Python interface to Android Bluetooth Low Energy API.
Generated documentation: http://able.readthedocs.org
Quick start development environment

*able is included in PythonHere app, together with the Jupyter Notebook it could be used as a development environment. Usage example: https://herethere.me/ble.html*
The following instructions are for building app with buildozer tool. 

*able_recipe* recipe should be added to buildozer.spec requirements:

```
requirements = python3,kivy,android,able_recipe
```

Bluetooth permissions should be requested in buildozer.spec:

```
android.permissions = BLUETOOTH, BLUETOOTH_ADMIN, ACCESS_FINE_LOCATION
```

App configuration example: buildozer.spec
To build app with a local (modified) version of *able*,
path to *able* recipes directory should be set in buildozer.spec:

```
p4a.local_recipes = /path/to/cloned/repo/recipes
```
Thanks,

andfmart
andreamerello
datmaniac95
dgatf
dwmoflatt
Enkumicahe1
FalkorDev
jacklinquan
juasiepo
PapoKar1o
RoberWare
5.1 Classes

class able.BluetoothDispatcher (queue_timeout=0.5, enable_ble_code=43806)
Bluetooth Low Energy interface

Parameters

• queue_timeout – BLE operations queue timeout
• enable_ble_code – request code to identify activity that allows user to turn on Bluetooth

bonded_devices

Type List[BluetoothDevice]

close_gatt ()
Close current GATT client

connect_by_device_address (address: str)
Connect to GATT Server of the device with a given Bluetooth hardware address, without scanning. Start a system activity that allows the user to turn on Bluetooth if Bluetooth is not enabled.

Parameters address – Bluetooth hardware address string in “XX:XX:XX:XX:XX:XX” format

Raises ValueError: if address is not a valid Bluetooth address

connect_gatt (device)
Connect to GATT Server hosted by device

discover_services ()
Discovers services offered by a remote device. The status of the discovery reported with services event.

Returns true, if the remote services discovery has been started
enable_notifications(characteristic, enable=True, indication=False)

Enable/disable notifications or indications for a given characteristic

Parameters

- characteristic – BluetoothGattCharacteristic Java object
- enable – enable notifications if True, else disable notifications
- indication – handle indications instead of notifications

Returns True, if the operation was initiated successfully

gatt

GATT profile of the connected device

Type BluetoothGatt Java object

on_characteristic_changed(characteristic)

characteristic_changed event handler

Parameters characteristic – BluetoothGattCharacteristic Java object

on_characteristic_read(characteristic, status)

characteristic_read event handler

Parameters

- characteristic – BluetoothGattCharacteristic Java object
- status – status of the operation, GATT_SUCCESS if the operation succeeds

on_characteristic_write(characteristic, status)

characteristic_write event handler

Parameters

- characteristic – BluetoothGattCharacteristic Java object
- status – status of the operation, GATT_SUCCESS if the operation succeeds

on_connection_state_change(status, state)

connection_state_change event handler

Parameters

- status – status of the operation, GATT_SUCCESS if the operation succeeds
- state – STATE_CONNECTED or STATE_DISCONNECTED

on_descriptor_read(descriptor, status)

descriptor_read event handler

Parameters

- descriptor – BluetoothGattDescriptor Java object
- status – status of the operation, GATT_SUCCESS if the operation succeeds

on_descriptor_write(descriptor, status)

descriptor_write event handler

Parameters

- descriptor – BluetoothGattDescriptor Java object
- status – status of the operation, GATT_SUCCESS if the operation succeeds
on_device \((device, rssi, advertisement)\)
device event handler. Event is dispatched when device is found during a scan.

Parameters
- \(device\) – BluetoothDevice Java object
- \(rssi\) – the RSSI value for the remote device
- \(advertisement\) – Advertisement data record

on_error \((msg)\)
Error handler

Parameters
- \(msg\) – error message

on_gatt_release ()
gatt_release event handler. Event is dispatched at every read/write completed operation

on_mtu_changed \((mtu, status)\)
onMtuChanged event handler Event is dispatched when MTU for a remote device has changed, reporting a new MTU size.

Parameters
- \(mtu\) – integer containing the new MTU size
- \(status\) – status of the operation, \(GATT_SUCCESS\) if the MTU has been changed successfully

on_rssi_updated \((rssi, status)\)
onReadRemoteRssi event handler. Event is dispatched at every RSSI update completed operation, reporting a RSSI value for a remote device connection.

