Adafruit BLE Library Documentation

Release 1.0

Dan Halbert

Mar 17, 2020

Contents

This module provides higher-level BLE (Bluetooth Low Energy) functionality, building on the native `_bleio` module.

CHAPTER 1

Dependencies

This driver depends on:

- Adafruit CircuitPython

Please ensure all dependencies are available on the CircuitPython filesystem. This is easily achieved by downloading the Adafruit library and driver bundle.

CHAPTER 2

Installing from PyPI

Warning: Linux support is **very** limited. See Adafruit Blinka _bleio for details.

On supported GNU/Linux systems like the Raspberry Pi, you can install the driver locally from PyPI. To install for current user:

```
pip3 install adafruit-circuitpython-ble
```

To install system-wide (this may be required in some cases):

```
sudo pip3 install adafruit-circuitpython-ble
```

To install in a virtual environment in your current project:

```
mkdir project-name && cd project-name
python3 -m venv .env
source .env/bin/activate
pip3 install adafruit-circuitpython-ble
```

CHAPTER 3

Usage Example

```
from adafruit_ble import BLERadio

radio = BLERadio()
print("scanning")
found = set()
for entry in radio.start_scan(timeout=60, minimum_rssi=-80):
    addr = entry.address
    if addr not in found:
        print(entry)
    found.add(addr)

print("scan done")
```

CHAPTER 4

Contributing

Contributions are welcome! Please read our Code of Conduct before contributing to help this project stay welcoming.

CHAPTER 5

Documentation

For information on building library documentation, please check out this guide.

CHAPTER 6

Table of Contents

6.1 Scan everything

Ensure your device works with this simple test. When working, will print out advertising data of nearby BLE devices.

Listing 1: examples/ble_simpletest.py

```
"""
This example scans for any BLE advertisements and prints one advertisement and one scan response
from every device found.
"""

from adafruit_ble import BLERadio

ble = BLERadio()
print("scanning")
found = set()
scan_responses = set()
for advertisement in ble.start_scan():
    addr = advertisement.address
    if advertisement.scan_response and addr not in scan_responses:
        scan_responses.add(addr)
    elif not advertisement.scan_response and addr not in found:
        found.add(addr)
    else:
        continue
    print(addr, advertisement)
    print("\t" + repr(advertisement))
    print()

print("scan done")
```

6.2 Detailed scan

Ensure your device works with this simple test.

Listing 2: examples/ble_detailed_scan.py

```
# This example scans for any BLE advertisements and prints one advertisement and one scan response
# from every device found. This scan is more detailed than the simple test because it includes
# specialty advertising types.

from adafruit_ble import BLERadio

from adafruit_ble.advertising import Advertisement
from adafruit_ble.advertising.standard import ProvideServicesAdvertisement

ble = BLERadio()
print("scanning")
found = set()
scan_responses = set()
# By providing Advertisement as well we include everything, not just specific advertisements.
for advertisement in ble.start_scan(ProvideServicesAdvertisement, Advertisement):
    addr = advertisement.address
    if advertisement.scan_response and addr not in scan_responses:
        scan_responses.add(addr)
    elif not advertisement.scan_response and addr not in found:
        found.add(addr)
    else:
        continue
    print(addr, advertisement)
    print("\t" + repr(advertisement))
    print()

print("scan done")
```

6.3 adafruit_ble

This module provides higher-level BLE (Bluetooth Low Energy) functionality, building on the native _bleio module.

class BLEConnection(*bleio_connection*)

Represents a connection to a peer BLE device. It acts as a map from a *Service* type to a *Service* instance for the connection.

Parameters **_bleio.Connection** (*bleio_connection*) – the native _bleio.Connection object to wrap

connected

True if the connection to the peer is still active.

paired

True if the paired to the peer.

connection_interval

Time between transmissions in milliseconds. Will be multiple of 1.25ms. Lower numbers increase speed and decrease latency but increase power consumption.

When setting connection_interval, the peer may reject the new interval and *connection_interval* will then remain the same.

Apple has additional guidelines that dictate should be a multiple of 15ms except if HID is available. When HID is available Apple devices may accept 11.25ms intervals.

pair(*, *bond=True*)
Pair to the peer to increase security of the connection.

disconnect()
Disconnect from peer.

class BLERadio(*adapter=None*)
BLERadio provides the interfaces for BLE advertising, scanning for advertisements, and connecting to peers. There may be multiple connections active at once.

