pyHomeKit Documentation

Release 0.0.1

Henri Dwyer

Oct 05, 2017

Contents

CHAPTER 1

PyHomeKit - HomeKit for Python

Author Henri Dwyer

PyHomeKit is a set of python libraries that let you control HomeKit compatible accessories, both BLE and HTTP.

Important: PyHomeKit is currently in pre-alpha. Many features are not yet implemented or broken.

Getting Started

Usage

Connect to a HAP characteristics and view its signature:

```
import pyhomekit

device_mac = "aa:aa:aa:aa:aa"

characteristic_uuid = "00000000-0000-0000-0000-000000000000"

accessory = pyhomekit.ble.HapAccessory(device_mac, 'random')
accessory.connect()
hap_characteristic = pyhomekit.ble.HapCharacteristic(accessory=accessory,␣
↪uuid=characteristic_uuid)

print(hap_characteristic.signature)
```

View the debug logs in stdout:

```
import logging

logging.basicConfig()
logging.getLogger('pyhomekit').setLevel(logging.DEBUG)
```

Installation

`pyHomeKit` is on Pypi, so you can `pip` install it:

```
pip install pyhomekit
```

If you want to install from source, clone the repository:

```
git clone git://github.com/henridwyer/pyhomekit.git
cd pyhomekit
pip install -r requirements.txt
pip install -e .
```

Then you can build the documentation:

```
make doc
```

And run the tests:

```
make tests
```

Links

For more information about HomeKit, see the Apple Developper HomeKit page.

Bluetooth Low Energy device compatibility is provided by bluepy.

Encryption is provided by the libsodium library.

CHAPTER 2

pyhomekit

pyhomekit package

Module contents

pyhomekit.ble module

Contains all of the HAP-BLE classes.

class pyhomekit.ble.**HapAccessory**(*address: str*, *address_type: str = 'static'*) → None
Bases: object

HAP Accesory.

Parameters

- **address** – MAC address of the accessory
- **address_type** – Type of the address: static or random

charateristic(*uuid: str*) → sphinx.ext.autodoc.Characteristic
Return the GATT characteristic for the given UUID.

connect() → None
Connect to BLE peripheral.

discover_hap_characteristics() → typing.List[pyhomekit.ble.HapCharacteristic]
Discovers all of the HAP Characteristics and performs a signature read on each one.

get_characteristic(*name: str*, *uuid: str*) → pyhomekit.ble.HapCharacteristic

pair() → None

pair_verify() → None

save_key() → None

class pyhomekit.ble.**HapAccessoryLock**(*address: str*, *address_type: str = 'static'*) → None
Bases: *pyhomekit.ble.HapAccessory*

administrator_only_access() → None

audio_feedback() → bytes

current_door_state() → int

lock_control_point() → typing.Any

lock_current_state() → int

lock_last_known_action() → int

lock_management_auto_security_timeout() → None

lock_target_state() → None

logs() → str

motion_detected() → bool

version() → str

class pyhomekit.ble.**HapBlePdu**(*header: pyhomekit.ble.HapBlePduHeader, TLVs: typing.Sequence[typing.Tuple[int, bytes]]*) → None

Bases: object

HAP BLE PDU

fragmented

max_len = 512

pdu_fragments() → typing.Iterator[bytes]

raw_data

class pyhomekit.ble.**HapBlePduHeader**(*response: bool*, *continuation: bool*) → None

Bases: object

Interface for HAP-BLE Headers.

This class is not meant to be instantiated. Use the children HapBlePduRequestHeader and HapBlePduResponseHeader.

Parameters

- **continuation** – indicates the fragmentation status of the HAP-BLE PDU. False indicates a first fragment or no fragmentation.
- **response** – indicates whether the PDU is a response (versus a request)

control_field
Get Control Field as int.

control_field_bits
Get Control Field as string of bits.

data

class pyhomekit.ble.**HapBlePduRequestHeader**(*cid_sid: bytes*, *op_code: int*, *response: bool = False*, *continuation: bool = False*, *transaction_id: int = None*) → None

Bases: *pyhomekit.ble.HapBlePduHeader*

HAP-BLE PDU Request Header.

Parameters

- **continuation** – indicates the fragmentation status of the HAP-BLE PDU. False indicates a first fragment or no fragmentation.
- **response** – indicates whether the PDU is a response (versus a request)
- **transation_id** – Transaction Identifier
- **op_code** – HAP Opcode field, which indicates the opcode for the HAP Request PDU.