Parameters
- \(rssi\) – integer containing RSSI value in dBm
- \(status\) – status of the operation, \(GATT_SUCCESS\) if the operation succeeds

on_scan_completed ()
scan_completed event handler

on_scan_started \((success)\)
scan_started event handler

Parameters
- \(success\) – true, if scan was started successfully

on_services \((services, status)\)
services event handler

Parameters
- \(services\) – Services dict filled with discovered characteristics (BluetoothGattCharacteristic Java objects)
- \(status\) – status of the operation, \(GATT_SUCCESS\) if the operation succeeds

read_characteristic \((characteristic)\)
Read a given characteristic from the associated remote device

Parameters
- \(characteristic\) – BluetoothGattCharacteristic Java object

request_mtu \((mtu: int)\)
Request to change the ATT Maximum Transmission Unit value

Parameters
- \(value\) – new MTU size
set_queue_timeout (timeout)
   Change the BLE operations queue timeout

start_scan()
   Start a scan for devices. Ask for runtime permission to access location. Start a system activity that allows
   the user to turn on Bluetooth, if Bluetooth is not enabled. The status of the scan start are reported with
   scan_started event.

stop_scan()
   Stop the ongoing scan for devices.

update_rssi()
   Triggers an update for the RSSI from the associated remote device

write_characteristic (characteristic, value, write_type: Optional[able.WriteType] = None)
   Write a given characteristic value to the associated remote device

   Parameters
   • characteristic – BluetoothGattCharacteristic Java object
   • value – value to write
   • write_type – specific write type to set for the characteristic

write_descriptor (descriptor, value)
   Set and write the value of a given descriptor to the associated remote device

   Parameters
   • descriptor – BluetoothGattDescriptor Java object
   • value – value to write

class able.Advertisement (data)
   Advertisement data record parser

   >>> ad = Advertisement([2, 1, 0x6, 6, 255, 82, 83, 95, 82, 48])
   >>> for data in ad:
   ...     data
   AD(ad_type=1, data=bytearray(b'\x06'))
   AD(ad_type=255, data=bytearray(b'RS_R0'))
   >>> list(ad)[0].ad_type == Advertisement.ad_types.flags
   True

class ad_types
   Assigned numbers for some of advertisement data types.

   flags : “Flags” (0x01)
   complete_local_name : “Complete Local Name” (0x09)
   service_data : “Service Data” (0x16)
   manufacturer_specific_data : “Manufacturer Specific Data” (0xff)

class able.Services
   Services dict

   >>> services = Services({'service0': {'c1-aa': 0, 'aa-c2-aa': 1},
   ... 'service1': {'bb-c3-bb': 2}})
   >>> services.search('c3')
   2
   >>> services.search('c4')
search (pattern, flags=<RegexFlag.IGNORECASE: 2>)
Search for characteristic by pattern

Parameters

- pattern – regexp pattern
- flags – regexp flags, re.IGNORECASE by default

5.2 Constants

ABLE.GATT_SUCCESS = 0
GATT operation completed successfully

ABLE.STATE_CONNECTED = 2
The profile is in connected state

ABLE.STATE_DISCONNECTED = 0
The profile is in disconnected state

class able.WriteType
GATT characteristic write types constants.

DEFAULT = 2
Write characteristic, requesting acknowledgement by the remote device

NO_RESPONSE = 1
Write characteristic without requiring a response by the remote device