It uses this library's *Advertisement* classes and the *BLEConnection* class.

start_advertising(*advertisement*, *scan_response=None*, *interval=0.1*)
Starts advertising the given advertisement.

Parameters

- **scan_response** (*buf*) – scan response data packet bytes. If `None`, a default scan response will be generated that includes *BLERadio.name* and *BLERadio.tx_power*.
- **interval** (*float*) – advertising interval, in seconds

stop_advertising()
Stops advertising.

start_scan(**advertisement_types*, *buffer_size=512*, *extended=False*, *timeout=None*, *interval=0.1*, *window=0.1*, *minimum_rssi=-80*, *active=True*)
Starts scanning. Returns an iterator of advertisement objects of the types given in advertisement_types. The iterator will block until an advertisement is heard or the scan times out.

If any `advertisement_types` are given, only Advertisements of those types are produced by the returned iterator. If none are given then *Advertisement* objects will be returned.

Advertisements and scan responses are filtered and returned separately.

Parameters

- **buffer_size** (*int*) – the maximum number of advertising bytes to buffer.
- **extended** (*bool*) – When True, support extended advertising packets. Increasing buffer_size is recommended when this is set.
- **timeout** (*float*) – the scan timeout in seconds. If None, will scan until *stop_scan* is called.
- **interval** (*float*) – the interval (in seconds) between the start of two consecutive scan windows Must be in the range 0.0025 - 40.959375 seconds.
- **window** (*float*) – the duration (in seconds) to scan a single BLE channel. window must be <= interval.
- **minimum_rssi** (*int*) – the minimum rssi of entries to return.
- **active** (*bool*) – request and retrieve scan responses for scannable advertisements.

Returns If any `advertisement_types` are given, only Advertisements of those types are produced by the returned iterator. If none are given then *Advertisement* objects will be returned.

Return type iterable

stop_scan()
Stops any active scan.

The scan results iterator will return any buffered results and then raise StopIteration once empty.

connect(*advertisement*, *, *timeout=4*)
Initiates a *BLEConnection* to the peer that advertised the given advertisement.

Parameters

- **Advertisement** (*advertisement*) – An *Advertisement* or a subclass of *Advertisement*
- **float** (*timeout*) – how long to wait for a connection

Returns the connection to the peer

Return type *BLEConnection*

connected
True if any peers are connected.

connections
A tuple of active *BLEConnection* objects.

name
The name for this device. Used in advertisements and as the Device Name in the Generic Access Service, available to a connected peer.

tx_power
Transmit power, in dBm.

address_bytes
The device address, as a `bytes()` object of length 6.

6.3.1 advertising

Advertising is the first phase of BLE where devices can broadcast

to_hex(*seq*)
Pretty prints a byte sequence as hex values.

to_bytes_literal(*seq*)
Prints a byte sequence as a Python bytes literal that only uses hex encoding.

decode_data(*data*, *, *key_encoding='B'*)
Helper which decodes length encoded structures into a dictionary with the given key encoding.

compute_length(*data_dict*, *, *key_encoding='B'*)
Computes the length of the encoded data dictionary.

encode_data(*data_dict*, *, *key_encoding='B'*)
Helper which encodes dictionaries into length encoded structures with the given key encoding.

class AdvertisingDataField
Top level class for any descriptor classes that live in Advertisement or its subclasses.

class AdvertisingFlag(*bit_position*)
A single bit flag within an AdvertisingFlags object.

class AdvertisingFlags(*advertisement*, *advertising_data_type*)
Standard advertising flags

limited_discovery
Discoverable only for a limited time period.

general_discovery
Will advertise until discovered.

le_only
BR/EDR not supported.

class String(*, *advertising_data_type*)
UTF-8 encoded string in an Advertisement.

