RPi-SmartGadget Documentation

Release 0.1.0.dev0

Measurement Standards Laboratory

Aug 06, 2022

CONTENTS

Communicate with a Sensirion SHTxx Smart Gadget.

A Raspberry Pi is used for the Bluetooth Low Energy connectivity and the MSL-Network package allows any computer on the same network as the Raspberry Pi to request information from the Smart Gadgets that are within Bluetooth range of the Raspberry Pi.

CHAPTER

ONE

CONTENTS

1.1 Install RPi-SmartGadget

1.1.1 Raspberry Pi OS

This section describe how to set up a Raspberry Pi.

Note: Instructions for using ssh to remotely access the terminal of the Raspberry Pi can be found here.

Tip: The following command is optional *but recommended*. It will update the installed packages on the Raspberry Pi.

```
sudo apt update && sudo apt upgrade
```

Make sure that you have git installed and then clone the repository

```
sudo apt install git
git clone https://github.com/MSLNZ/rpi-smartgadget.git
```

The following will install the **RPi-SmartGadget** package in a virtual environment in the `/home/pi/shtenv` directory on the Raspberry Pi *(the* `shtenv` *directory will be automatically created)*

```
bash rpi-smartgadget/rpi-setup.sh
```

1.1.2 Windows, Linux or macOS

To install **RPi-SmartGadget** on a computer that is not a Raspberry Pi run

```
pip install https://github.com/MSLNZ/rpi-smartgadget/archive/main.tar.gz
```

Alternatively, using the MSL Package Manager run

```
msl install rpi-smartgadget
```

1.1.3 Dependencies

Tested with a Raspberry Pi 3 Model B+ and a Raspberry Pi 4 Model B running either Raspbian Stretch or Buster.

- Python 3.5+
- MSL-Network
- bluepy – only installed on the Raspberry Pi

Note: Although **RPi-SmartGadget** has been tested with a Raspberry Pi to establish a Bluetooth connection with a Smart Gadget, any Linux-based operating system that bluepy supports could be used.

1.2 Basic Usage

Place a Raspberry Pi in a location where the Smart Gadgets are within Bluetooth range.

On another computer that is on the same network as the Raspberry Pi run the following

---

Note: You will have to change the value of *host* below for your Raspberry Pi. The reason for including `assert_hostname=False` is because we show that we are specifying an IP address for the value of *host* instead of its hostname. The hostname of the Raspberry Pi is *(most likely)* `'raspberrypi'` and so `'xxx.xxx.xxx.xxx'` won't equal `'raspberrypi'` when the security of the connection is checked behind the scenes. If you specify the hostname of the Raspberry Pi then you can do hostname verification and not include the `assert_hostname=False` keyword argument. In general, use `assert_hostname=False` at your own risk if there is a possibility of a man-in-the-middle hijack between your computer and the remote computer.

```
>>> from smartgadget import connect
>>> rpi = connect(host='xxx.xxx.xxx.xxx', assert_hostname=False)
```

To find out what can be requested from a Smart Gadget, run the following

```
>>> print(rpi.manager(as_string=True, indent=2))
Manager[raspberrypi:1875]
  attributes:
    identity() -> dict
    link(service: str) -> bool
  language: Python 3.7.3
  os: Linux 4.19.97-v7+ armv7l
Clients [1]:
  LinkedClient[192.168.1.69:63117]
    language: Python 3.7.7
    os: Windows 10 AMD64
Services [1]:
  Smart Humigadget[raspberrypi:36834]
    attributes:
      battery(mac_address) -> int
```

(continues on next page)

