS backend to fix #213 and #116.
- Merged #270, fixing a critical bug in `get_services` method in Core Bluetooth backend.
- Improved handling of disconnections and `is_connected` in BlueZ backend to fix #259.
- Fix for `set_disconnected_callback` on Core Bluetooth. Fixes #276.
- Safer *Core Bluetooth* presence check. Merged #280.

1.9.8 0.7.1 (2020-07-02)

Changed

- Improved, more explanatory error on BlueZ backend when `BleakClient` cannot find the desired device when trying to connect. (#238)
- Better-than-nothing documentation about scanning filters added (#230).
- Ran black on code which was forgotten in 0.7.0. Large diffs due to that.
- Re-adding Python 3.8 CI "tests" on Windows again.

Fixed

- Fix when characteristic updates value faster than asyncio schedule (#240 & #241)
- Incorrect `MANIFEST.in` corrected. (#244)

1.9.9 0.7.0 (2020-06-30)

Added

- Better feedback of communication errors to user in .NET backend and implementing error details proposed in #174.
- Two devices example file to use for e.g. debugging.

- Detection/discovery callbacks in Core Bluetooth backend `Scanner` implemented.
- Characteristic handle printout in `service_explorer.py`.
- Added scanning filters to .NET backend's `discover` method.

Changed

- Replace `NSRunLoop` with dispatch queue in Core Bluetooth backend. This causes callbacks to be dispatched on a background thread instead of on the main dispatch queue on the main thread. `call_soon_threadsafe()` is used to synchronize the events with the event loop where the central manager was created. Fixes #111.
- The Central Manager is no longer global in the Core Bluetooth backend. A new one is created for each `BleakClient` and `BleakScanner`. Fixes #206 and #105.
- Merged #167 and reworked characteristics handling in Bleak. Implemented in all backends; bleak now uses the characteristics' handle to identify and keep track of them. Fixes #139 and #159 and allows connection for devices with multiple instances of the same characteristic UUIDs.
- In `requirements.txt` and `Pipfile`, the requirement on `pythonnet` was bumped to version 2.5.1, which seems to solve issues described in #217 and #225.
- Renamed `HISTORY.rst` to `CHANGELOG.rst` and adopted the Keep a Changelog format.
- Python 3.5 support from macOS is officially removed since pyobjc>6 requires 3.6+
- Pin `pyobjc` dependencies to use at least version 6.2. (PR #194)
- Pin development requirement on *bump2version* to version 1.0.0
- Added `.pyup.yml` for Pyup
- Using CBManagerState constants from pyobj instead of integers.

Removed

- Removed documentation note about not using new event loops in Linux. This was fixed by #143.
- `_central_manager_delegate_ready` was removed in macOS backend.
- Removed the `bleak.backends.bluez.utils.get_gatt_service_path` method. It is not used by bleak and possibly generates errors.

Fixed

- Improved handling of the txdbus connection to avoid hanging of disconnection clients in BlueZ backend. Fixes #216, #219 & #221.
- #150 hints at the device path not being possible to create as is done in the *get_device_object_path* method. Now, we try to get it from BlueZ first. Otherwise, use the old fallback.
- Minor documentation errors corrected.
- `CBManagerStatePoweredOn` is now properly handled in Core Bluetooth.
- Device enumeration in `discover``and ``Scanner` corrected. Fixes #211
- Updated documentation about scanning filters.
- Added workaround for `isScanning` attribute added in macOS 10.13. Fixes #234.

1.9.10 0.6.4 (2020-05-20)

Fixed

- Fix for bumpversion usage

1.9.11 0.6.3 (2020-05-20)

Added

- Building and releasing from Github Actions

Removed

- Building and releasing on Azure Pipelines

1.9.12 0.6.2 (2020-05-15)

Added

- Added `disconnection_callback` functionality for Core Bluetooth (#184 & #186)
- Added `requirements.txt`

Fixed

- Better cleanup of Bluez notifications (#154)
- Fix for `read_gatt_char` in Core Bluetooth (#177)
- Fix for `is_disconnected` in Core Bluetooth (#187 & #185)
- Documentation fixes

1.9.13 0.6.1 (2020-03-09)

Fixed

- Including #156, lost notifications on macOS backend, which was accidentally missed on previous release.

