bleak Documentation

Release 0.22.3

Henrik Blidh

Oct 05, 2024

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

ONE

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 Develop branch

The develop branch can also be installed using `pip`. This is useful for testing the latest changes before they reach the stable release.

```
$ pip install https://github.com/hbldh/bleak/archive/refs/heads/develop.zip
```

For checking out a copy of Bleak for developing Bleak itself, see the *Contributing* page.

1.2 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 asynchronous context manager like this:

```
import asyncio
from bleak import BleakClient

address = "24:71:89:cc:09:05"
MODEL_NBR_UUID = "2A24"

async def main(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))))

asyncio.run(main(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 = "2A24"

async def main(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()

asyncio.run(main(address))
```

Warning: Do not name your script `bleak.py`! It will cause a circular import error.

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.

1.3 API reference

Contents:

1.3.1 BleakScanner class

class bleak.**BleakScanner**(*detection_callback: Optional[AdvertisementDataCallback] = None*, *service_uuids: Optional[List[str]] = None*, *scanning_mode: Literal['active', 'passive'] = 'active'*, ***, *bluez: BlueZScannerArgs = {}*, *cb: CBScannerArgs = {}*, *backend: Optional[Type[BaseBleakScanner]] = None*, ***kwargs*)

Interface for Bleak Bluetooth LE Scanners.

The scanner will listen for BLE advertisements, optionally filtering on advertised services or other conditions, and collect a list of `BLEDevice` objects. These can subsequently be used to connect to the corresponding BLE server.

A *`BleakScanner`* can be used as an asynchronous context manager in which case it automatically starts and stops scanning.

Parameters

- **`detection_callback`** – Optional function that will be called each time a device is discovered or advertising data has changed.
- **`service_uuids`** – Optional list of service UUIDs to filter on. Only advertisements containing this advertising data will be received. Required on macOS >= 12.0, < 12.3 (unless you create an app with `py2app`).
- **`scanning_mode`** – Set to `"passive"` to avoid the `"active"` scanning mode. Passive scanning is not supported on macOS! Will raise `BleakError` if set to `"passive"` on macOS.
- **`bluez`** – Dictionary of arguments specific to the BlueZ backend.
- **`cb`** – Dictionary of arguments specific to the CoreBluetooth backend.
- **`backend`** – Used to override the automatically selected backend (i.e. for a custom backend).
- **`**kwargs`** – Additional args for backwards compatibility.

Tip: The first received advertisement in `detection_callback` may or may not include scan response data if the remote device supports it. Be sure to take this into account when handing the callback. For example, the scan response often contains the local name of the device so if you are matching a device based on other data but want to display the local name to the user, be sure to wait for `adv_data.local_name is not None`.

Changed in version 0.15: `detection_callback`, `service_uuids` and `scanning_mode` are no longer keyword-only. Added `bluez` parameter.

Changed in version 0.18: No longer is alias for backend type and no longer inherits from `BaseBleakScanner`. Added `backend` parameter.

Easy methods

These methods and handy for simple programs but are not recommended for more advanced use cases like long running programs, GUIs or connecting to multiple devices.

async classmethod BleakScanner.**discover**(*timeout: float = 5.0*, ***, *return_adv: Literal[False] = False*, ***kwargs*) → List[*BLEDevice*]

async classmethod BleakScanner.**discover**(*timeout: float = 5.0*, ***, *return_adv: Literal[True]*, ***kwargs*) → Dict[str, Tuple[*BLEDevice*, *AdvertisementData*]]

Scan continuously for `timeout` seconds and return discovered devices.

Parameters

- **timeout** – Time, in seconds, to scan for.
- **return_adv** – If `True`, the return value will include advertising data.
- ****kwargs** – Additional arguments will be passed to the *BleakScanner* constructor.

Returns

The value of *discovered_devices_and_advertisement_data* if `return_adv` is `True`, otherwise the value of *discovered_devices*.

Changed in version 0.19: Added `return_adv` parameter.

async classmethod BleakScanner.**find_device_by_name**(*name: str*, *timeout: float = 10.0*, ***kwargs: Unpack[*ExtraArgs*]*) → Optional[*BLEDevice*]

Obtain a `BLEDevice` for a BLE server specified by the local name in the advertising data.

Parameters

- **name** – The name sought.
- **timeout** – Optional timeout to wait for detection of specified peripheral before giving up. Defaults to 10.0 seconds.
- ****kwargs** – additional args passed to the *BleakScanner* constructor.

Returns

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

New in version 0.20.

async classmethod BleakScanner.**find_device_by_address**(*device_identifier: str*, *timeout: float = 10.0*, ***kwargs: Unpack[*ExtraArgs*]*) → Optional[*BLEDevice*]

Obtain a `BLEDevice` for a BLE server specified by Bluetooth address or (macOS) UUID address.

Parameters

- **device_identifier** – The Bluetooth/UUID address of the Bluetooth peripheral sought.
- **timeout** – Optional timeout to wait for detection of specified peripheral before giving up. Defaults to 10.0 seconds.
- ****kwargs** – additional args passed to the *BleakScanner* constructor.

Returns

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

async classmethod BleakScanner.**find_device_by_filter**(*filterfunc: AdvertisementDataFilter*, *timeout: float = 10.0*, ***kwargs: Unpack[*ExtraArgs*]*) → Optional[*BLEDevice*]

Obtain a `BLEDevice` for a BLE server that matches a given filter function.

This can be used to find a BLE server by other identifying information than its address, for example its name.

Parameters

- **filterfunc** – A function that is called for every BLEDevice found. It should return `True` only for the wanted device.
- **timeout** – Optional timeout to wait for detection of specified peripheral before giving up. Defaults to 10.0 seconds.
- ****kwargs** – Additional arguments to be passed to the *BleakScanner* constructor.

Returns

The `BLEDevice` sought or `None` if not detected before the timeout.

class bleak.BleakScanner.**ExtraArgs**

Keyword args from *BleakScanner* that can be passed to other convenience methods.

backend: Type[*BaseBleakScanner*]

Used to override the automatically selected backend (i.e. for a custom backend).

bluez: *BlueZScannerArgs*

Dictionary of arguments specific to the BlueZ backend.

cb: *CBScannerArgs*

Dictionary of arguments specific to the CoreBluetooth backend.

scanning_mode: Literal['active', 'passive']

Set to `"passive"` to avoid the `"active"` scanning mode. Passive scanning is not supported on macOS! Will raise `BleakError` if set to `"passive"` on macOS.

service_uuids: List[str]

Optional list of service UUIDs to filter on. Only advertisements containing this advertising data will be received. Required on macOS >= 12.0, < 12.3 (unless you create an app with `py2app`).

Starting and stopping

BleakScanner is an context manager so the recommended way to start and stop scanning is to use it in an `async with` statement:

```
import asyncio
from bleak import BleakScanner

async def main():
    stop_event = asyncio.Event()

    # TODO: add something that calls stop_event.set()

    def callback(device, advertising_data):
        # TODO: do something with incoming data
        pass

    async with BleakScanner(callback) as scanner:
        ...
```

(continues on next page)

(continued from previous page)

```
        # Important! Wait for an event to trigger stop, otherwise scanner
        # will stop immediately.
        await stop_event.wait()

    # scanner stops when block exits
    ...

asyncio.run(main())
```

It can also be started and stopped without using the context manager using the following methods:

async BleakScanner.**start**() → None

Start scanning for devices

async BleakScanner.**stop**() → None

Stop scanning for devices

Getting discovered devices and advertisement data

If you aren't using the "easy" class methods, there are three ways to get the discovered devices and advertisement data.

For event-driven programming, you can provide a `detection_callback` callback to the *BleakScanner* constructor. This will be called back each time and advertisement is received.

Alternatively, you can utilize the asynchronous iterator to iterate over advertisements as they are received. The method below returns an async iterator that yields the same tuples as otherwise provided to `detection_callback`.

async BleakScanner.**advertisement_data**() → AsyncGenerator[Tuple[*BLEDevice*, *AdvertisementData*], None]

Yields devices and associated advertising data packets as they are discovered.

Note: Ensure that scanning is started before calling this method.

Returns

An async iterator that yields tuples (`BLEDevice`, `AdvertisementData`).

New in version 0.21.

Otherwise, you can use one of the properties below after scanning has stopped.

property BleakScanner.**discovered_devices: List[*BLEDevice*]**

Gets list of the devices that the scanner has discovered during the scanning.

If you also need advertisement data, use *discovered_devices_and_advertisement_data* instead.

property BleakScanner.**discovered_devices_and_advertisement_data: Dict[str, Tuple[*BLEDevice*, *AdvertisementData*]]**

Gets a map of device address to tuples of devices and the most recently received advertisement data for that device.

The address keys are useful to compare the discovered devices to a set of known devices. If you don't need to do that, consider using `discovered_devices_and_advertisement_data.values()` to just get the values instead.

New in version 0.19.

Deprecated

BleakScanner.**register_detection_callback**(*callback: Optional[Callable[[*BLEDevice, AdvertisementData*], Optional[Coroutine[Any, Any, None]]]]*) → None

Register a callback that is called when a device is discovered or has a property changed.

Deprecated since version 0.17.0: This method will be removed in a future version of Bleak. Pass the callback directly to the *BleakScanner* constructor instead.

Parameters
callback – A function, coroutine or None.

BleakScanner.**set_scanning_filter**(***kwargs*) → None

Set scanning filter for the BleakScanner.

Deprecated since version 0.17.0: This method will be removed in a future version of Bleak. Pass arguments directly to the *BleakScanner* constructor instead.

Parameters
****kwargs** – The filter details.

async BleakScanner.**get_discovered_devices**() → List[*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.

1.3.2 BleakClient class

class bleak.**BleakClient**(*address_or_ble_device: Union[*BLEDevice, *str]*, *disconnected_callback: Optional[Callable[[*BleakClient*], None]] = None*, *services: Optional[Iterable[str]] = None*, ***, *timeout: float = 10.0*, *winrt:* WinRTClientArgs *= {}*, *backend: Optional[Type[*BaseBleakClient*]] = None*, ***kwargs*)

The Client interface for connecting to a specific BLE GATT server and communicating with it.

A BleakClient can be used as an asynchronous context manager in which case it automatically connects and disconnects.

How many BLE connections can be active simultaneously, and whether connections can be active while scanning depends on the Bluetooth adapter hardware.

Parameters

- **address_or_ble_device** – A BLEDevice received from a *BleakScanner* or a Bluetooth address (device UUID on macOS).
- **disconnected_callback** – 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.
- **services** – Optional list of services to filter. If provided, only these services will be resolved. This may or may not reduce the time needed to enumerate the services depending on if the OS supports such filtering in the Bluetooth stack or not (should affect Windows and Mac). These can be 16-bit or 128-bit UUIDs.

- **timeout** – Timeout in seconds passed to the implicit `discover` call when `address_or_ble_device` is not a `BLEDevice`. Defaults to 10.0.
- **winrt** – Dictionary of WinRT/Windows platform-specific options.
- **backend** – Used to override the automatically selected backend (i.e. for a custom backend).
- ****kwargs** – Additional keyword arguments for backwards compatibility.