(continued from previous page)

```
      connect_gadget(mac_address, strict=True) -> bool
      connect_gadgets(mac_addresses, strict=True) -> Tuple[list, list]
      connected_gadgets() -> List[str]
      dewpoint(mac_address, temperature=None, humidity=None) -> float
      disable_humidity_notifications(mac_address)
      disable_temperature_notifications(mac_address)
      disconnect_gadget(mac_address)
      disconnect_gadgets()
      enable_humidity_notifications(mac_address)
      enable_temperature_notifications(mac_address)
      fetch_logged_data(mac_address, *, enable_temperature=True, enable_
↪humidity=True, sync=None, oldest=None, newest=None, as_datetime=False, num_
↪iterations=1) -> Tuple[list, list]
      humidity(mac_address) -> float
      humidity_notifications_enabled(mac_address) -> bool
      info(mac_address) -> dict
      logger_interval(mac_address) -> int
      max_attempts() -> int
      newest_timestamp(mac_address) -> int
      oldest_timestamp(mac_address) -> int
      restart_bluetooth()
      rpi_date() -> str
      rssi(mac_address) -> int
      scan(timeout=10, passive=False) -> List[str]
      set_logger_interval(mac_address, milliseconds)
      set_max_attempts(max_attempts)
      set_newest_timestamp(mac_address, timestamp)
      set_oldest_timestamp(mac_address, timestamp)
      set_rpi_date(date)
      set_sync_time(mac_address, timestamp=None)
      shutdown_service()
      temperature(mac_address) -> float
      temperature_humidity(mac_address) -> Tuple[float, float]
      temperature_humidity_dewpoint(mac_address) -> Tuple[float, float, float]
      temperature_notifications_enabled(mac_address) -> bool
   language: Python 3.7.3
   max_clients: -1
   os: Linux 4.19.97-v7+ armv7l
```

The information about the `Manager` and which `Client`s and `Service`s are connected to it will be shown. The `Smart Humigadget Service` indicates that it has the following methods that can be called: *battery*, *connect_gadget*, etc…

Next, we scan for all available Smart Gadgets, request the temperature, humidity and dew point and then disconnect from the Raspberry Pi

```
>>> mac_addresses = rpi.scan()
>>> for address in mac_addresses:
...     print(address, rpi.temperature_humidity_dewpoint(address))
c7:99:a8:77:e9:2a [20.329999923706055, 49.81999969482422, 9.521468351961703]
```

(continues on next page)

(continued from previous page)

```
cc:ea:2e:0c:11:f6 [19.56999969482422, 48.77000045776367, 8.507598739882166]
ed:8d:dd:6a:58:25 [20.229999542236328, 46.060001373291016, 8.267915590472189]
ea:12:51:be:f9:6e [20.40999984741211, 47.060001373291016, 8.749198797952799]
ef:ce:43:b4:83:f8 [21.399999618530273, 39.84000015258789, 7.196289989617892]
>>> rpi.disconnect()
```

1.3 Examples

The following examples are available to download from here

1.3.1 fetch_logged_data_remotely.py

```
"""
This example assumes that you are connecting to a Raspberry Pi from
another computer on the network to fetch the logged data.
"""
from smartgadget import connect, milliseconds_to_datetime

# The MAC address of the Smart Gadget you want to download the data from
mac_address = 'ef:ce:43:b4:83:f8'

# Connect to the Raspberry Pi (update the IP address of the Raspberry Pi)
rpi = connect(host='192.168.1.100', assert_hostname=False)

# We will be picky and only allow 1 attempt to perform a Smart Gadget request.
# Increasing this value will decrease the occurrence of getting a
# BTLEDisconnectError or a BrokenPipeError when sending requests.
rpi.set_max_attempts(1)

# Connect to the Smart Gadget. This is optional, you could call
# rpi.fetch_logged_data() without first connecting to the Smart Gadget.
rpi.connect_gadget(mac_address)

# Fetch all temperature logger data.
# The humidity data is also returned but it will be an emtpy list.
#
# This step can take a very long time (minutes) if there is a lot of data
# to fetch or if the Bluetooth connection is slow/keeps dropping out.
# There is no option to display the current status of the request
# -- see fetch_logged_data_rpi.py which will display status updates.
print('Fetching data...')
temperatures, humidities = rpi.fetch_logged_data(mac_address, enable_
↪humidity=False)
print('Fetched {} temperature values'.format(len(temperatures)))

# Disconnect from the Raspberry Pi when finished communicating with it
rpi.disconnect()
```