SIGNED = 4
Write characteristic including authentication signature
6.1 Alert

```python
### Turn the alert on Mi Band device

from kivy.app import App
from kivy.uix.button import Button

from able import BluetoothDispatcher, GATT_SUCCESS
from error_message import install_exception_handler

class BLE(BluetoothDispatcher):
    device = alert_characteristic = None

    def start_alert(self, *args, **kwargs):
        if self.alert_characteristic:
            # alert service is already discovered
            self.alert(self.alert_characteristic)
        elif self.device:
            # device is already founded during the scan
            self.connect_gatt(self.device)  # reconnect
        else:
            self.stop_scan()  # stop previous scan
            self.start_scan()  # start a scan for devices

    def on_device(self, device, rssi, advertisement):
        # some device is found during the scan
        name = device.getName()
        if name and name.startswith('MI'):  # is a Mi Band device
            self.device = device
            self.stop_scan()

    def on_scan_completed(self):
        if self.device:
            self.connect_gatt(self.device)  # connect to device

(continues on next page)```
def on_connection_state_change(self, status, state):
    if status == GATT_SUCCESS and state:  # connection established
        self.discover_services()  # discover what services a device offer
    else:  # disconnection or error
        self.alert_characteristic = None
        self.close_gatt()  # close current connection

def on_services(self, status, services):
    # 0x2a06 is a standard code for "Alert Level" characteristic
    self.alert_characteristic = services.search('2a06')
    self.alert(self.alert_characteristic)

def alert(self, characteristic):
    self.write_characteristic(characteristic, [2])  # 2 is for "High Alert"

class AlertApp(App):
    def build(self):
        self.ble = None
        return Button(text='Press to Alert Mi', on_press=self.start_alert)

def start_alert(self, *args, **kwargs):
    if not self.ble:
        self.ble = BLE()
        self.ble.start_alert()

if __name__ == '__main__':
    install_exception_handler()
    AlertApp().run()

6.2 Change MTU

"""Request MTU change, and write 100 bytes to a characteristic."""
from kivy.app import App
from kivy.clock import Clock
from kivy.logger import Logger
from kivy.uix.widget import Widget
from able import BluetoothDispatcher, GATT_SUCCESS

class BLESender(BluetoothDispatcher):
    def __init__(self):
        super().__init__()
        self.characteristic_to_write = None
        Clock.schedule_once(self.connect, 0)

    def connect(self, _):
        (continues on next page)
self.connect_by_device_address("FF:FF:FF:FF:FF:FF")

def on_connection_state_change(self, status, state):
    if status == GATT_SUCCESS and state:
        self.discover_services()

def on_services(self, status, services):
    if status == GATT_SUCCESS:
        self.characteristic_to_write = services.search("0d03")
        # Need to request 100 + 3 extra bytes for ATT packet header
        self.request_mtu(103)

def on_mtu_changed(self, mtu, status):
    if status == GATT_SUCCESS and mtu == 103:
        Logger.info("MTU changed: now it is possible to send 100 bytes at once")
        self.write_characteristic(self.characteristic_to_write, range(100))
    else:
        Logger.error("MTU not changed: mtu=\d, status=\d", mtu, status)

def on_characteristic_write(self, characteristic, status):
    if status == GATT_SUCCESS:
        Logger.info("Characteristic write succeed")
    else:
        Logger.error("Write status: \d", status)

class MTUApp(App):
    def build(self):
        BLESender()
        return Widget()

if __name__ == '__main__':
    MTUApp().run()
Python Module Index

a

able, 9
Index

A
able (module), 9
Advertisement (class in able), 14
Advertisement.ad_types (class in able), 14

B
BluetoothDispatcher (class in able), 11
bonded_devices (able.BluetoothDispatcher attribute), 11

C
close_gatt() (able.BluetoothDispatcher method), 11
connect_by_device_address() (able.BluetoothDispatcher method), 11
connect_gatt() (able.BluetoothDispatcher method), 11

discover_services() (able.BluetoothDispatcher method), 11

e
enable_notifications() (able.BluetoothDispatcher method), 11

G
gatt (able.BluetoothDispatcher attribute), 12
GATT_SUCCESS (in module able), 15

N
NO_RESPONSE (able.WriteType attribute), 15

O
on_characteristic_changed() (able.BluetoothDispatcher method), 12
on_characteristic_read() (able.BluetoothDispatcher method), 12
on_characteristic_write() (able.BluetoothDispatcher method), 12
on_connection_state_change() (able.BluetoothDispatcher method), 12
on_descriptor_read() (able.BluetoothDispatcher method), 12
on_descriptor_write() (able.BluetoothDispatcher method), 12
on_device() (able.BluetoothDispatcher method), 12
on_error() (able.BluetoothDispatcher method), 13
on_gatt_release() (able.BluetoothDispatcher method), 13
on_mtu_changed() (able.BluetoothDispatcher method), 13
on_rssi_updated() (able.BluetoothDispatcher method), 13
on_scan_completed() (able.BluetoothDispatcher method), 13
on_scan_started() (able.BluetoothDispatcher method), 13
on_services() (able.BluetoothDispatcher method), 13

R
read_characteristic() (able.BluetoothDispatcher method), 13
request_mtu() (able.BluetoothDispatcher method), 13

S
search() (able.Services method), 14
Services (class in able), 14
set_queue_timeout() (able.BluetoothDispatcher method), 13
SIGNED (able.WriteType attribute), 15
start_scan() (able.BluetoothDispatcher method), 14
STATE_CONNECTED (in module able), 15
STATE_DISCONNECTED (in module able), 15
stop_scan() (able.BluetoothDispatcher method), 14
U
update_rssi() (*able.BluetoothDispatcher method), 14

W
write_characteristic() (*able.BluetoothDispatcher method), 14
write_descriptor() (*able.BluetoothDispatcher method), 14
WriteType (*class in able), 15