- **cid_sid** – Characteristic / Service Instance Identifier is the instance id of the characteristic / service for a particular request.

data
Byte representation of the PDU Header.

Depends on whether it is a continuation header or not.

transaction_id
Get the transaction identifier, or generate a new one if none exists.

The transation ID is an 8 bit number identifying the transaction number of this PDU. The TID is randomly generated by the originator of the request and is used to match a request/response pair.

class pyhomekit.ble.**HapBlePduResponseHeader**(*status_code: int*, *transaction_id: int*, *continuation: bool = False*, *response: bool = True*) → None

Bases: *pyhomekit.ble.HapBlePduHeader*

HAP-BLE PDU Response Header.

Parameters

- **continuation** – indicates the fragmentation status of the HAP-BLE PDU. False indicates a first fragment or no fragmentation.
- **response** – indicates whether the PDU is a response (versus a request)
- **transaction_id** – Transaction Identifier
- **status_code** – HAP Status code for the request.

data
Byte representation of the PDU Header.

classmethod from_data(*data: bytes*) → pyhomekit.ble.HapBlePduResponseHeader
Creates a header from response bytes

class pyhomekit.ble.**HapCharacteristic**(*accessory: pyhomekit.ble.HapAccessory*, *uuid: str*, *retry: bool = False*, *retry_max_attempts: int = 1*, *retry_wait_time: int = 2*) → None

Bases: object

Represents data or an associated behavior of a service.

The characteristic is defined by a universally unique type, and has additional properties that determine how the value of the characteristic can be accessed.

Parameters

- **accessory** – The accessory this characteristic belongs to.
- **uuid** – The UUID of the underlying GATT characteristic
- **retry** – Attempt to reconnect when error.
- **retry_max_attempts** – How many times to attempt reconnection.
- **retry_wait_time** – How long to wait in s between reconnection attempts.

cid
Get the Characteristic ID, reading it from the device if required.

read(*request_header: pyhomekit.ble.HapBlePduRequestHeader*) → typing.Dict[str, typing.Any]
Perform a HAP Characteristic read.

Fragmented read if required.

signature
Returns the signature, and adds the attributes.

write(*request_header: pyhomekit.ble.HapBlePduRequestHeader, TLVs: typing.List[typing.Tuple[int, bytes]]*) → typing.Dict[str, typing.Any]
Perform a HAP Characteristic write.

Fragmented read/write if required.

write_ktlvs(*request_header: pyhomekit.ble.HapBlePduRequestHeader, kTLVs: typing.Sequence[typing.Tuple[int, bytes]]*) → typing.Dict[str, typing.Any]
Perform a HAP Characteristic write for a pairing.

Fragmented read/write if required.

pyhomekit.ble.**fragment_tlvs**(*header: pyhomekit.ble.HapBlePduRequestHeader, TLVs: typing.List[typing.Tuple[int, bytes]]*) → typing.Iterator[bytes]
Returns the fragmented TLVs to write.

pyhomekit.ble.**reconnect_callback_factory**(*accessory: pyhomekit.ble.HapAccessory*) → typing.Callable[[typing.Any, int], NoneType]
Factory for creating tenacity before callbacks to reconnect to a peripheral.

pyhomekit.ble.**reconnect_tenacity_retry**(*reconnect_callback: typing.Callable[[typing.Any, int], typing.Any], max_attempts: int = 2, wait_time: int = 2*) → tenacity.Retrying
Build tenacity retry object

pyhomekit.constants module

HAP Constants

class pyhomekit.constants.**HapBleOpCodes**
Bases: object

HAP Opcode Descriptions.

Characteristic_Execute_Write = 5

Characteristic_Read = 3

Characteristic_Signature_Read = 1

Characteristic_Timed_Write = 4

Characteristic_Write = 2

Service_Signature_Read = 6

class pyhomekit.constants.**HapBleStatusCodes**
Bases: object

HAP Status code definitions and descriptions.