Warning: Although example code frequently initializes *BleakClient* with a Bluetooth address for simplicity, it is not recommended to do so for more complex use cases. There are several known issues with providing a Bluetooth address as the `address_or_ble_device` argument.

1. macOS does not provide access to the Bluetooth address for privacy/ security reasons. Instead it creates a UUID for each Bluetooth device which is used in place of the address on this platform.
2. Providing an address or UUID instead of a `BLEDevice` causes the *connect()* method to implicitly call *BleakScanner.discover()*. This is known to cause problems when trying to connect to multiple devices at the same time.

Changed in version 0.15: `disconnected_callback` is no longer keyword-only. Added `winrt` parameter.

Changed in version 0.18: No longer is alias for backend type and no longer inherits from `BaseBleakClient`. Added `backend` parameter.

Connecting and disconnecting

Before doing anything else with a `BleakClient` object, it must be connected.

bleak.BleakClient is a an async context manager, so the recommended way of connecting is to use it as such:

```
import asyncio
from bleak import BleakClient

async def main():
    async with BleakClient("XX:XX:XX:XX:XX:XX") as client:
        # Read a characteristic, etc.
        ...

    # Device will disconnect when block exits.
    ...

# Using asyncio.run() is important to ensure that device disconnects on
# KeyboardInterrupt or other unhandled exception.
asyncio.run(main())
```

It is also possible to connect and disconnect without a context manager, however this can leave the device still connected when the program exits:

async BleakClient.**connect**(***kwargs*) → bool

Connect to the specified GATT server.

Parameters
****kwargs** – For backwards compatibility - should not be used.

Returns
Always returns `True` for backwards compatibility.

async BleakClient.**disconnect**() → bool

Disconnect from the specified GATT server.

Returns
Always returns True for backwards compatibility.

The current connection status can be retrieved with:

property BleakClient.**is_connected: bool**

Check connection status between this client and the GATT server.

Returns
Boolean representing connection status.

A callback can be provided to the BleakClient constructor via the disconnect_callback argument to be notified of disconnection events.

Device information

property BleakClient.**address: str**

Gets the Bluetooth address of this device (UUID on macOS).

property BleakClient.**mtu_size: int**

Gets the negotiated MTU size in bytes for the active connection.

Consider using *bleak.backends.characteristic.BleakGATTCharacteristic.max_write_without_response_size* instead.

> **Warning:** The BlueZ backend will always return 23 (the minimum MTU size). See the mtu_size.py example for a way to hack around this.

GATT Client Operations

All Bluetooth Low Energy devices use a common Generic Attribute Profile (GATT) for interacting with the device after it is connected. Some GATT operations like discovering the services/characteristic/descriptors and negotiating the MTU are handled automatically by Bleak and/or the OS Bluetooth stack.

The primary operations for the Bleak client are reading, writing and subscribing to characteristics.

Services

The available services on a device are automatically enumerated when connecting to a device. Services describe the devices capabilities.

property BleakClient.**services: *BleakGATTServiceCollection***

Gets the collection of GATT services available on the device.

The returned value is only valid as long as the device is connected.

Raises
BleakError – if service discovery has not been performed yet during this connection.

GATT characteristics

Most I/O with a device is done via the characteristics.

async BleakClient.**read_gatt_char**(*char_specifier: Union[BleakGATTCharacteristic, int, str, UUID]*, ***kwargs*) → bytearray

Perform read operation on the specified GATT characteristic.

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

Returns
The read data.

async BleakClient.**write_gatt_char**(*char_specifier: Union[BleakGATTCharacteristic, int, str, UUID]*, *data: Buffer*, *response: bool = None*) → None

Perform a write operation on the specified GATT characteristic.

There are two possible kinds of writes. *Write with response* (sometimes called a *Request*) will write the data then wait for a response from the remote device. *Write without response* (sometimes called *Command*) will queue data to be written and return immediately.

Each characteristic may support one kind or the other or both or neither. Consult the device's documentation or inspect the properties of the characteristic to find out which kind of writes are supported.

Tip: Explicit is better than implicit. Best practice is to always include an explicit `response=True` or `response=False` when calling this method.

Parameters

- **char_specifier** – The characteristic to write to, specified by either integer handle, UUID or directly by the *BleakGATTCharacteristic* object representing it. If a device has more than one characteristic with the same UUID, then attempting to use the UUID wil fail and a characteristic object must be used instead.
- **data** – The data to send. When a write-with-response operation is used, the length of the data is limited to 512 bytes. When a write-without-response operation is used, the length of the data is limited to *max_write_without_response_size*. Any type that supports the buffer protocol can be passed.
- **response** – If `True`, a write-with-response operation will be used. If `False`, a write-without-response operation will be used. If omitted or `None`, the "best" operation will be used based on the reported properties of the characteristic.

Changed in version 0.21: The default behavior when `response=` is omitted was changed.

Example:

```
MY_CHAR_UUID = "1234"
...
await client.write_gatt_char(MY_CHAR_UUID, b"", response=True)
```

async BleakClient.**start_notify**(*char_specifier: Union[BleakGATTCharacteristic, int, str, UUID]*, *callback: Callable[[BleakGATTCharacteristic, bytearray], Union[None, Awaitable[None]]]*, ***kwargs*) → None

Activate notifications/indications on a characteristic.

Callbacks must accept two inputs. The first will be the characteristic and the second will be a `bytearray` containing the data received.

```
def callback(sender: BleakGATTCharacteristic, data: bytearray):
    print(f"{sender}: {data}")

client.start_notify(char_uuid, callback)
```

Parameters

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

Changed in version 0.18: The first argument of the callback is now a `BleakGATTCharacteristic` instead of an `int`.

async BleakClient.**stop_notify**(*char_specifier: Union[*BleakGATTCharacteristic*, int, str, UUID]*) → None

Deactivate notification/indication on a specified characteristic.

Parameters

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

Tip: Notifications are stopped automatically on disconnect, so this method does not need to be called unless notifications need to be stopped some time before the device disconnects.

GATT descriptors

Descriptors can provide additional information about a characteristic.

async BleakClient.**read_gatt_descriptor**(*handle: int*, ***kwargs*) → bytearray

Perform read operation on the specified GATT descriptor.

Parameters

handle – The handle of the descriptor to read from.

Returns

The read data.

async BleakClient.**write_gatt_descriptor**(*handle: int*, *data: Buffer*) → None

Perform a write operation on the specified GATT descriptor.

Parameters

- **handle** – The handle of the descriptor to read from.
- **data** – The data to send.

Pairing/bonding

On some devices, some characteristics may require authentication in order to read or write the characteristic. In this case pairing/bonding the device is required.

`async BleakClient.pair(*args, **kwargs) → bool`

Pair with the specified GATT server.

This method is not available on macOS. Instead of manually initiating paring, the user will be prompted to pair the device the first time that a characteristic that requires authentication is read or written. This method may have backend-specific additional keyword arguments.

Returns
Always returns `True` for backwards compatibility.

`async BleakClient.unpair() → bool`

Unpair from the specified GATT server.

Unpairing will also disconnect the device.

This method is only available on Windows and Linux and will raise an exception on other platforms.

Returns
Always returns `True` for backwards compatibility.

Deprecated

`BleakClient.set_disconnected_callback(callback: Optional[Callable[[BleakClient], None]], **kwargs) → None`

Set the disconnect callback.

Deprecated since version 0.17.0: This method will be removed in a future version of Bleak. Pass the callback to the *BleakClient* constructor instead.

Parameters
callback – callback to be called on disconnection.

`async BleakClient.get_services(**kwargs) → BleakGATTServiceCollection`

Get all services registered for this GATT server.

Deprecated since version 0.17.0: This method will be removed in a future version of Bleak. Use the *services* property instead.

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

1.3.3 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: str, name: Optional[str], details: Any, rssi: int, **kwargs)`

A simple wrapper class representing a BLE server detected during scanning.

address

The Bluetooth address of the device on this machine (UUID on macOS).

details

The OS native details required for connecting to the device.

property metadata: dict

Gets additional advertisement data for the device.

Deprecated since version 0.19.0: Use `AdvertisementData` from detection callback or `BleakScanner.discovered_devices_and_advertisement_data` instead.

name

The operating system name of the device (not necessarily the local name from the advertising data), suitable for display to the user.

property rssi: int

Gets the RSSI of the last received advertisement.

Deprecated since version 0.19.0: Use `AdvertisementData` from detection callback or `BleakScanner.discovered_devices_and_advertisement_data` instead.

1.3.4 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: Any*)

Interface for the Bleak representation of a GATT Service.

abstract add_characteristic(*characteristic:* BleakGATTCharacteristic) → None

Add a `BleakGATTCharacteristic` to the service.

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

abstract property characteristics: List[*BleakGATTCharacteristic*]

List of characteristics for this service

property description: str

String description for this service

get_characteristic(*uuid: Union[str, UUID]*) → Optional[*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.

abstract property handle: int

The handle of this service

abstract property uuid: str

The UUID to this service

class bleak.backends.service.**BleakGATTServiceCollection**

Simple data container for storing the peripheral's service complement.

add_characteristic(*characteristic:* BleakGATTCharacteristic) → None

Add a `BleakGATTCharacteristic` to the service collection.

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

add_descriptor(*descriptor:* BleakGATTDescriptor) → None

Add a `BleakGATTDescriptor` to the service collection.

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

add_service(*service:* BleakGATTService) → None

Add a *BleakGATTService* to the service collection.

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

property characteristics: Dict[int, *BleakGATTCharacteristic*]

Returns dictionary of handles mapping to BleakGATTCharacteristic

property descriptors: Dict[int, *BleakGATTDescriptor*]

Returns a dictionary of integer handles mapping to BleakGATTDescriptor

get_characteristic(*specifier: Union[int, str, UUID]*) → Optional[*BleakGATTCharacteristic*]

Get a characteristic by handle (int) or UUID (str or uuid.UUID)

get_descriptor(*handle: int*) → Optional[*BleakGATTDescriptor*]

Get a descriptor by integer handle

get_service(*specifier: Union[int, str, UUID]*) → Optional[*BleakGATTService*]

Get a service by handle (int) or UUID (str or uuid.UUID)

property services: Dict[int, *BleakGATTService*]

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*, *max_write_without_response_size: Callable[[], int]*)

Interface for the Bleak representation of a GATT Characteristic

abstract add_descriptor(*descriptor:* BleakGATTDescriptor) → None

Add a `BleakGATTDescriptor` to the characteristic.

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

property description: str

Description for this characteristic

abstract property descriptors: List[*BleakGATTDescriptor*]

List of descriptors for this service

abstract get_descriptor(*specifier: Union[int, str, UUID]*) → Optional[*BleakGATTDescriptor*]

Get a descriptor by handle (int) or UUID (str or uuid.UUID)

abstract property handle: int

The handle for this characteristic

property max_write_without_response_size: int

Gets the maximum size in bytes that can be used for the *data* argument of `BleakClient.write_gatt_char()` when `response=False`.