(continues on next page)

(continued from previous page)

```
# Save the temperature results (the returned timestamps are in milliseconds)
with open('temperature.csv', mode='wt') as fp:
    fp.write('timestamp,temperature[C]\n')
    for timestamp, value in temperatures:
        fp.write('{},{}\n'.format(milliseconds_to_datetime(timestamp), value))
```

1.3.2 fetch_logged_data_rpi.py

```
#!/home/pi/shtenv/bin/python
"""
This example assumes that you are directly running the script on a Raspberry Pi.

You must execute this script as the root user in order to have access to
the Bluetooth drivers.

First, make this script executable

  $ chmod +x fetch_logged_data_rpi.py

Next, execute the script

  $ sudo ./fetch_logged_data_rpi.py

"""
import logging
from smartgadget import SHT3XService

# This allows you to see some status messages displayed to the terminal
logging.basicConfig(level=logging.DEBUG, format='%(asctime)s [%(levelname)-7s] %(message)s')

# The MAC address of the Smart Gadget you want to download the data from
mac_address = 'ef:ce:43:b4:83:f8'

# Create an instance of the Smart Gadget Service
s = SHT3XService()

# Fetch all available temperature and humidity logger data.
# Perform this 2 times to reduce missing data packets during the Bluetooth download.
# This step can take a very long time (minutes) if there is a lot of data
# to fetch or if the Bluetooth connection is slow/keeps dropping out.
temperatures, humidities = s.fetch_logged_data(mac_address, num_iterations=2, as_datetime=True)

# Disconnect from the Smart Gadget when finished communicating with it
```

(continues on next page)

(continued from previous page)

```
s.disconnect_gadgets()

# Save the results
with open('temperature.csv', mode='wt') as fp:
    fp.write('timestamp,temperature[C]\n')
    for row in temperatures:
        fp.write('{},{}\n'.format(*row))

with open('humidity.csv', mode='wt') as fp:
    fp.write('timestamp,humidity[%RH]\n')
    for row in humidities:
        fp.write('{},{}\n'.format(*row))
```

1.3.3 set_logger_interval.py

```
"""
This script will set the logger interval for all available Smart Gadgets.

This will delete all values from the memory of each Smart Gadget so that the
timestamps for all Smart Gadgets are in sync.
"""
from time import perf_counter
from smartgadget import connect

# Connect to the Raspberry Pi (update the IP address of the Raspberry Pi)
rpi = connect(host='192.168.1.100', assert_hostname=False)

# Get all available Smart Gadgets
mac_addresses = rpi.scan()

# We connect to all Smart Gadgets now so that the start time of each Smart
# Gadget logger is as close as possible to all other Smart Gadgets. Without
# first connecting to all Smart Gadgets the start time for each Smart Gadget
# would accumulate approximately a 7-second delay (the time it takes to
# connect to a Smart Gadget) compared to the start time of the
# previously-configured Smart Gadget.
rpi.connect_gadgets(mac_addresses)
print('Connected to: {}'.format(rpi.connected_gadgets()))

# Set the logger interval to be 10 seconds (10000 milliseconds)
t0 = perf_counter()
for address in mac_addresses:
    print('Setting the logger interval for {!r}'.format(address))
    rpi.set_logger_interval(address, 10000)
dt = perf_counter() - t0
print('All Smart Gadgets should be in sync to within {:.3f} seconds'.
↪format(dt))
```

(continues on next page)

(continued from previous page)

```
# Disconnect from the Raspberry Pi when finished communicating with it
rpi.disconnect()
```