Not null terminated once encoded because length is always transmitted.

class Struct(*struct_format*, *, *advertising_data_type*)
`struct` encoded data in an Advertisement.

class LazyObjectField(*cls*, *attribute_name*, *, *advertising_data_type*, **kwargs*)
Non-data descriptor useful for lazily binding a complex object to an advertisement object.

advertising_data_type
Return the data type value used to indicate this field.

class Advertisement
Core Advertisement type

short_name
Short local device name (shortened to fit).

complete_name
Complete local device name.

tx_power
Transmit power level

appearance
Appearance.

classmethod from_entry(*entry*)
Create an Advertisement based on the given ScanEntry. This is done automatically by *BLERadio* for all scan results.

rssi
Signal strength of the scanned advertisement. Only available on Advertisements returned from *BLERadio.start_scan()*. (read-only)

classmethod matches(*entry*)
Returns true if the given `_bleio.ScanEntry` matches all portions of the Advertisement type's prefix.

6.3.1.1 standard

This module provides BLE standard defined advertisements. The Advertisements are single purpose even though multiple purposes may actually be present in a single packet.

class BoundServiceList(*advertisement*, *, *standard_services*, *vendor_services*)
Sequence-like object of Service UUID objects. It stores both standard and vendor UUIDs.

append(*service*)
Append a service to the list.

extend(*services*)
Appends all services in the iterable to the list.

class ServiceList(*, *standard_services*, *vendor_services*)
Descriptor for a list of Service UUIDs that lazily binds a corresponding BoundServiceList.

class ProvideServicesAdvertisement(**services*)
Advertise what services that the device makes available upon connection.

services
List of services the device can provide.

classmethod matches(*entry*)
Returns true if the given `_bleio.ScanEntry` matches all portions of the Advertisement type's prefix.

class SolicitServicesAdvertisement(**services*)
Advertise what services the device would like to use over a connection.

solicited_services
List of services the device would like to use.

class ManufacturerData(*obj*, *, *advertising_data_type=255*, *company_id*, *key_encoding='B'*)
Encapsulates manufacturer specific keyed data bytes. The manufacturer is identified by the company_id and the data is structured like an advertisement with a configurable key format.

class ManufacturerDataField(*key*, *value_format*, *field_names=None*)
A single piece of data within the manufacturer specific data. The format can be repeated.

class ServiceData(*service*)
Encapsulates service data. It is read as a memoryview which can be manipulated or set as a bytearray to change the size.

6.3.1.2 adafruit

This module provides Adafruit defined advertisements.

Adafruit manufacturing data is key encoded like advertisement data and the Apple manufacturing data. However, the keys are 16-bits to enable many different uses. Keys above 0xf000 can be used by Adafruit customers for their own data.

class AdafruitColor
Broadcast a single RGB color.

color
Color to broadcast as RGB integer.

6.3.2 attributes

6.3.2.1 attributes

This module provides definitions common to all kinds of BLE attributes, specifically characteristics and descriptors.

class Attribute
Constants describing security levels.

NO_ACCESS
security mode: access not allowed

OPEN
security_mode: no security (link is not encrypted)

ENCRYPT_NO_MITM
security_mode: unauthenticated encryption, without man-in-the-middle protection

ENCRYPT_WITH_MITM
security_mode: authenticated encryption, with man-in-the-middle protection

LESC_ENCRYPT_WITH_MITM
security_mode: LESC encryption, with man-in-the-middle protection

SIGNED_NO_MITM
security_mode: unauthenticated data signing, without man-in-the-middle protection

SIGNED_WITH_MITM
security_mode: authenticated data signing, without man-in-the-middle protection

6.3.3 characteristics

This module provides core BLE characteristic classes that are used within Services.

class Characteristic(*, *uuid=None*, *properties=0*, *read_perm=0*, *write_perm=0*, *max_length=None*, *fixed_length=False*, *initial_value=None*)
Top level Characteristic class that does basic binding.

Parameters

- **uuid** (*UUID*) – The uuid of the characteristic
- **properties** (*int*) – The properties of the characteristic, specified as a bitmask of these values bitwise-or'd together: *BROADCAST*, *INDICATE*, *NOTIFY*, *READ*, *WRITE*, *WRITE_NO_RESPONSE*.
- **read_perm** (*int*) – Specifies whether the characteristic can be read by a client, and if so, which security mode is required. Must be one of the integer values *Attribute.NO_ACCESS*, *Attribute.OPEN*, *Attribute.ENCRYPT_NO_MITM*, *Attribute.ENCRYPT_WITH_MITM*, *Attribute.LESC_ENCRYPT_WITH_MITM*, *Attribute.SIGNED_NO_MITM*, or *Attribute.SIGNED_WITH_MITM*.
- **write_perm** (*int*) – Specifies whether the characteristic can be written by a client, and if so, which security mode is required. Values allowed are the same as `read_perm`.
- **max_length** (*int*) – Maximum length in bytes of the characteristic value. The maximum allowed is 512, or possibly 510 if `fixed_length` is False. The default, 20, is the maximum number of data bytes that fit in a single BLE 4.x ATT packet.
- **fixed_length** (*bool*) – True if the characteristic value is of fixed length.
- **initial_value** (*buf*) – The initial value for this characteristic. If not given, will be filled with zeros.