In rare cases, a device may take a long time to update this value, so reading this property may return the default value of `20` and reading it again after a some time may return the expected higher value.

If you *really* need to wait for a higher value, you can do something like this:

```
async with asyncio.timeout(10):
    while char.max_write_without_response_size == 20:
        await asyncio.sleep(0.5)
```

Warning: Linux quirk: For BlueZ versions < 5.62, this property will always return `20`.

New in version 0.16.

abstract property properties: List[str]

Properties of this characteristic

abstract property service_handle: int

The integer handle of the Service containing this characteristic

abstract property service_uuid: str

The UUID of the Service containing this characteristic

abstract property uuid: str

The UUID for this characteristic

class bleak.backends.characteristic.**GattCharacteristicsFlags**(*value*, *names=<not given>*, **values*, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

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

abstract property characteristic_handle: int

handle for the characteristic that this descriptor belongs to

abstract property characteristic_uuid: str

UUID for the characteristic that this descriptor belongs to

property description: str

A text description of what this descriptor represents

abstract property handle: int

Integer handle for this descriptor

abstract property uuid: str

UUID for this descriptor

1.3.5 Exceptions

exception bleak.exc.**BleakCharacteristicNotFoundError**(*char_specifier: Union[int, str, UUID]*)

Exception which is raised if a device does not support a characteristic.

exception bleak.exc.**BleakDBusError**(*dbus_error: str*, *error_body: list*)

Specialized exception type for D-Bus errors.

property dbus_error: str

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

property dbus_error_details: Optional[str]

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

exception bleak.exc.**BleakDeviceNotFoundError**(*identifier: str*, **args: object*)

Exception which is raised if a device can not be found by `connect`, `pair` and `unpair`. This is the case if the OS Bluetooth stack has never seen this device or it was removed and forgotten.

exception bleak.exc.**BleakError**

Base Exception for bleak.

1.3.6 Utilities

bleak.uuids.**normalize_uuid_16**(*uuid: int*) → str

Normaizes a 16-bit integer UUID to the format used by Bleak.

Returns
128-bit UUID as string with the format `"0000xxxx-0000-1000-8000-00805f9b34fb"`.

Example:

```
uuid = normalize_uuid_16(0x1234)
# uuid == "00001234-0000-1000-8000-00805f9b34fb"
```

New in version 0.21.

bleak.uuids.**normalize_uuid_32**(*uuid: int*) → str

Normaizes a 32-bit integer UUID to the format used by Bleak.

Returns
128-bit UUID as string with the format `"xxxxxxxx-0000-1000-8000-00805f9b34fb"`.

Example:

```
uuid = normalize_uuid_32(0x12345678)
# uuid == "12345678-0000-1000-8000-00805f9b34fb"
```

New in version 0.21.

bleak.uuids.**normalize_uuid_str**(*uuid: str*) → str

Normaizes a UUID to the format used by Bleak.

- Converted to lower case.
- 16-bit and 32-bit UUIDs are expanded to 128-bit.

Example:

```
# 16-bit
uuid1 = normalize_uuid_str("1234")
# uuid1 == "00001234-0000-1000-8000-00805f9b34fb"

# 32-bit
uuid2 = normalize_uuid_str("12345678")
# uuid2 == "12345678-0000-1000-8000-00805f9b34fb"

# 128-bit
uuid3 = normalize_uuid_str("12345678-0000-1234-1234-1234567890ABC")
# uuid3 == "12345678-0000-1234-1234-1234567890abc"
```

New in version 0.20.

Changed in version 0.21: Added support for 32-bit UUIDs.

bleak.uuids.**register_uuids**(*uuids_to_descriptions: Dict[str, str]*) → None

Add or modify the mapping of 128-bit UUIDs for services and characteristics to descriptions.

Parameters
uuids_to_descriptions – A dictionary of new mappings

1.3.7 Deprecated

bleak.**discover**(**args*, ***kwargs*)

Deprecated since version 0.17.0: This method will be removed in a future version of Bleak. Use *BleakScanner.discover()* instead.

1.4 Backend implementations

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
- Partial Android support mostly using Python-for-Android/Kivy.

These pages document platform specific differences from the interface API.

Contents:

1.4.1 Windows backend

The Windows backend of bleak is written using PyWinRT to provide bindings for the Windows Runtime (WinRT).

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

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.

API

Utilities

`bleak.backends.winrt.util.`**`allow_sta`**`()`

Suppress check for MTA thread type and allow STA.

Bleak will hang forever if the current thread is not MTA - unless there is a Windows event loop running that is properly integrated with asyncio in Python.

If your program meets that condition, you must call this function do disable the check for MTA. If your program doesn't have a graphical user interface you probably shouldn't call this function. and use `uninitialize_sta()` instead.

New in version 0.22.1.

`async` `bleak.backends.winrt.util.`**`assert_mta`**`()` → None

Asserts that the current apartment type is MTA.

Raises

`BleakError` – If the current apartment type is not MTA and there is no Windows message loop running.

New in version 0.22.

Changed in version 0.22.2: Function is now async and will not raise if the current apartment type is STA and the Windows message loop is running.

`bleak.backends.winrt.util.`**`uninitialize_sta`**`()`

Uninitialize the COM library on the current thread if it was not initialized as MTA.

This is intended to undo the implicit initialization of the COM library as STA by packages like pywin32.

It should be called as early as possible in your application after the offending package has been imported.

New in version 0.22.

Scanner

`class` `bleak.backends.winrt.scanner.`**`BleakScannerWinRT`**`(`*detection_callback: Optional[Callable[[BLEDevice, AdvertisementData], Optional[Coroutine[Any, Any, None]]]]*, *service_uuids: Optional[List[str]]*, *scanning_mode: Literal['active', 'passive']*, ***kwargs*`)`

The native Windows Bleak BLE Scanner.

Implemented using Python/WinRT.

Parameters

- **detection_callback** – Optional function that will be called each time a device is discovered or advertising data has changed.
- **service_uuids** – Optional list of service UUIDs to filter on. Only advertisements containing this advertising data will be received.
- **scanning_mode** – Set to "passive" to avoid the "active" scanning mode.

set_scanning_filter(***kwargs*) → None

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.

async start() → None

Start scanning for devices

async stop() → None

Stop scanning for devices

Client

BLE Client for Windows 10 systems, implemented with WinRT.

Created on 2020-08-19 by hbldh <henrik.blidh@nedomkull.com>

class bleak.backends.winrt.client.**BleakClientWinRT**(*address_or_ble_device: Union[BLEDevice, str]*, *services: Optional[Set[str]] = None*, ***, *winrt: WinRTClientArgs*, ***kwargs*)

Native Windows Bleak Client.

Parameters

- **address_or_ble_device** (*str or BLEDevice*) – The Bluetooth address of the BLE peripheral to connect to or the BLEDevice object representing it.
- **services** – Optional set of service UUIDs that will be used.
- **winrt** (*dict*) – A dictionary of Windows-specific configuration values.
- ****timeout** (*float*) – Timeout for required BleakScanner.find_device_by_address call. Defaults to 10.0.

async 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.

async disconnect() → bool

Disconnect from the specified GATT server.

Returns
Boolean representing if device is disconnected.

async get_services(*, *service_cache_mode: Optional[winrt.windows.devices.bluetooth.BluetoothCacheMode] = None*, *cache_mode: Optional[winrt.windows.devices.bluetooth.BluetoothCacheMode] = None*, ***kwargs*) → *BleakGATTServiceCollection*

Get all services registered for this GATT server.

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

property is_connected: bool

Check connection status between this client and the server.

Returns
Boolean representing connection status.

property mtu_size: int

Get ATT MTU size for active connection

async pair(*protection_level: int = None*, ***kwargs*) → bool

Attempts to pair with the device.

Keyword Arguments
protection_level (*int*) – A `DevicePairingProtectionLevel` enum value:

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.

async read_gatt_char(*char_specifier: Union[*BleakGATTCharacteristic*, int, str, 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.

async 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.

async start_notify(*characteristic:* BleakGATTCharacteristic, *callback: Callable[[bytearray], None]*, ***kwargs*) → None

Activate notifications/indications on a characteristic.

Keyword Arguments
force_indicate (*bool*) – If this is set to True, then Bleak will set up a indication request instead of a notification request, given that the characteristic supports notifications as well as indications.

async stop_notify(*char_specifier: Union[*BleakGATTCharacteristic*, int, str, 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.

async 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.

async write_gatt_char(*characteristic:* BleakGATTCharacteristic, *data: Buffer*, *response: bool*) → None

Perform a write operation on the specified GATT characteristic.

Parameters

- **characteristic** – The characteristic to write to.
- **data** – The data to send.
- **response** – If write-with-response operation should be done.

async write_gatt_descriptor(*handle: int*, *data: Buffer*) → None

Perform a write operation on the specified GATT descriptor.

Parameters

- **handle** – The handle of the descriptor to read from.
- **data** – The data to send (any bytes-like object).

class bleak.backends.winrt.client.**FutureLike**(*op: winrt.windows.foundation.IAsyncOperation*)

Wraps a WinRT IAsyncOperation in a "future-like" object so that it can be passed to Python APIs.

Needed until https://github.com/pywinrt/pywinrt/issues/14

class bleak.backends.winrt.client.**WinRTClientArgs**

Windows-specific arguments for BleakClient.

address_type: Literal['public', 'random']

Can either be "public" or "random", depending on the required address type needed to connect to your device.

use_cached_services: bool

True allows Windows to fetch the services, characteristics and descriptors from the Windows cache instead of reading them from the device. Can be very much faster for known, unchanging devices, but not recommended for DIY peripherals where the GATT layout can change between connections.

False will force the attribute database to be read from the remote device instead of using the OS cache.

If omitted, the OS Bluetooth stack will do what it thinks is best.

1.4.2 Linux backend

The Linux backend of Bleak communicates with BlueZ over DBus. Communication uses the dbus-fast package for async access to DBus messaging.

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".

Resolving services with `get_services`

By default, calling `get_services` will wait for services to be resolved before returning the `BleakGATTServiceCollection`. If a previous connection to the device was made, passing the `dangerous_use_bleak_cache` argument will return the cached services without waiting for them to be resolved again. This is useful when you know services have not changed, and you want to use the services immediately, but don't want to wait for them to be resolved again.

Parallel Access

Each Bleak object should be created and used from a single asyncio event loop. Simple asyncio programs will only have a single event loop. It's also possible to use Bleak with multiple event loops, even at the same time, but individual Bleak objects should not be shared between event loops. Otherwise, RuntimeErrors similar to `[...] got Future <Future pending> attached to a different loop` will be thrown.

D-Bus Authentication

Connecting to the host DBus from within a user namespace will fail. This is because the remapped UID will not match the UID that the hosts sees. To work around this, you can hardcode a UID with the *BLEAK_DBUS_AUTH_UID* environment variable.