1.3.4 custom_logging.py

```
"""
Sample script to log the battery, temperature, humidity and dew point from all
available Smart Gadgets to separate CSV files. The MAC address of a Smart␣
↪Gadget
is used as the name of the file.

Sometimes the number of Smart Gadgets discovered during a Bluetooth scan is
less than the number expected. This example script does not rescan until the
expected number are found. Alternatively, if you know the MAC addresses of the
Smart Gadgets that you want to log then you could specify the MAC addresses
in a list and not perform the Bluetooth scan.
"""
import os
import time
from datetime import datetime

from smartgadget import connect, kill_manager

# The IP address of the Raspberry Pi
host = '192.168.1.100'

# The password of the Raspberry Pi (this should be read from a file or from␣
↪the terminal)
rpi_password = '<PASSWORD>'

# The folder to save the data to (default value is the current working␣
↪directory)
save_dir = ''

# You could also specify the MAC addresses rather than performing the scan␣
↪below
mac_addresses = []

# The number of seconds to wait after fetching the data from all Smart Gadgets
sleep = 10

# Initialize these parameters (they don't need to be changed)
files = {}
rpi = None

while True:
    try:
        if rpi is None:
```

(continues on next page)

(continued from previous page)

```
            # Connect to the Raspberry Pi and scan for Smart Gadgets
            print('Connecting to the Raspberry Pi...')
            rpi = connect(host=host, rpi_password=rpi_password, assert_
↪hostname=False)
            print('Scanning for Smart Gadgets...')
            mac_addresses = rpi.scan()
            print('Found {} Smart Gadgets'.format(len(mac_addresses)))
            if not mac_addresses:
                break
            print('Connecting to {} Smart Gadgets...'.format(len(mac_
↪addresses)))
            rpi.connect_gadgets(mac_addresses)
            for address in mac_addresses:
                files[address] = os.path.join(save_dir, address.replace(':',
↪'-') + '.csv')
                if not os.path.isfile(files[address]):
                    print('Create logging file {!r}'.format(files[address]))
                    with open(files[address], mode='wt') as f:
                        f.write('Timestamp,Battery[%],Temperature[C],Humidity[
↪%RH],Dewpoint[C]\n')

        for address in mac_addresses:
            # Fetch the data and append to the appropriate file
            battery = rpi.battery(address)
            values = rpi.temperature_humidity_dewpoint(address)
            now = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
            print('{} [{}] {:3d} {:.3f} {:.3f} {:.3f}'.format(now, address,␣
↪battery, *values))
            with open(files[address], mode='at') as fp:
                fp.write('{},{},{},{},{}\n'.format(now, battery, *values))

        time.sleep(sleep)

    except KeyboardInterrupt:
        print('CTRL+C received...')
        break

    except Exception as e:
        rpi = None
        msg = str(e).splitlines()[-1]
        print(msg)
        if msg.endswith('address already in use'):
            print('Killing the Network Manager...')
            kill_manager(host=host, rpi_password=rpi_password)

# Must wait for the previous request to finish before sending the disconnect␣
↪request
try:
    rpi.wait()
```

(continues on next page)

(continued from previous page)

```
except:
    pass

# Disconnect from the Raspberry Pi
try:
    rpi.disconnect()
except:
    pass

print('Disconnected from the Raspberry Pi')
```

1.4 API Documentation

The root package is

smartgadget	Communicate with a Sensirion SHTxx Smart Gadget.

which has the following class for interfacing with a SHT3X Smart Gadget,

SHT3XService([interface])	The Service for a *SHT3X* Smart Gadget.

and the following class for interfacing with a SHTC1 Smart Gadget,

Attention: This class has not been tested since the hardware has not been available.

SHTC1Service([interface])	The Service for a *SHTC1* Smart Gadget.

1.4.1 Package Structure

smartgadget package

Communicate with a Sensirion SHTxx Smart Gadget.

smartgadget.**connect**(*, *host='raspberrypi'*, *rpi_username='pi'*, *rpi_password=None*, *timeout=10*, ***kwargs*)

Connect to the *SHT3XService* on the Raspberry Pi.

Parameters

- **host** (str, optional) – The hostname or IP address of the Raspberry Pi.
- **rpi_username** (str, optional) – The username for the Raspberry Pi.
- **rpi_password** (str, optional) – The password for *rpi_username*.