1.9.14 0.6.0 (2020-03-09)

- New Scanner object to allow for async device scanning.
- Updated `txdbus` requirement to version 1.1.1 (Merged #122)
- Implemented `write_gatt_descriptor` for Bluez backend.
- Large change in Bluez backend handling of Twisted reactors. Fixes #143
- Modified `set_disconnect_callback` to actually call the callback as a callback. Fixes #108.
- Added another required parameter to disconnect callbacks.

- Added Discovery filter option in BlueZ backend (Merged #124)
- Merge #138: comments about Bluez version check.
- Improved scanning data for macOS backend. Merge #126.
- Merges #141, a critical fix for macOS.
- Fix for #114, write with response on macOS.
- Fix for #87, DIctionary changes size on .NET backend.
- Fix for #127, uuid or str on macOS.
- Handles str/uuid for characteristics better.
- Merge #148, Run .NET backend notifications on event loop instead of main loop.
- Merge #146, adapt characteristic write log to account for WriteWithoutResponse on macOS.
- Fix for #145, Error in cleanup on Bluez backend.
- Fix for #151, only subscribe to BlueZ messages on DBus. Merge #152.
- Fix for #142, Merge #144, Improved scanning for macOS backend.
- Fix for #155, Merge #156, lost notifications on macOS backend.
- Improved type hints
- Improved error handling for .NET backend.
- Documentation fixes.

1.9.15 0.5.1 (2019-10-09)

- Active Scanning on Windows, #99 potentially solving #95
- Longer timeout in service discovery on BlueZ
- Added `timeout` to constructors and connect methods
- Fix for `get_services` on macOS. Relates to #101
- Fixes for disconnect callback on BlueZ, #86 and #83
- Fixed reading of device name in BlueZ. It is not readable as regular characteristic. #104
- Removed logger feedback in BlueZ discovery method.
- More verbose exceptions on macOS, #117 and #107

1.9.16 0.5.0 (2019-08-02)

- macOS support added (thanks to @kevincar)
- Merged #90 which fixed #89: Leaking callbacks in BlueZ
- Merged #92 which fixed #91, Prevent leaking of DBus connections on discovery
- Merged #96: Regex patterns
- Merged #86 which fixed #83 and #82
- Recovered old .NET discovery method to try for #95
- Merged #80: macOS development

1.9.17 0.4.3 (2019-06-30)

- Fix for #76
- Fix for #69
- Fix for #74
- Fix for #68
- Fix for #70
- Merged #66

1.9.18 0.4.2 (2019-05-17)

- Fix for missed part of PR #61.

1.9.19 0.4.1 (2019-05-17)

- Merging of PR #61, improvements and fixes for multiple issues for BlueZ backend
- Implementation of issue #57
- Fixing issue #59
- Documentation fixes.

1.9.20 0.4.0 (2019-04-10)

- Transferred code from the BleakUWPBridge C# support project to pythonnet code
- Fixed BlueZ >= 5.48 issues regarding Battery Service
- Fix for issue #55

1.9.21 0.3.0 (2019-03-18)

- Fix for issue #53: Windows and Python 3.7 error
- Azure Pipelines used for CI

1.9.22 0.2.4 (2018-11-30)

- Fix for issue #52: Timing issue getting characteristics
- Additional fix for issue #51.
- Bugfix for string method for BLEDevice.

1.9.23 0.2.3 (2018-11-28)

- Fix for issue #51: `dpkg-query not found on all Linux systems`

1.9.24 0.2.2 (2018-11-08)

- Made it compliant with Python 3.5 by removing f-strings

1.9.25 0.2.1 (2018-06-28)

- Improved logging on .NET discover method
- Some type annotation fixes in .NET code

1.9.26 0.2.0 (2018-04-26)

- Project added to Github
- First version on PyPI.
- Working Linux (BlueZ DBus API) backend.
- Working Windows (UWP Bluetooth API) backend.

1.9.27 0.1.0 (2017-10-23)

- Bleak created.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

b

Index

W