API

Scanner

class bleak.backends.bluezdbus.scanner.**BleakScannerBlueZDBus**(*detection_callback: Optional[Callable[[BLEDevice, AdvertisementData], Optional[Coroutine[Any, Any, None]]]]*, *service_uuids: Optional[List[str]]*, *scanning_mode: Literal['active', 'passive']*, ***, *bluez: BlueZScannerArgs*, ***kwargs*)

The native Linux Bleak BLE Scanner.

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

Parameters

- **detection_callback** – Optional function that will be called each time a device is discovered or advertising data has changed.
- **service_uuids** – Optional list of service UUIDs to filter on. Only advertisements containing this advertising data will be received. Specifying this also enables scanning while the screen is off on Android.
- **scanning_mode** – Set to `"passive"` to avoid the `"active"` scanning mode.
- ****bluez** – Dictionary of arguments specific to the BlueZ backend.
- ****adapter** (*str*) – Bluetooth adapter to use for discovery.

set_scanning_filter(***kwargs*) → None

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.

async start() → None

Start scanning for devices

async stop() → None

Stop scanning for devices

class bleak.backends.bluezdbus.scanner.**BlueZDiscoveryFilters**

Dictionary of arguments for the `org.bluez.Adapter1.SetDiscoveryFilter` D-Bus method.

https://github.com/bluez/bluez/blob/master/doc/org.bluez.Adapter.rst#void-setdiscoveryfilterdict-filter

Discoverable: bool

Make adapter discoverable while discovering, if the adapter is already discoverable setting this filter won't do anything.

DuplicateData: bool

Disables duplicate detection of advertisement data.

This does not affect the `Filter Duplicates` parameter of the `LE Set Scan Enable` HCI command to the Bluetooth adapter!

Although the default value for BlueZ is `True`, Bleak sets this to `False` by default.

Pathloss: int

Pathloss threshold value.

Pattern: str

Discover devices where the pattern matches either the prefix of the address or device name which is convenient way to limited the number of device objects created during a discovery.

RSSI: int

RSSI threshold value.

Transport: str

Transport parameter determines the type of scan.

This should not be used since it is required to be set to `"le"`.

UUIDs: List[str]

Filter by service UUIDs, empty means match _any_ UUID.

Normally, the `service_uuids` argument of *bleak.BleakScanner* is used instead.

class bleak.backends.bluezdbus.scanner.**BlueZScannerArgs**

`BleakScanner` args that are specific to the BlueZ backend.

filters: *BlueZDiscoveryFilters*

Filters to pass to the adapter SetDiscoveryFilter D-Bus method.

Only used for active scanning.

or_patterns: List[Union[OrPattern, Tuple[int, AdvertisementDataType, bytes]]]

Or patterns to pass to the AdvertisementMonitor1 D-Bus interface.

Only used for passive scanning.

Client

BLE Client for BlueZ on Linux

class bleak.backends.bluezdbus.client.**BleakClientBlueZDBus**(*address_or_ble_device: Union[BLEDevice, str]*, *services: Optional[Set[str]] = None*, ***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.
- **services** – Optional list of service UUIDs that will be used.

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.

async connect(*dangerous_use_bleak_cache: bool = False*, ***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

async 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** –

async get_services(*dangerous_use_bleak_cache: bool = False*, ***kwargs*) → *BleakGATTServiceCollection*

Get all services registered for this GATT server.

Parameters
dangerous_use_bleak_cache (*bool*) – Use cached services if available.

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

property is_connected: bool

Check connection status between this client and the server.

Returns
Boolean representing connection status.

property mtu_size: int

Get ATT MTU size for active connection

async 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.

async read_gatt_char(*char_specifier: Union[BleakGATTCharacteristicBlueZDBus, int, str, 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.

async 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.

async start_notify(*characteristic:* BleakGATTCharacteristic, *callback: Callable[[bytearray], None]*, ***kwargs*) → None

Activate notifications/indications on a characteristic.

async stop_notify(*char_specifier: Union[BleakGATTCharacteristicBlueZDBus, int, str, 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.

async unpair() → bool

Unpair with the peripheral.

Returns

Boolean regarding success of unpairing.

async write_gatt_char(*characteristic:* BleakGATTCharacteristic, *data: Buffer*, *response: bool*) → None

Perform a write operation on the specified GATT characteristic.

Parameters

- **characteristic** – The characteristic to write to.
- **data** – The data to send.
- **response** – If write-with-response operation should be done.

async write_gatt_descriptor(*handle: int*, *data: Buffer*) → None

Perform a write operation on the specified GATT descriptor.

Parameters

- **handle** – The handle of the descriptor to read from.
- **data** – The data to send (any bytes-like object).

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.

API

Scanner

class bleak.backends.corebluetooth.scanner.**BleakScannerCoreBluetooth**(*detection_callback: Optional[Callable[[BLEDevice, AdvertisementData], Optional[Coroutine[Any, Any, None]]]]*, *service_uuids: Optional[List[str]]*, *scanning_mode: Literal['active', 'passive']*, ***, *cb: CBScannerArgs*, ***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.

Parameters

- **detection_callback** – Optional function that will be called each time a device is discovered or advertising data has changed.
- **service_uuids** – Optional list of service UUIDs to filter on. Only advertisements containing this advertising data will be received. Required on macOS >= 12.0, < 12.3 (unless you create an app with `py2app`).

- **scanning_mode** – Set to "passive" to avoid the "active" scanning mode. Not supported on macOS! Will raise BleakError if set to "passive"
- ****timeout** (*float*) – The scanning timeout to be used, in case of missing stopScan_ method.

set_scanning_filter(***kwargs*) → None

Set scanning filter for the scanner.

Note: This is not implemented for macOS yet.

Raises
NotImplementedError –

async start() → None

Start scanning for devices

async stop() → None

Stop scanning for devices

class bleak.backends.corebluetooth.scanner.**CBScannerArgs**

Platform-specific BleakScanner args for the CoreBluetooth backend.

use_bdaddr: bool

If true, use Bluetooth address instead of UUID.

Warning: This uses an undocumented IOBluetooth API to get the Bluetooth address and may break in the future macOS releases. It is known to not work on macOS 10.15.

Client

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[BLEDevice, str]*, *services: Optional[Set[str]] = None*, ***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.
- **services** – Optional set of service UUIDs that will be used.

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

async 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.

async disconnect() → bool

Disconnect from the peripheral device

async get_rssi() → int

To get RSSI value in dBm of the connected Peripheral

async get_services(***kwargs*) → *BleakGATTServiceCollection*

Get all services registered for this GATT server.

Returns

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

property is_connected: bool

Checks for current active connection

property mtu_size: int

Get ATT MTU size for active connection

async 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.

async read_gatt_char(*char_specifier: Union[*BleakGATTCharacteristic*, int, str, UUID]*, *use_cached: bool = 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.

async read_gatt_descriptor(*handle: int*, *use_cached: bool = 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.

async start_notify(*characteristic: BleakGATTCharacteristic*, *callback: Callable[[bytearray], None]*, ***kwargs*) → None

Activate notifications/indications on a characteristic.

async stop_notify(*char_specifier: Union[BleakGATTCharacteristic, int, str, 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.

async unpair() → bool

Returns:

async write_gatt_char(*characteristic: BleakGATTCharacteristic*, *data: Buffer*, *response: bool*) → None

Perform a write operation on the specified GATT characteristic.

Parameters

- **characteristic** – The characteristic to write to.
- **data** – The data to send.
- **response** – If write-with-response operation should be done.

async write_gatt_descriptor(*handle: int*, *data: Buffer*) → None

Perform a write operation on the specified GATT descriptor.

Parameters

- **handle** – The handle of the descriptor to read from.
- **data** – The data to send (any bytes-like object).

1.4.4 Android backend

For an example of building an android bluetooth app, see the example folder and its accompanying README file.

There are a handful of ways to run Python on Android. Presently some code has been written for the Python-for-Android build tool, and the code has only been tested using the Kivy Framework. The Kivy framework provides a way to make graphical applications using bluetooth that run on both android and desktop.

An alternative framework is BeeWare. An implementation for BeeWare would likely be very similar to Python-for-Android, if anybody is interested in contributing one. As of 2020, the major task to tackle is making a custom template to embed Java subclasses of the Bluetooth Android interfaces, for forwarding callbacks.

The Python-for-Android backend classes are found in the `bleak.backends.p4android` package and are automatically selected when building with python-for-android or Buildozer, Kivy's automated build tool.

Considerations on Android

For one thing, the python-for-android backend has not been fully tested. Please run applications with `adb logcat` or `buildozer android logcat` and file issues that include the output, so that any compatibility concerns with devices the developer did not own can be eventually addressed. This backend was originally authored by @xloem for a project that has mostly wrapped up now, so it would be good to tag him in the issues.

When fixing issues, often the Android documentation is lacking, and other resources may need to be consulted to find information on various device quirks, such as community developer forums.

Sometimes device drivers will give off new, undocumented error codes. There is a developing list of these at `bleak.backends.p4android.defs.GATT_STATUS_NAMES`. Please add to the list if you find new status codes, which is indicated by a number being reported instead of a name.

Additionally a few small features are missing. Please file an issue if you need a missing feature, and ideally contribute code, so that soon they will all be implemented.

Two missing features include scanning filters and indications (notifications without replies).

Additionally reading from a characteristic has not been tested at all, as xloem's test device did not provide for this.

On Android, Bluetooth needs permissions for access. These permissions need to be added to the android application in the buildozer.spec file, and are also requested from the user at runtime. This means that enabling bluetooth may not succeed if the user does not accept permissions.

API

Scanner

class bleak.backends.p4android.scanner.**BleakScannerP4Android**(*detection_callback: Optional[Callable[[BLEDevice, AdvertisementData], Optional[Coroutine[Any, Any, None]]]]*, *service_uuids: Optional[List[str]]*, *scanning_mode: Literal['active', 'passive']*, ***kwargs*)

The python-for-android Bleak BLE Scanner.

Parameters

- **detection_callback** – Optional function that will be called each time a device is discovered or advertising data has changed.
- **service_uuids** – Optional list of service UUIDs to filter on. Only advertisements containing this advertising data will be received. Specifying this also enables scanning while the screen is off on Android.
- **scanning_mode** – Set to `"passive"` to avoid the `"active"` scanning mode.

set_scanning_filter(***kwargs*) → None

Set scanning filter for the BleakScanner.

Parameters

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

async start() → None

Start scanning for devices

async stop() → None

Stop scanning for devices

Client

BLE Client for python-for-android

class bleak.backends.p4android.client.**BleakClientP4Android**(*address_or_ble_device: Union[*BLEDevice*, str]*, *services: Optional[Set[UUID]]*, ***kwargs*)

A python-for-android Bleak Client

Parameters

- **address_or_ble_device** – The Bluetooth address of the BLE peripheral to connect to or the BLEDevice object representing it.
- **services** – Optional set of services UUIDs to filter.

async connect(***kwargs*) → bool

Connect to the specified GATT server.

Returns
Boolean representing connection status.

async disconnect() → bool

Disconnect from the specified GATT server.

Returns
Boolean representing if device is disconnected.

async get_services() → *BleakGATTServiceCollection*

Get all services registered for this GATT server.