- **timeout** (float, optional) – The maximum number of seconds to wait for the connection.
- **kwargs** – Keyword arguments that are passed to run_services().

Returns
SmartGadgetClient – A connection to the *SHT3XService* on the Raspberry Pi.

smartgadget.**dewpoint**(*temperature*, *humidity*)

Calculate the dew point.

Parameters

- **temperature** (float) – The temperature [degree C].
- **humidity** (float) – The humidity [%RH].

Returns
float – The dew point [degree C].

smartgadget.**kill_manager**(*, *host='raspberrypi'*, *rpi_username='pi'*, *rpi_password=None*, *timeout=10*, ***kwargs*)

Kill the Network Manager on the Raspberry Pi.

Parameters

- **host** (str, optional) – The hostname or IP address of the Raspberry Pi.
- **rpi_username** (str, optional) – The username for the Raspberry Pi.
- **rpi_password** (str, optional) – The password for *rpi_username*.
- **timeout** (float, optional) – The maximum number of seconds to wait for the connection.
- **kwargs** – Keyword arguments that are passed to connect().

smartgadget.**milliseconds_to_datetime**(*milliseconds*)

Convert a timestamp in milliseconds to a datetime.

Parameters
milliseconds (int) – A timestamp in milliseconds.

Returns
datetime – The *milliseconds* converted to a datetime object.

smartgadget.**scan**(*, *interface=0*, *delegate=None*, *timeout=10*, *passive=False*)

Scan for Bluetooth devices.

This function can only be called if the *bluepy* package is installed.

Parameters

- **interface** (int, optional) – The Bluetooth interface to use (where 0 is /dev/hci0).
- **delegate** (DefaultDelegate, optional) – Receives a callback when broadcasts from devices are received.
- **timeout** (float, optional) – Scan for devices for the given timeout in seconds. During this period, callbacks to the *delegate* object will be called.

- **passive** (bool, optional) – Use active (to obtain more information when connecting) or passive scanning.

Returns
A *view* of ScanEntry objects.

smartgadget.**start_service_on_rpi**()
Starts the Network Manager and the *SHT3XService*.

This function should only be called from the smartgadget console script (see setup.py).

smartgadget.**timestamp_to_milliseconds**(*obj*)
Convert an object into a timestamp in milliseconds.

Parameters
obj – A datetime object, an ISO-8601 formatted str, a float in seconds, or an int in milliseconds.

Returns
int – The timestamp in milliseconds.

smartgadget.client module

Connect to the Raspberry Pi and link with a Smart Gadget Service.

class smartgadget.client.**SmartGadgetClient**(*service_name*, ***kwargs*)
Bases: LinkedClient

Connect to the Raspberry Pi and link with the SmartGadget Service.

Parameters
- **service_name** (str) – The name of the Service to link with.
- **kwargs** – Keyword arguments that are passed to connect().

disconnect()
Shut down the Smart Gadget Service and the Network Manager.

service_error_handler()
Shut down the Smart Gadget Service and the Network Manager if there was an error.

smartgadget.service module

Base class for a Smart Gadget Service.

class smartgadget.service.**SmartGadgetService**(*cls*, *interface=None*)
Bases: Service

Base class for a Smart Gadget Service.

Parameters
- **cls** – A *SHT3XService* or a *SHTC1Service* class type.
- **interface** (int, optional) – The Bluetooth interface to use for the connection. For example, 0 or None means /dev/hci0, 1 means /dev/hci1.

battery(*mac_address*) → int

Returns the battery level for the specified MAC address.

Parameters
mac_address (str) – The MAC address of the Smart Gadget.

Returns
int – The battery level [%].

connect_gadget(*mac_address*, *strict=True*) → bool

Connect to the specified Smart Gadget.

It is not necessary to call this method to connect to a Smart Gadget via Bluetooth before fetching data from it. The Bluetooth connection will automatically be created and destroyed when requesting information from the Smart Gadget if the Bluetooth connection does not already exist.