BROADCAST
property: allowed in advertising packets

INDICATE
property: server will indicate to the client when the value is set and wait for a response

NOTIFY
property: server will notify the client when the value is set

READ
property: clients may read this characteristic

WRITE
property: clients may write this characteristic; a response will be sent back

WRITE_NO_RESPONSE
property: clients may write this characteristic; no response will be sent back

class ComplexCharacteristic(*, *uuid=None*, *properties=0*, *read_perm=0*, *write_perm=0*, *max_length=20*, *fixed_length=False*, *initial_value=None*)
Characteristic class that does complex binding where the subclass returns a full object for interacting with the characteristic data. The Characteristic itself will be shadowed once it has been bound to the corresponding instance attribute.

bind(*service*)
Binds the characteristic to the local Service or remote Characteristic object given.

class StructCharacteristic(*struct_format*, *, *uuid=None*, *properties=0*, *read_perm=0*, *write_perm=0*, *initial_value=None*)
Data descriptor for a structure with a fixed format.

Parameters

- **struct_format** – a `struct` format string describing how to pack multiple values into the characteristic bytestring
- **uuid** (`UUID`) – The uuid of the characteristic
- **properties** (*int*) – see *Characteristic*
- **read_perm** (*int*) – see *Characteristic*
- **write_perm** (*int*) – see *Characteristic*
- **initial_value** (*buf*) – see *Characteristic*

6.3.3.1 int

This module provides integer characteristics that are usable directly as attributes.

class IntCharacteristic(*format_string*, *min_value*, *max_value*, *, *uuid=None*, *properties=0*, *read_perm=0*, *write_perm=0*, *initial_value=None*)
Superclass for different kinds of integer fields.

class Int8Characteristic(*, *min_value=-128*, *max_value=127*, ***kwargs*)
Int8 number.

class Uint8Characteristic(*, *min_value=0*, *max_value=255*, ***kwargs*)
Uint8 number.

class Int16Characteristic(*, *min_value=-32768*, *max_value=32767*, ***kwargs*)
Int16 number.

class Uint16Characteristic(*, *min_value=0*, *max_value=65535*, ***kwargs*)
Uint16 number.

class Int32Characteristic(*, *min_value=-2147483648*, *max_value=2147483647*, ***kwargs*)
Int32 number.

class Uint32Characteristic(*, *min_value=0*, *max_value=4294967295*, ***kwargs*)
Uint32 number.

6.3.3.2 stream

This module provides stream characteristics that bind readable or writable objects to the Service object they are on.

class BoundWriteStream(*bound_characteristic*)
Writes data out to the peer.

write(*buf*)
Write data from buf out to the peer.

class StreamOut(*, *uuid=None*, *timeout=1.0*, *buffer_size=64*, *properties=0*, *read_perm=0*, *write_perm=0*)
Output stream from the Service server.

bind(*service*)
Binds the characteristic to the given Service.

class StreamIn(*, *uuid=None*, *timeout=1.0*, *buffer_size=64*, *properties=0*, *write_perm=0*)
Input stream into the Service server.

bind(*service*)
Binds the characteristic to the given Service.

6.3.3.3 string

This module provides string characteristics.

class StringCharacteristic(*, *uuid=None*, *properties=0*, *read_perm=0*, *write_perm=0*, *initial_value=None*)
UTF-8 Encoded string characteristic.

class FixedStringCharacteristic(*, *uuid=None*, *read_perm=0*)
Fixed strings are set once when bound and unchanged after.

6.3.4 services

This module provides the top level Service definition.

class Service(*, *service=None*, *secondary=False*, ***initial_values*)
Top level Service class that handles the hard work of binding to a local or remote service.