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

property is_connected: bool

Check connection status between this client and the server.

Returns
Boolean representing connection status.

property mtu_size: Optional[int]

Gets the negotiated MTU.

async 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.

async read_gatt_char(*char_specifier: Union[BleakGATTCharacteristicP4Android, int, str, UUID]*, ***kwargs*) → bytearray

Perform read operation on the specified GATT characteristic.

Parameters

char_specifier (*BleakGATTCharacteristicP4Android, int, str or UUID*) – The characteristic to read from, specified by either integer handle, UUID or directly by the BleakGATTCharacteristicP4Android object representing it.

Returns

(bytearray) The read data.

async read_gatt_descriptor(*desc_specifier: Union[BleakGATTDescriptorP4Android, str, UUID]*, ***kwargs*) → bytearray

Perform read operation on the specified GATT descriptor.

Parameters

desc_specifier (*BleakGATTDescriptorP4Android, str or UUID*) – The descriptor to read from, specified by either UUID or directly by the BleakGATTDescriptorP4Android object representing it.

Returns

(bytearray) The read data.

async start_notify(*characteristic:* BleakGATTCharacteristic, *callback: Callable[[bytearray], None]*, ***kwargs*) → None

Activate notifications/indications on a characteristic.

async stop_notify(*char_specifier: Union[BleakGATTCharacteristicP4Android, int, str, UUID]*) → None

Deactivate notification/indication on a specified characteristic.

Parameters

char_specifier (*BleakGATTCharacteristicP4Android, int, str or UUID*) – The characteristic to deactivate notification/indication on, specified by either integer handle, UUID or directly by the BleakGATTCharacteristicP4Android object representing it.

async unpair() → bool

Unpair with the peripheral.

Returns

Boolean regarding success of unpairing.

async write_gatt_char(*characteristic:* BleakGATTCharacteristic, *data: bytearray*, *response: bool*) → None

Perform a write operation on the specified GATT characteristic.

Parameters

- **characteristic** – The characteristic to write to.
- **data** – The data to send.
- **response** – If write-with-response operation should be done.

async write_gatt_descriptor(*desc_specifier: Union[BleakGATTDescriptorP4Android, str, UUID]*, *data: bytearray*) → None

Perform a write operation on the specified GATT descriptor.

Parameters

- **desc_specifier** (*BleakGATTDescriptorP4Android, str or UUID*) – The descriptor to write to, specified by either UUID or directly by the BleakGATTDescriptorP4Android object representing it.
- **data** (*bytes or bytearray*) – The data to send.

1.4.5 Shared Backend API

Warning: The backend APIs are not considered part of the stable API and may change without notice.

Scanner

class bleak.backends.scanner.**AdvertisementData**(*local_name: Optional[str]*, *manufacturer_data: Dict[int, bytes]*, *service_data: Dict[str, bytes]*, *service_uuids: List[str]*, *tx_power: Optional[int]*, *rssi: int*, *platform_data: Tuple*)

Wrapper around the advertisement data that each platform returns upon discovery

local_name: Optional[str]

The local name of the device or `None` if not included in advertising data.

manufacturer_data: Dict[int, bytes]

Dictionary of manufacturer data in bytes from the received advertisement data or empty dict if not present.

The keys are Bluetooth SIG assigned Company Identifiers and the values are bytes.

https://www.bluetooth.com/specifications/assigned-numbers/company-identifiers/

platform_data: Tuple

Tuple of platform specific data.

This is not a stable API. The actual values may change between releases.

rssi: int

The Radio Receive Signal Strength (RSSI) in dBm.

New in version 0.19.

service_data: Dict[str, bytes]

Dictionary of service data from the received advertisement data or empty dict if not present.

service_uuids: List[str]

List of service UUIDs from the received advertisement data or empty list if not present.

tx_power: Optional[int]

TX Power Level of the remote device from the received advertising data or `None` if not present.

New in version 0.17.

bleak.backends.scanner.**AdvertisementDataCallback**

Type alias for callback called when advertisement data is received.

alias of `Callable[[BLEDevice, AdvertisementData], Optional[Coroutine[Any, Any, None]]]`

bleak.backends.scanner.**AdvertisementDataFilter**

Type alias for an advertisement data filter function.

Implementations should return `True` for matches, otherwise `False`.

alias of `Callable[[BLEDevice, AdvertisementData], bool]`

class bleak.backends.scanner.**BaseBleakScanner**(*detection_callback: Optional[Callable[[*BLEDevice, AdvertisementData*], Optional[Coroutine[Any, Any, None]]]]*, *service_uuids: Optional[List[str]]*)

Interface for Bleak Bluetooth LE Scanners

Parameters

- **detection_callback** – Optional function that will be called each time a device is discovered or advertising data has changed.
- **service_uuids** – Optional list of service UUIDs to filter on. Only advertisements containing this advertising data will be received.

call_detection_callbacks(*device:* BLEDevice, *advertisement_data:* AdvertisementData) → None

Calls all registered detection callbacks.

Backend implementations should call this method when an advertisement event is received from the OS.

create_or_update_device(*address: str*, *name: str*, *details: Any*, *adv:* AdvertisementData) → *BLEDevice*

Creates or updates a device in *seen_devices*.

Parameters

- **address** – The Bluetooth address of the device (UUID on macOS).
- **name** – The OS display name for the device.
- **details** – The platform-specific handle for the device.
- **adv** – The most recent advertisement data received.

Returns

The updated device.

is_allowed_uuid(*service_uuids: Optional[List[str]]*) → bool

Check if the advertisement data contains any of the service UUIDs matching the filter. If no filter is set, this will always return `True`.

Parameters

service_uuids – The service UUIDs from the advertisement data.

Returns

True if the advertisement data should be allowed or False

if the advertisement data should be filtered out.

register_detection_callback(*callback: Optional[Callable[[*BLEDevice, AdvertisementData*], Optional[Coroutine[Any, Any, None]]]]*) → Callable[[], None]

Register a callback that is called when an advertisement event from the OS is received.

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

Parameters

callback – A function, coroutine or `None`.

Returns

A method that can be called to unregister the callback.

seen_devices: Dict[str, Tuple[*BLEDevice*, *AdvertisementData*]]

Map of device identifier to BLEDevice and most recent advertisement data.

This map must be cleared when scanning starts.

abstract set_scanning_filter(***kwargs*) → None

Set scanning filter for the BleakScanner.

Parameters

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

abstract async start() → None

Start scanning for devices

abstract async stop() → None

Stop scanning for devices

bleak.backends.scanner.**get_platform_scanner_backend_type**() → Type[*BaseBleakScanner*]

Gets the platform-specific *BaseBleakScanner* type.

Client

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[*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.

abstract async connect(***kwargs*) → bool

Connect to the specified GATT server.

Returns

Boolean representing connection status.

abstract async disconnect() → bool

Disconnect from the specified GATT server.

Returns

Boolean representing connection status.

abstract async get_services(***kwargs*) → *BleakGATTServiceCollection*

Get all services registered for this GATT server.

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

abstract property is_connected: bool
Check connection status between this client and the server.

Returns
Boolean representing connection status.

abstract property mtu_size: int
Gets the negotiated MTU.

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

abstract async read_gatt_char(*char_specifier: Union[*BleakGATTCharacteristic*, int, str, 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.

abstract async 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[[], None]]*, ***kwargs*) → None
Set the disconnect callback. The callback will only be called on unsolicited disconnect event.

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

Parameters
callback – callback to be called on disconnection.

abstract async start_notify(*characteristic:* BleakGATTCharacteristic, *callback: Callable[[bytearray], None]*, ***kwargs*) → None
Activate notifications/indications on a characteristic.

Implementers should call the OS function to enable notifications or indications on the characteristic.

To keep things the same cross-platform, notifications should be preferred over indications if possible when a characteristic supports both.

abstract async stop_notify(*char_specifier: Union[*BleakGATTCharacteristic*, int, str, 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.

abstract async unpair() → bool

Unpair with the peripheral.

abstract async write_gatt_char(*characteristic:* BleakGATTCharacteristic, *data: Buffer*, *response: bool*) → None

Perform a write operation on the specified GATT characteristic.

Parameters

- **characteristic** – The characteristic to write to.
- **data** – The data to send.
- **response** – If write-with-response operation should be done.

abstract async write_gatt_descriptor(*handle: int*, *data: Buffer*) → None

Perform a write operation on the specified GATT descriptor.

Parameters

- **handle** – The handle of the descriptor to read from.
- **data** – The data to send (any bytes-like object).

bleak.backends.client.**get_platform_client_backend_type**() → Type[*BaseBleakClient*]

Gets the platform-specific *BaseBleakClient* type.

1.5 Troubleshooting

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

1.5.1 Common Mistakes

Calling `asyncio.run()` more than once

Bleak requires the same asyncio run loop to be used for all of its operations. And it requires the loop to always be running because there are background tasks that need to always be running. Therefore, make sure you only call `asyncio.run()` once at the start of your program. **Your program will not work correctly if you call it more than once.** Even if it seems like it is working, crashes and other problems will occur eventually.

DON'T!

```
async def scan():
    return await BleakScanner.find_device_by_name("My Device")

async def connect(device):
    async with BleakClient(device) as client:
        data = await client.read_gatt_char(MY_CHAR_UUID)
        print("received:" data)

# Do not wrap each function call in asyncio.run() like this!
device = asyncio.run(scan())
if not device:
    print("Device not found")
else:
    asyncio.run(connect(device))
```

DO!

```
async def scan():
    return await BleakScanner.find_device_by_name("My Device")

async def connect(device):
    async with BleakClient(device) as client:
        data = await client.read_gatt_char(MY_CHAR_UUID)
        print("received:" data)

# Do have one async main function that does everything.
async def main():
    device = await scan()
    if not device:
        print("Device not found")
        return

    await connect(device)

asyncio.run(main())
```

DON'T!

```
async def scan_and_connect():
    device = await BleakScanner.find_device_by_name("My Device")
    if not device:
        print("Device not found")
        return

    async with BleakClient(device) as client:
        data = await client.read_gatt_char(MY_CHAR_UUID)
        print("received:" data)


while True:
    # Don't call asyncio.run() multiple times like this!
    asyncio.run(scan_and_connect())
    # Never use blocking sleep in an asyncio programs!
    time.sleep(5)
```

DO!

```
async def scan_and_connect():
    device = await BleakScanner.find_device_by_name("My Device")
    if not device:
        print("Device not found")
        return

    async with BleakClient(device) as client:
        data = await client.read_gatt_char(MY_CHAR_UUID)
        print("received:" data)

# Do have one async main function that does everything.
async def main():
```

(continues on next page)

(continued from previous page)

```
    while True:
        await scan_and_connect()
        # Do use asyncio.sleep() in an asyncio program.
        await asyncio.sleep(5)

asyncio.run(main())
```

Naming your script `bleak.py`

Many people name their first script `bleak.py`. This causes the script to crash with an `ImportError` similar to:

```
ImportError: cannot import name 'BleakClient' from partially initialized module 'bleak'␣
↪(most likely due to a circular import) (bleak.py)`
```

To fix the error, change the name of the script to something other than `bleak.py`.