Establishing a Bluetooth connection to a Smart Gadget takes approximately 7 seconds. If you are only requesting data from a couple of Smart Gadgets then connecting to each Smart Gadget at the beginning of your script and then fetching data in a loop would be more efficient if you want to fetch data as quickly as possible. However, there are hardware limits to how many Smart Gadgets can simultaneously have a Bluetooth connection with the Raspberry Pi. So, there is a compromise between how quickly your program can fetch data and how many Smart Gadgets you want to fetch data from.

Parameters

- **mac_address** (str) – The MAC address of the Smart Gadget to connect to.
- **strict** (bool, optional) – Whether to raise an error if the Smart Gadget could not be connected to.

Returns
bool – Whether the connection was successful.

connect_gadgets(*mac_addresses*, *strict=True*) → Tuple[list, list]

Connect to the specified Smart Gadgets.

See *connect_gadget()* for more details.

Parameters

- **mac_addresses** (list of str) – A list of MAC addresses of the Smart Gadgets to connect to.
- **strict** (bool, optional) – Whether to raise an error if a Smart Gadget could not be connected to.

Returns
tuple of list – A list of MAC addresses of the Smart Gadgets that were successfully connected to and the MAC addresses of the Smart Gadgets that could not be connected to.

connected_gadgets() → List[str]

Returns the MAC addresses of the Smart Gadgets that are currently connected.

Returns
list of str – The MAC addresses of the currently-connected Smart Gadgets.

dewpoint(*mac_address*, *temperature=None*, *humidity=None*) → float

Returns the dew point for the specified MAC address.

Parameters

- **mac_address** (str) – The MAC address of the Smart Gadget.
- **temperature** (float, optional) – The temperature [degree C]. If None then reads the current temperature value from the Smart Gadget.
- **humidity** (float, optional) – The humidity [%RH]. If None then reads the current humidity value from the Smart Gadget.

Returns

float – The dew point [degree C].

disconnect_gadget(*mac_address*)

Disconnect the Smart Gadget with the specified MAC address.

Parameters

mac_address (str) – The MAC address of the Smart Gadget to disconnect from.

disconnect_gadgets()

Disconnect from all Smart Gadgets.

humidity(*mac_address*) → float

Returns the current humidity for the specified MAC address.

Parameters

mac_address (str) – The MAC address of the Smart Gadget.

Returns

float – The humidity [%RH].

info(*mac_address*) → dict

Returns all available information from the Smart Gadget.

Parameters

mac_address (str) – The MAC address of the Smart Gadget.

Returns

dict – Includes information such as the firmware, hardware and software version numbers, the battery level, the temperature, humidity and dew point values and the timing information about the data logger (if the Smart Gadgets supports logging).

max_attempts() → int

Returns the maximum number of times to try to connect or read/write data from/to a Smart Gadget.

Returns

int – The maximum number of times to retry.

restart_bluetooth()

Restart the Bluetooth driver on the Raspberry Pi.

This can fix scanning issues or connection timeouts.

> **Attention:** Calling this method will disconnect all Smart Gadgets that are currently connected to the Raspberry Pi.

static rpi_date() → str

Returns the current date of the Raspberry Pi.

Returns
str – The current date of the Raspberry Pi in the ISO-8601 format.

rssi(*mac_address*) → int

Returns the Received Signal Strength Indication (RSSI) for the last received broadcast from the device.

This is an integer value measured in dB, where 0 dB is the maximum (theoretical) signal strength, and more negative numbers indicate a weaker signal.

Parameters
mac_address (str) – The MAC address of the Smart Gadget.

Returns
int or None – The RSSI value if the *SmartGadget* was initialized with a ScanEntry object. Otherwise returns None.

scan(*timeout=10*, *passive=False*) → List[str]

Scan for Smart Gadgets that are within Bluetooth range.