Insufficient_Authentication = 5

Insufficient_Authorization = 3

Invalid_Instance_ID = 4

Invalid_Request = 6

Max_Procedures = 2

Success = 0

Unsupported_PDU = 1

class pyhomekit.constants.**HapParamTypes**
Bases: object

Additional_Authorization_Data = 2

Characteristic_Instance_ID = 5

Characteristic_Type = 4

GATT_Presentation_Format_Descriptor = 12

GATT_User_Description_Descriptor = 11

GATT_Valid_Range = 13

HAP_Characteristic_Properties_Descriptor = 10

HAP_Linked_Services = 16

HAP_Service_Properties = 15

HAP_Step_Value_Descriptor = 14

HAP_Valid_Values_Descriptor = 17

HAP_Valid_Values_Range_Descriptor = 18

Origin_local_vs_remote = 3

Return_Response = 9

Service_Instance_ID = 7

Service_Type = 6

TTL = 8

Value = 1

class pyhomekit.constants.**PairingKTLVErrorCodes**
Bases: object

kTLVError_Authenticatio = 2

kTLVError_Backof = 3

kTLVError_Bus = 7

kTLVError_MaxPeer = 4

kTLVError_MaxTrie = 5

kTLVError_Unavailabl = 6

kTLVError_Unknow = 1

class pyhomekit.constants.**PairingKTLVMethodValues**
Bases: object

Pairing service kTLV method values

Add_Pairing = 3

List_Pairings = 5

Pair_Setup = 1

Pair_Verify = 2

Remove_Pairing = 4

Reserved = 0

class pyhomekit.constants.**PairingKTlvValues**
Bases: object

Pairng service TLV Values.

kTLVType_Certificate = 9

kTLVType_EncryptedData = 5

kTLVType_Error = 7

kTLVType_FragmentData = 12

kTLVType_FragmentLast = 13

kTLVType_Identifier = 1

kTLVType_Method = 0

kTLVType_Permissions = 11

kTLVType_Proof = 4

kTLVType_PublicKey = 3

kTLVType_RetryDelay = 8

kTLVType_Salt = 2

kTLVType_Separator = 255

kTLVType_Signature = 10

kTLVType_State = 6

pyhomekit.constants.**identity**(*x: typing.Any*) → typing.Any
Identity

pyhomekit.constants.**parse_format**(*b: bytes*) → typing.Tuple[int, int]
Parse the bluetooth characteristic presentation format to format and unit code

pyhomekit.constants.**to_bool**(*b: bytes*) → bool
Convert to bytes to bool (little endian).

pyhomekit.constants.**to_float**(*b: bytes*) → int
Convert to bytes to float (little endian).

pyhomekit.constants.**to_int32**(*b: bytes*) → int
Convert to bytes to 32 bit signed int (little endian).

pyhomekit.constants.**to_uint16**(*b: bytes*) → int
Convert to bytes to 16 bit unsigned short (little endian).

pyhomekit.constants.**to_uint32**(*b: bytes*) → int
Convert to bytes to 32 bit unsigned int (little endian).

pyhomekit.constants.**to_uint64**(*b: bytes*) → int
Convert to bytes to 64 bit unsigned int (little endian).

pyhomekit.constants.**to_uint8**(*b: bytes*) → int
Convert to bytes to 8 bit unsigned short (little endian).

pyhomekit.constants.**to_utf8**(*b: bytes*) → str
Convert bytes to str utf-8 encoded.

pyhomekit.constants.**to_uuid**(*b: bytes*) → str
Convert bytes to string representation of uuid.

The bytes are reversed first.

pyhomekit.utils module

Utility functions for BLE

exception pyhomekit.utils.**HapBleError**(*status_code: int = None*, *name: str = None*, *message: str = None*, **args: str*) → None
Bases: Exception

HAP Error.

pyhomekit.utils.**iterate_tvl**(*response: bytes*) → typing.Iterator[typing.Tuple[int, int, bytes]]
Iterate through response bytes, 1 tlv at a time.

pyhomekit.utils.**parse_ktlvs**(*data: bytes*) → typing.Dict[str, typing.Any]
Parse ktlvs.

pyhomekit.utils.**prepare_tlv**(*param_type: int*, *value: bytes*) → typing.Iterator[bytes]
Formats the TLV into the expected format of the PDU.

Parameters

- **param_type** – The name or code for the HAP Parameter type
- **value** – The value in bytes of the parameter.

CHAPTER 3

Changelog

0.0.1.4

- Read The Docs integration
- Travis integration
- Code quality checking
- Tests

0.0.1

- Initial version

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

I

K

L