1.5.2 macOS Bugs

Bleak crashes with SIGABRT on macOS

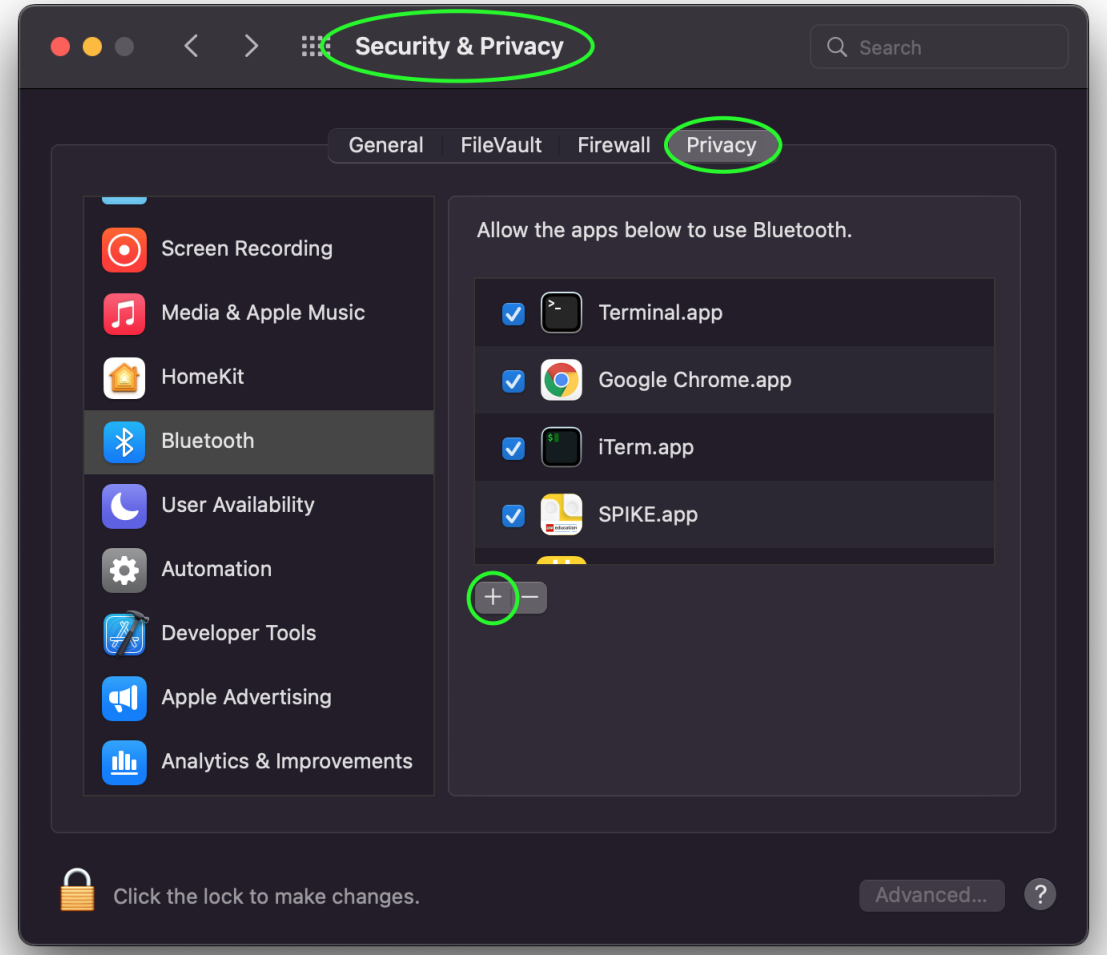
If you see a crash similar to this:

```
Crashed Thread:        1  Dispatch queue: com.apple.root.default-qos

Exception Type:        EXC_CRASH (SIGABRT)
Exception Codes:       0x0000000000000000, 0x0000000000000000
Exception Note:        EXC_CORPSE_NOTIFY

Termination Reason:    Namespace TCC, Code 0
This app has crashed because it attempted to access privacy-sensitive data without a␣
↪usage description. The app's Info.plist must contain an␣
↪NSBluetoothAlwaysUsageDescription key with a string value explaining to the user how␣
↪the app uses this data.
```

It is not a problem with Bleak. It is a problem with your terminal application.

Ideally, the terminal application should be fixed by adding `NSBluetoothAlwaysUsageDescription` to the `Info.plist` file (example).

It is also possible to manually add the app to the list of Bluetooth apps in the *Privacy* settings in the macOS *System Preferences*.



If the app is already in the list but the checkbox for Bluetooth is disabled, you will get the a `BleakError`: "BLE is not authorized - check macOS privacy settings". instead of crashing with `SIGABRT`, in which case you need to check the box to allow Bluetooth for the app that is running Python.

No devices found when scanning on macOS 12

A bug was introduced in macOS 12.0 that causes scanning to not work unless a list of service UUIDs is provided to `BleakScanner`. This bug was fixed in macOS 12.3. On the affected version, users of bleak will see the following error logged:

```
macOS 12.0, 12.1 and 12.2 require non-empty service_uuids kwarg, otherwise no␣
↪advertisement data will be received
```

See #635 and #720 for more information including some partial workarounds if you need to support these macOS versions.

1.5.3 Windows Bugs

Not working when threading model is STA

Packages like `pywin32` and it's subsidiaries have an unfortunate side effect of initializing the threading model to Single Threaded Apartment (STA) when imported. This causes async WinRT functions to never complete if Bleak is being used in a console application (no Windows graphical user interface). This is because there isn't a Windows message loop running to handle async callbacks. Bleak, when used in a console application, needs to run in a Multi Threaded Apartment (MTA) instead (this happens automatically on the first WinRT call).

Bleak should detect this and raise an exception with a message similar to:

```
Thread is configured for Windows GUI but callbacks are not working.
```

You can tell a `pywin32` package caused the issue by checking for `"pythoncom" in sys.modules`. If it is there, then likely it triggered the problem. You can avoid this by setting `sys.coinit_flags = 0` before importing any package that indirectly imports `pythoncom`. This will cause `pythoncom` to use the default threading model (MTA) instead of STA.

Example:

```
import sys
sys.coinit_flags = 0  # 0 means MTA

import win32com  # or any other package that causes the issue
```

If the issue was caused by something other than the `pythoncom` module, there are a couple of other helper functions you can try.

If your program has a graphical user interface and the UI framework *and* it is properly integrated with asyncio *and* Bleak is not running on a background thread then call `allow_sta()` before calling any other Bleak APIs:

```
try:
    from bleak.backends.winrt.util import allow_sta
    # tell Bleak we are using a graphical user interface that has been properly
    # configured to work with asyncio
    allow_sta()
except ImportError:
    # other OSes and older versions of Bleak will raise ImportError which we
    # can safely ignore
    pass
```

The more typical case, though, is that some library has imported something similar to `pythoncom` with the same unwanted side effect of initializing the main thread of a console application to STA. In this case, you can uninitialize the threading model like this:

```
import naughty_module  # this sets current thread to STA :-(

try:
    from bleak.backends.winrt.util import uninitialize_sta

    uninitialize_sta()  # undo the unwanted side effect
except ImportError:
    # not Windows, so no problem
    pass
```

1.5.4 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.

1.5.5 Connecting to multiple devices at the same time

If you're having difficulty connecting to multiple devices, try to do a scan first and pass the returned `BLEDevice` objects to `BleakClient` calls.

Python:

```
import asyncio
from typing import Sequence

from bleak import BleakClient, BleakScanner
from bleak.backends.device import BLEDevice


async def find_all_devices_services():
    devices: Sequence[BLEDevice] = await BleakScanner.discover(timeout=5.0)

    for d in devices:
        async with BleakClient(d) as client:
            print(client.services)


asyncio.run(find_all_devices_services())
```

1.5.6 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

There is a Windows hardware developer package that includes a tool that supports capturing Bluetooth traffic directly in Wireshark.

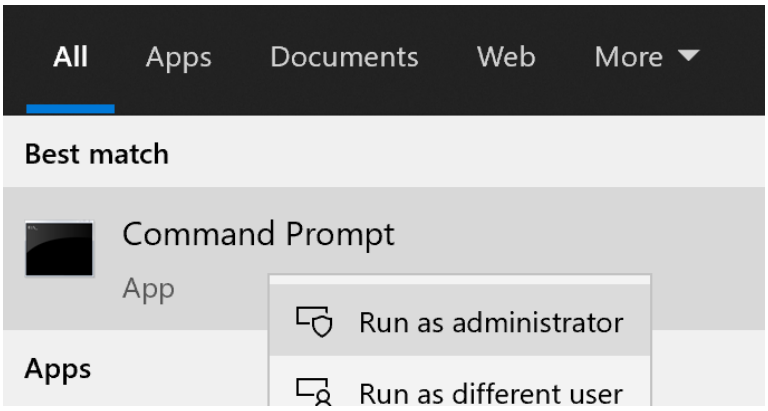
Install

1. Download and install Wireshark.
2. Download and install the BTP software package.

Capture

To capture Bluetooth traffic:

1. Open a terminal as Administrator.
   - Search start menu for `cmd`. (Powershell and Windows Terminal are fine too.)
   - Right-click *Command Prompt* and select *Run as Administrator*.



2. Run `C:\BTP\v1.9.0\x86\btvs.exe`. This should automatically start Wireshark in capture mode.

   Tip: The version needs to match the installed version. `v1.9.0` was the current version at the time this was written. Additionally, `C:` may not be the root drive on some systems.

3. Run your Python script in a different terminal (not as Administrator) to reproduce the problem.
4. Click the stop button in Wireshark to stop the capture.

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. Starting with macOS 14.5, you will need to download and install the Bluetooth logging profile from the Apple develop Profiles and Logs page. Follow the instruction provided in the link, then continue with the steps below.

Tip: After installing the `Bluetooth_macOS.mobileconfig` file, the profile can be found in *System Settings* under *Privacy and Security* > *Others* > *Profiles*. You have to go there to actually install the profile. Then reboot your computer.

If you have an older version of macOS, you can skip this step.

2. Go to the Apple developer More Downloads page and download *Additional Tools for Xcode ...* where ... is the Xcode version corresponding to your macOS version (the XCode version is generally one higher than the macOS version, e.g. XCode 15 for macOS Sonoma 14).
3. Open the disk image and in the *Hardware* folder, double-click the *PacketLogger.app* to run it.
4. Click the *Clear* button in the toolbar to clear the old data.
5. Run your Python script to reproduce the problem.
6. 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.

1.5.7 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.