Parameters

- **timeout** (float, optional) – The number of seconds to scan for Smart Gadgets.
- **passive** (bool, optional) – Use active (to obtain more information when connecting) or passive scanning.

Returns
list of str – A list of MAC addresses of the Smart Gadgets that are available for this particular SHTxx class.

set_max_attempts(*max_attempts*)

Set the maximum number of times to try to connect or read/write data from/to a Smart Gadget.

Since a Bluetooth connection can drop unexpectedly, this provides the opportunity to automatically re-connect or re-send a request to a Smart Gadget.

Parameters
max_attempts (int) – The maximum number of times to try to connect or read/write data from/to a Smart Gadget. Increasing the number of attempts will decrease the occurrence of getting a BTLEDisconnectError or a BrokenPipeError when sending requests, but may make sending a request take a long time while the connection automatically tries to be re-established.

static set_rpi_date(*date*)

Set the date of the Raspberry Pi.

This is useful if the Raspberry Pi does not have internet access on startup to sync with an online NTP server. Does not set the time zone.

Parameters
date – Can be a datetime object, an ISO-8601 formatted str, a float in seconds, or an int in milliseconds.

shutdown_service()

Shutdown the Smart Gadget Service and the Network Manager.

temperature(*mac_address*) → float

Returns the current temperature for the specified MAC address.

Parameters
mac_address (str) – The MAC address of the Smart Gadget.

Returns
float – The temperature [degree C].

temperature_humidity(*mac_address*) → Tuple[float, float]

Returns the current temperature and humidity for the specified MAC address.

Parameters
mac_address (str) – The MAC address of the Smart Gadget.

Returns

- float – The temperature [degree C].
- float – The humidity [%RH].

temperature_humidity_dewpoint(*mac_address*) → Tuple[float, float, float]

Returns the current temperature, humidity and dew point for the specified MAC address.

Parameters
mac_address (str) – The MAC address of the Smart Gadget.

Returns

- float – The temperature [degree C].
- float – The humidity [%RH].
- float – The dew point [degree C].

smartgadget.sht3x module

class smartgadget.sht3x.**SHT3X**(*device*, *interface=None*)

Bases: *SmartGadget*

Base class for a Smart Gadget.

Parameters

- **device** – A MAC address as a str or a ScanEntry object.
- **interface** (int, optional) – The Bluetooth interface to use for the connection. For example, 0 or None means /dev/hci0, 1 means /dev/hci1.

APPEARANCE_HANDLE = 5

BATTERY_LEVEL_HANDLE = 29

```
DEVICE_NAME = 'Smart Humigadget'

DEVICE_NAME_HANDLE = 3

FIRMWARE_REVISION_STRING_HANDLE = 24

HARDWARE_REVISION_STRING_HANDLE = 22

HUMIDITY_CHARACTERISTIC_UUID = <bluepy.btle.UUID object>

HUMIDITY_HANDLE = 50

HUMIDITY_NOTIFICATION_HANDLE = 52

HUMIDITY_SERVICE_UUID = <bluepy.btle.UUID object>

LOGGER_INTERVAL_MS_CHARACTERISTIC_UUID = <bluepy.btle.UUID object>

LOGGER_INTERVAL_MS_HANDLE = 46

LOGGER_SERVICE_UUID = <bluepy.btle.UUID object>

MANUFACTURER_NAME_STRING_HANDLE = 16

MODEL_NUMBER_STRING_HANDLE = 18

NEWEST_TIMESTAMP_MS_CHARACTERISTIC_UUID = <bluepy.btle.UUID object>

NEWEST_TIMESTAMP_MS_HANDLE = 39

OLDEST_TIMESTAMP_MS_CHARACTERISTIC_UUID = <bluepy.btle.UUID object>

OLDEST_TIMESTAMP_MS_HANDLE = 36

PERIPHERAL_PREFERRED_CONNECTION_PARAMETERS_HANDLE = 7

SERIAL_NUMBER_STRING_HANDLE = 20

SOFTWARE_REVISION_STRING_HANDLE = 26

START_LOGGER_DOWNLOAD_CHARACTERISTIC_UUID = <bluepy.btle.UUID object>

START_LOGGER_DOWNLOAD_HANDLE = 42

SYNC_TIME_MS_CHARACTERISTIC_UUID = <bluepy.btle.UUID object>

SYNC_TIME_MS_HANDLE = 33

SYSTEM_ID_HANDLE = 14

TEMPERATURE_CHARACTERISTIC_UUID = <bluepy.btle.UUID object>

TEMPERATURE_HANDLE = 55

TEMPERATURE_NOTIFICATION_HANDLE = 57

TEMPERATURE_SERVICE_UUID = <bluepy.btle.UUID object>
```