Providers of a local service should instantiate their Service with service=None, the default. The local Service's characteristics will be lazily made available to clients as they are used locally. In other words, a characteristic won't be available to remote clients until it has been read or written locally.

To use a remote Service, get the item with the key of the Service type on the *BLEConnection*. For example, `connection[UartService]` will return the UartService instance for the connection's peer.

remote
True if the service is provided by a peer and accessed remotely.

6.3.4.1 standard

This module provides Service classes for BLE defined standard services.

class AppearanceCharacteristic(***kwargs*)
What type of device it is

class GenericAccess(*, *service=None*, *secondary=False*, ***initial_values*)
Required service that provides basic device information

class GenericAttribute(*, *service=None*, *secondary=False*, ***initial_values*)
Required service that provides notifications when Services change

class BatteryService(*, *service=None*, *secondary=False*, ***initial_values*)
Provides battery level information

class CurrentTimeService(*, *service=None*, *secondary=False*, ***initial_values*)
Provides the current time.

current_time
(year, month, day, hour, minute, second, weekday, subsecond, adjust_reason)

Type A tuple describing the current time

local_time_info
(timezone, dst_offset)

Type A tuple of location information

struct_time
The current time as a `time.struct_time`. Day of year and whether DST is in effect are always -1.

device_info

class DeviceInfoService(*, *manufacturer=None*, *software_revision=None*, *model_number=None*, *serial_number=None*, *firmware_revision=None*, *hardware_revision=None*, *service=None*)
Device information

hid

BLE Human Interface Device (HID)

- Author(s): Dan Halbert for Adafruit Industries

DEFAULT_HID_DESCRIPTOR = b'\x05\x01\t\x06\xa1\x01\x85\x01\x05\x07\x19\xe0)\xe7\x15\x00%\x0
provides mouse, keyboard, and consumer control devices.

Type Default HID descriptor

class ReportIn(*service*, *report_id*, *usage_page*, *usage*, *, *max_length*)
A single HID report that transmits HID data into a client.

send_report(*report*)
Send a report to the peers

class ReportOut(*service*, *report_id*, *usage_page*, *usage*, *, *max_length*)
A single HID report that receives HID data from a client.

class HIDService(*hid_descriptor=b'x05x01tx06xa1x01x85x01x05x07x19xe0)xe7x15x00%x01ux01x95x08x81x02x81x01x19x00*, *service=None*)
Provide devices for HID over BLE.

Parameters **hid_descriptor** (*str*) – USB HID descriptor that describes the structure of the reports. Known as the report map in BLE HID.

Example:

```
from adafruit_ble.hid_server import HIDServer

hid = HIDServer()
```

protocol_mode
boot (0) or report (1)

Type Protocol mode

hid_information
Hid information including version, country code and flags.

report_map
This is the USB HID descriptor (not to be confused with a BLE Descriptor). It describes which report characteristic are what.

suspended
Controls whether the device should be suspended (0) or not (1).

6.3.4.2 circuitpython

This module provides Services defined by CircuitPython. **Out of date.**

class CircuitPythonUUID(*uuid16*)
UUIDs with the CircuitPython base UUID.

class CircuitPythonService(*, *service=None*, *secondary=False*, ***initial_values*)
Core CircuitPython service that allows for file modification and REPL access. Unimplemented.

6.3.4.3 midi

This module provides Services defined by the MIDI group.

class MidiIOCharacteristic(***kwargs*)
Workhorse MIDI Characteristic that carries midi messages both directions. Unimplemented.

class MidiService(*, *service=None*, *secondary=False*, ***initial_values*)
BLE Service that transports MIDI messages. Unimplemented.

write()
Placeholder for transmitting midi bytes to the other device.

read()
Placeholder for receiving midi bytes from the other device.

6.3.5 uuid

This module provides core Unique ID (UUID) classes.

class UUID
Top level UUID

pack_into(*buffer*, *offset=0*)
Packs the UUID into the buffer at the given offset.

class StandardUUID(*uuid16*)
Standard 16-bit UUID defined by the Bluetooth SIG.

class VendorUUID(*uuid128*)
Vendor defined, 128-bit UUID.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index

D

E

F

G

H

I

L

M

N

P

R

S

T

U

V

W