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 XX:XX:XX:XX:XX:XX
# prior to BlueZ 5.62 you also need to manually delete the GATT cache
sudo rm "/var/lib/bluetooth/YY:YY:YY:YY:YY:YY/cache/XX:XX:XX:XX:XX:XX"
```

…where `XX:XX:XX:XX:XX:XX` is the Bluetooth address of your device and `YY:YY:YY:YY:YY:YY` is the Bluetooth address of the Bluetooth adapter on your computer.

1.6 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:

1.6.1 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 :)

1.6.2 Get Started!

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

You will need Git and Poetry and your favorite text editor. And Python of course.

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

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

3. Set up the Python environment:

```
$ cd bleak/
$ poetry install
```

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 linting and the tests:

```
$ poetry run isort .
$ poetry run black .
$ poetry run flake8
$ poetry run pytest
```

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.

1.6.3 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 3.8+ 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!

1.7 Credits

1.7.1 Development Lead

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

1.7.2 Development Team / Collaborators

- David Lechner <david@pybricks.com>

1.7.3 Contributors

- Chad Spensky <chad@allthenticate.net>
- Bernie Conrad <bernie@allthenticate.net>
- Jonathan Soto <jsotogaviard@alum.mit.edu>
- Kyle J. Williams <kyle@kjwill.tech>
- Edward Betts <edward@4angle.com>
- Robbe Gaeremynck <robbe.gaeremynck@outlook.com>
- David Johansen <davejohansen@gmail.com>
- JP Hutchins <jphutchins@gmail.com>
- Bram Duvigneau <bram@bramd.nl>

1.7.4 Sponsors

- Nabu Casa <https://www.nabucasa.com/>

1.8 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.

1.8.1 Unreleased

1.8.2 0.22.3 (2024-10-05)

Changed

- Don't change ctypes' global state `bleak.backends.winrt.util`.
- Improved performance of BlueZ backend when there are many adapters.
- Added support for Python 3.13.

1.8.3 0.22.2 (2024-06-01)

Changed

- Retrieve the BLE address required by `BleakClientWinRT` from scan response if advertising is None (WinRT).
- Changed type hint for `adv` attribute of `bleak.backends.winrt.scanner._RawAdvData`.
- `BleakGATTCharacteristic.max_write_without_response_size` is now dynamic.

Fixed

- Fixed `discovered_devices_and_advertisement_data` returning devices that should be filtered out by service UUIDs. Fixes #1576.
- Fixed a `Descriptor None was not found!` exception occurring in `start_notify()` on Android. Fixes #823.
- Fixed exception raised when starting `BleakScanner` while running in a Windows GUI app.

1.8.4 0.22.1 (2024-05-07)

Added

- Added `bleak.backends.winrt.util.allow_sta()` method to allow integration with graphical user interfaces on Windows. Fixes #1565.

1.8.5 0.22.0 (2024-05-04)

Added

- Added `BleakCharacteristicNotFoundError` which is raised if a device does not support a characteristic.
- Added utility function to work around `pywin32` setting threading model to STA on Windows.

Changed

- Updated PyObjC dependency on macOS to v10.x.
- Updated missing Bluetooth SIG characteristics and service UUIDs.
- Updated `BlueZManager` to remove empty interfaces from *_properties* during InterfacesRemoved message.
- Updated PyWinRT dependency to v2. Fixes #1529.
- Raise exception when trying to scan while in a single-treaded apartment (STA) on Windows. Fixes #1132.

Fixed

- Fixed BlueZ version in passive scanning error message. Fixes #1433.
- Fixed mypy requiring `Unpack[ExtraArgs]` that were intended to be optional. Fixes #1487.
- Fixed `KeyError` in BlueZ `is_connected()` and `get_global_bluez_manager()` when device is not present. Fixes #1507.
- Fixed BlueZ `_wait_removed` completion on invalid object path. Fixes #1489.
- Fixed rare unhandled exception when scanning on macOS when using `use_bdaddr`. Fixes #1523.
- Fixed scanning silently failing on Windows when Bluetooth is off. Fixes #1535.
- Fixed using wrong value for `tx_power` in Android backend. Fixes #1532.
- Fixed 4-character UUIDs not working on `BleakClient.*_gatt_char` methods. Fixes #1498.
- Fixed race condition with getting max PDU size on Windows. Fixes #1497. [REVERTED in v0.22.2]
- Fixed filtering advertisement data by service UUID when multiple apps are scanning. Fixes #1534.

1.8.6 0.21.1 (2023-09-08)

Changed

- Changed `dbus-fast` dependency to include v2.x. Fixes #1412.

1.8.7 0.21.0 (2023-09-02)

Added

- Added `bleak.uuids.normalize_uuid_16()` function.
- Added `bleak.uuids.normalize_uuid_32()` function.
- Added `advertisement_data()` async iterator method to `BleakScanner`. Merged #1361.
- Added type hints for kwargs on `BleakScanner` class methods.
- Added support for Python 3.12.

Changed

- Improved error messages when failing to get services in WinRT backend.
- Improved error messages with enum values in WinRT backend. Fixes #1284.
- Scanner backends modified to allow multiple advertisement callbacks. Merged #1367.
- Changed default handling of the `response` argument in `BleakClient.write_gatt_char`. Fixes #909.
- Bleak recipe now automatically installs bleak from GitHub release in Kivy example.
- Changed *BlueZManager* methods to raise *BleakError* when device is not in BlueZ.
- Optimized BlueZ backend device watchers and condition callbacks to avoid linear searches.
- Changed type hint for buffer protocol to `collections.abc.Buffer`.

Fixed

- Fixed handling all access denied errors when enumerating characteristics on Windows. Fixes #1291.
- Added support for 32bit UUIDs. Fixes #1314.
- Fixed typing for `BaseBleakScanner` detection callback.
- Fixed possible crash in `_stopped_handler()` in WinRT backend. Fixes #1330.
- Reduced expensive logging in the BlueZ backend. Merged #1376.
- Fixed race condition with `"InterfaceRemoved"` when getting services in BlueZ backend.
- Fixed missing permissions and requirements in android Kivy example. Fixes #1184.
- Fixed WinRT backend sometimes hanging forever when a device goes out of range during connection. Fixes #1359.

Removed

Dropped support for Python 3.7.

1.8.8 0.20.2 (2023-04-19)

Fixed

- Fixed `org.bluez.Error.InProgress` in characteristic and descriptor read and write methods in BlueZ backend.
- Fixed `OSError: [WinError -2147483629] The object has been closed` when connecting on Windows. Fixes #1280.

1.8.9 0.20.1 (2023-03-24)

Fixed

- Fixed possible garbage collection of running async callback from `BleakClient.start_notify()`.
- Fixed possible garbage collection of running async callback from `BleakScanner(detection_callback=)`.
- Fixed possible garbage collection of disconnect monitor in BlueZ backend. Fixed #1258.

1.8.10 0.20.0 (2023-03-17)

Added

- Added `BLEAK_DBUS_AUTH_UID` environment variable for hardcoding D-Bus UID. Merged #1182.
- Added return type `None` to some scanner methods.
- Added optional hack to use Bluetooth address instead of UUID on macOS. Merged #1073.
- Added `BleakScanner.find_device_by_name()` class method.
- Added optional command line argument to use debug log level to all applicable examples.
- Added `bleak.uuids.normalize_uuid_str()` function.
- Added optional `services` argument to `BleakClient()` to filter services. Merged #654.
- Added automatic retry on `le-connection-abort-by-local` in BlueZ backend. Fixes #1220.

Changed

- Dropped `async-timeout` dependency on Python >= 3.11.
- Deprecated `BLEDevice.rssi` and `BLEDevice.metadata`. Fixes #1025.
- `BLEDevice` now uses `__slots__` to reduce memory usage. Merged #1117.
- `BaseBleakClient.services` is now `None` instead of empty service collection until services are discovered.
- Include thread name in `BLEAK_LOGGING` output. Merged #1144.
- Updated PyObjC dependency on macOS to v9.x.

Fixed

- Fixed invalid UTF-8 in `uuids.uuid16_dict`.
- Fixed `AttributeError` in `_ensure_success` in WinRT backend.
- Fixed `BleakScanner.stop()` can raise `BleakDBusError` with `org.bluez.Error.NotReady` in BlueZ backend.
- Fixed `BleakScanner.stop()` hanging in WinRT backend when Bluetooth is disabled.
- Fixed leaking services when `get_services()` is cancelled in WinRT backend.
- Fixed disconnect monitor task not always cancelled on the BlueZ client. Merged #1159.
- Fixed WinRT scanner never calling `detection_callback` when a device does not send a scan response. Fixes #1211.
- Fixed `BLEDevice` name sometimes incorrectly `None`.
- Fixed unhandled exception in `CentralManagerDelegate` destructor on macOS. Fixes #1219.
- Fixed object passed to `disconnected_callback` is not `BleakClient`. Fixes #1200.

1.8.11 0.19.5 (2022-11-19)

Fixed

- Fixed more issues with getting services in WinRT backend.

1.8.12 0.19.4 (2022-11-06)

Fixed

- Fixed `TypeError` in WinRT backend introduced in v0.19.3.

1.8.13 0.19.3 (2022-11-06)

Fixed

- Fixed `TimeoutError` when connecting to certain devices with WinRT backend. Fixes #604.

1.8.14 0.19.2 (2022-11-06)

Fixed

- Fixed crash when getting services in WinRT backend in Python 3.11. Fixes #1112.
- Fixed cache mode when retrying get services in WinRT backend. Merged #1102.
- Fixed `KeyError` crash in BlueZ backend when removing non-existent property. Fixes #1107.

1.8.15 0.19.1 (2022-10-29)

Fixed

- Fixed crash in Android backend introduced in v0.19.0. Fixes #1085.
- Fixed service discovery blocking forever if device disconnects in BlueZ backend. Merged #1092.
- Fixed `AttributeError` crash when scanning on Windows builds < 19041. Fixes #1094.

1.8.16 0.19.0 (2022-10-13)

Added

- Added support for Python 3.11. Merged #990.
- Added better error message for Bluetooth not authorized on macOS. Merged #1033.
- Added `BleakDeviceNotFoundError` which should is raised if a device can not be found by `connect`, `pair` and `unpair`. Merged #1022.
- Added `rssi` attribute to `AdvertisementData`. Merged #1047.
- Added `BleakScanner.discovered_devices_and_advertisement_data` property. Merged #1047.
- Added `return_adv` argument to `BleakScanner.discover` method. Merged #1047.
- Added `BleakClient.unpair()` implementation for BlueZ backend. Merged #1067.

Changed

- Changed `AdvertisementData` to a named tuple. Merged #1047.
- A faster `unpack_variants` is now provided by dbus-fast. Merged #1055.

Fixed

- On BlueZ, support creating additional instances running on a different event loops (i.e. multiple pytest-asyncio cases). Merged #1034.
- Fixed unhandled exception in `max_pdu_size_changed_handler` in WinRT backend. Fixes #1039.
- Fixed stale services in WinRT backend causing `WinError -2147483629`. Fixes #1061.

Removed

Removed `bleak.__version__`. Use `importlib.metadata.version('bleak')` instead.

1.8.17 0.18.1 (2022-09-25)

Fixed

- Reverted unintentional breaking parameter name changes. Fixes #1028.

1.8.18 0.18.0 (2022-09-23)

Changed

- Relaxed `async-timeout` dependency version to support different installations. Merged #1009.
- `BleakClient.unpair()` in WinRT backend can be called without being connected first. Merged #1012.
- Use relative imports internally. Merged #1007.
- `BleakScanner` and `BleakClient` are now concrete classes. Fixes #582.
- Deprecated `BleakScanner.register_detection_callback()`.
- Deprecated `BleakScanner.set_scanning_filter()`.
- Deprecated `BleakClient.set_disconnected_callback()`.
- Deprecated `BleakClient.get_services()`.
- Refactored common code in `BleakClient.start_notify()`.
- (BREAKING) Changed notification callback argument from `int` to `BleakGattCharacteristic`. Fixes #759.

Fixed

- Fixed `tx_power` not included in `AdvertisementData.__repr__` when 0. Merged #1017.

1.8.19 0.17.0 (2022-09-12)

Added

- `AdvertisementData` class now has an attribute `tx_power`. Merged #987.

Changed

- `BleakClient` methods now raise `BleakError` if called when not connected in WinRT backend. Merged #973.
- Extended disconnect timeout to 120 seconds in WinRT backend. Fixes #807.
- Changed version check for BlueZ battery workaround to exclude versions >= 5.55. Merged #976.
- Use Poetry for build system and dependencies. Merged #978.
- The BlueZ D-Bus backend implements a services cache between connections to significancy improve reconnect performance. To use the cache, call `connect` and `get_services` with the `dangerous_use_bleak_cache` argument to avoid services being resolved again. Merged #923.
- The BlueZ D-Bus backend now uses `dbus-fast` package instead of `dbus-next` which significantly improves performance. Merged #988.
- The BlueZ D-Bus backend will not avoid trying to connect to devices that are already connected. Fixes #992.

- Updated logging to lazy version and replaced format by f-string for `BleakClientWinRT`. #1000.
- Added deprecation warning to `discover()` method. Merged #1005.
- BlueZ adapter is chosen dynamically if not provided, instead of using hardcoded "hci0". Fixes #513.

Fixed

- Fixed wrong error message for BlueZ "Operation failed with ATT error". Merged #975.
- Fixed possible `AttributeError` when enabling notifications for battery service in BlueZ backend. Merged #976.
- Fixed use of wrong enum in unpair function of WinRT backend. Merged #986.
- Fixed inconsistent return types for `properties` and `descriptors` properties of `BleakGATTCharacteristic`. Merged #989.
- Handle device being removed before `GetManagedObjects` returns in BlueZ backend. Fixes #996.
- Fixed crash in `max_pdu_size_changed_handler` in WinRT backend. Fixes #998.
- Fixes a race in the BlueZ D-Bus backend where the disconnect monitor would be removed before it could be awaited. Merged #999.

Removed

- Removed `BLEDeviceCoreBluetooth` type from CoreBluetooth backend. Merged #977.

1.8.20 0.16.0 (2022-08-31)

Added

- Added `BleakGattCharacteristic.max_write_without_response_size` property. Fixes #738.

Fixed

- Fixed regression in v0.15 where devices removed from BlueZ while scanning were still listed in `BleakScanner.discovered_devices`. Fixes #942.
- Fixed possible bad connection state in BlueZ backend. Fixes #951.

Changed

- Made BlueZ D-Bus signal callback logging lazy to improve performance. Merged #912.
- Switch to using `async_timeout` instead of `asyncio.wait_for for performance`. Merged #916.
- Improved performance of `BlueZManager.get_services()`. Fixes #927.

Removed

- Removed explicit inheritance from object in class declarations. Merged #922.
- Removed first seen filter in `BleakScanner` detection callbacks on BlueZ backend. Merged #964.

1.8.21 0.15.1 (2022-08-03)

Fixed

- The global BlueZ manager now disconnects correctly on exception. Merged #918.
- Handle the race in the BlueZ D-Bus backend where the device disconnects during the connection process which presented as `Failed to cancel connection`. Merged #919.
- Ensure the BlueZ D-Bus scanner can reconnect after DBus disconnection. Merged #920.
- Adjust default timeout for `read_gatt_char()` with CoreBluetooth to 20s. Fixes #926.

1.8.22 0.15.0 (2022-07-29)

Added

- Added new `assigned_numbers` module and `AdvertisementDataType` enum.
- Added new `bluez` kwarg to `BleakScanner` in BlueZ backend.
- Added support for passive scanning in the BlueZ backend. Fixes #606.
- Added option to use cached services, characteristics and descriptors in WinRT backend. Fixes #686.
- Added `PendingDeprecationWarning` to use of `address_type` as keyword argument. It will be moved into the `winrt` keyword instead according to #623.
- Added better error message when adapter is not present in BlueZ backend. Fixes #889.

Changed

- Add `py.typed` file so mypy discovers Bleak's type annotations.
- UUID descriptions updated to 2022-03-16 assigned numbers document.
- Replace use of deprecated `asyncio.get_event_loop()` in Android backend.
- Adjust default timeout for `read_gatt_char()` with CoreBluetooth to 10s. Merged #891.
- `BleakScanner()` args `detection_callback` and `service_uuids` are no longer keyword-only.
- `BleakScanner()` arg `scanning_mode` is no longer Windows-only and is no longer keyword-only.
- All `BleakScanner()` instances in BlueZ backend now use common D-Bus object manager.
- Deprecated `filters` kwarg in `BleakScanner` in BlueZ backend.
- BlueZ version is now checked on first connection instead of import to avoid import side effects. Merged #907.

Fixed

- Documentation fixes.
- On empty characteristic description from WinRT, use the lookup table instead of returning empty string.
- Fixed detection of first advertisement in BlueZ backend. Merged #903.
- Fixed performance issues in BlueZ backend caused by calling "GetManagedObjects" each time a `BleakScanner` scans or `BleakClient` is connected. Fixes #500.
- Fixed not handling "InterfacesRemoved" in `BleakClient` in BlueZ backend. Fixes #882.
- Fixed leaking D-Bus socket file descriptors in BlueZ backend. Fixes #805.

Removed

- Removed fallback to call "ConnectDevice" when "Connect" fails in Bluez backend. Fixes #806.

1.8.23 0.14.3 (2022-04-29)

Changed

- Suppress macOS 12 scanner bug error message for macOS 12.3 and higher. Fixes #720.
- Added filters `Discoverable` and `Pattern` to BlueZ D-Bus scanner. Fixes #790.

Fixed

- Fixed reading the battery level returns a zero-filled `bytearray` on BlueZ >= 5.48. Fixes #750.
- Fixed unpairing does not work on windows with WinRT. Fixes #699
- Fixed leak of `_disconnect_futures` in `CentralManagerDelegate`.
- Fixed callback not removed from `_disconnect_callbacks` on disconnect in `CentralManagerDelegate`.

1.8.24 0.14.2 (2022-01-26)

Changed

- Updated `bleak-winrt` dependency to v1.1.1. Fixes #741.

Fixed

- Fixed `name` is `'Unknown'` in WinRT backend. Fixes #736.

1.8.25 0.14.1 (2022-01-12)

Fixed

- Fixed `AttributeError` when passing `BLEDevice` to `BleakClient` constructor on WinRT backend. Fixes #731.

1.8.26 0.14.0 (2022-01-10)

Added

- Added `service_uuids` kwarg to `BleakScanner`. This can be used to work around issue of scanning not working on macOS 12. Fixes #230. Works around #635.
- Added UUIDs for LEGO Powered Up Smart Hubs.

Changed

- Changed WinRT backend to use GATT session status instead of actual device connection status.
- Changed handling of scan response data on WinRT backend. Advertising data and scan response data is now combined in callbacks like other platforms.
- Updated `bleak-winrt` dependency to v1.1.0. Fixes #698.

Fixed

- Fixed `InvalidStateError` in CoreBluetooth backend when read and notification of the same characteristic are used. Fixes #675.
- Fixed reading a characteristic on CoreBluetooth backend also triggers notification callback.
- Fixed in Linux, scanner callback not setting metadata parameters. Merged #715.

1.8.27 0.13.0 (2021-10-20)

Added

- Allow 16-bit UUID string arguments to `get_service()` and `get_characteristic()`.
- Added `register_uuids()` to augment the uuid-to-description mapping.
- Added support for Python 3.10.
- Added `force_indicate` keyword argument for WinRT backend client's `start_notify` method. Fixes #526.
- Added python-for-android backend.

Changed

- Changed from `winrt` dependency to `bleak-winrt`.
- Improved error when connecting to device fails in WinRT backend. Fixes #647.
- Changed examples to use `asyncio.run()`.
- Changed the default notify method for the WinRT backend from `Indicate` to `Notify`.
- Refactored GATT error handling in WinRT backend.
- Changed Windows Bluetooth packet capture instructions. Fixes #653.
- Replaced usage of deprecated `@abc.abstractproperty`.
- Use `asyncio.get_running_loop()` instead of `asyncio.get_event_loop()`.
- Changed "service is already present" exception to logged error in BlueZ backend. Merged #622.

Removed

- Removed `dotnet` backend.
- Dropped support for Python 3.6.
- Removed `use_cached` kwarg from `BleakClient` `connect()` and `get_services()` methods. Fixes #646.

Fixed

- Fixed unused timeout in the implementation of BleakScanner's `find_device_by_address()` function.
- Fixed BleakClient ignoring the `adapter` kwarg. Fixes #607.
- Fixed writing descriptors in WinRT backend. Fixes #615.
- Fixed race on disconnect and cleanup of BlueZ matches when device disconnects early. Fixes #603.
- Fixed memory leaks on Windows.
- Fixed protocol error code descriptions on WinRT backend. Fixes #532.
- Fixed race condition hitting assentation in BlueZ `disconnect()` method. Fixes #641.
- Fixed enumerating services on a device with HID service on WinRT backend. Fixes #599.
- Fixed subprocess running to check BlueZ version each time a client is created. Fixes #602.
- Fixed exception when discovering services after reconnecting in CoreBluetooth backend.

1.8.28 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.

1.8.29 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.

1.8.30 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.

1.8.31 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.

1.8.32 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.

1.8.33 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.

1.8.34 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.

1.8.35 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)

1.8.36 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.

1.8.37 0.6.4 (2020-05-20)

Fixed

- Fix for bumpversion usage

1.8.38 0.6.3 (2020-05-20)

Added

- Building and releasing from Github Actions

Removed

- Building and releasing on Azure Pipelines

1.8.39 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

1.8.40 0.6.1 (2020-03-09)

Fixed

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

1.8.41 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_disconnected_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.

1.8.42 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

1.8.43 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

1.8.44 0.4.3 (2019-06-30)

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

1.8.45 0.4.2 (2019-05-17)

- Fix for missed part of PR #61.

1.8.46 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.

1.8.47 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

1.8.48 0.3.0 (2019-03-18)

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

1.8.49 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.

1.8.50 0.2.3 (2018-11-28)

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

1.8.51 0.2.2 (2018-11-08)

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

1.8.52 0.2.1 (2018-06-28)

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

1.8.53 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.

1.8.54 0.1.0 (2017-10-23)

- Bleak created.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

b

INDEX

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

W