battery() → int

Returns the battery level.

Returns
float – The current battery level [%].

disable_humidity_notifications()

Disable humidity notifications.

disable_temperature_notifications()

Disable temperature notifications.

enable_humidity_notifications()

Enable humidity notifications.

enable_temperature_notifications()

Enable temperature notifications.

fetch_logged_data(*, *enable_temperature=True*, *enable_humidity=True*, *sync=None*, *oldest=None*, *newest=None*, *as_datetime=False*) → Tuple[list, list]

Returns the logged temperature and humidity values.

The maximum number of temperature and humidity values that can be logged is 15872 (for each).

It can take approximately 80 seconds to fetch the maximum amount of data that can be saved in the internal memory of the Smart Gadget.

The data is returned as an N x 2 list:

- The first column is the timestamp → int or datetime
- The second column is the value → float or None (if there was an error downloading the value, see *SHT3XService.fetch_logged_data()* for more details)

Parameters

- **enable_temperature** (bool, optional) – Whether to download the temperature values.
- **enable_humidity** (bool, optional) – Whether to download the humidity values.
- **sync** – Passed to *set_sync_time()*.
- **oldest** – Passed to *set_oldest_timestamp()*.
- **newest** – Passed to *set_newest_timestamp()*.
- **as_datetime** (bool) – If True then return the timestamps as datetime objects otherwise return the timestamps as an int in milliseconds.

Returns

- list – The logged temperature values [degree C].
- list – The logged humidity values [%RH].

humidity() → float

Returns the current humidity.

Returns
`float` – The current humidity [%RH].

humidity_notifications_enabled() → bool

Returns whether humidity notifications are enabled.

Returns
`bool` – Whether humidity notifications are enabled.

info() → dict

Returns all available information from the Smart Gadget.

Returns
`dict` – Includes information such as the firmware, hardware and software version numbers, the battery level, the temperature, humidity and dew point values and the timing information about the data logger.

logger_interval() → int

Returns the data logger interval.

Returns
`int` – The time between log events [milliseconds].

newest_timestamp() → int

Returns the newest timestamp of the data logger.

Returns
`int` – The newest timestamp [milliseconds]. See also *milliseconds_to_datetime()*.

oldest_timestamp() → int

Returns the oldest timestamp of the data logger.

Returns
`int` – The oldest timestamp [milliseconds]. See also *milliseconds_to_datetime()*.

set_logger_interval(*milliseconds*)

Set the data logger interval.

> **Attention:** This will clear all values that are currently saved in memory.

Parameters
milliseconds (`int`) – The time between log events [milliseconds].

set_newest_timestamp(*timestamp*)

Set the newest timestamp of the data logger.

Parameters
timestamp – Can be a `datetime` object, an ISO-8601 formatted `str`, a `float` in seconds, or an `int` in milliseconds.

set_oldest_timestamp(*timestamp*)

Set the oldest timestamp of the data logger.

Parameters
timestamp – Can be a `datetime` object, an ISO-8601 formatted `str`, a `float` in seconds, or an `int` in milliseconds.

set_sync_time(*timestamp=None*)

Sync the timestamps of the data logger.

Parameters
timestamp – Can be a `datetime` object, an ISO-8601 formatted `str`, a `float` in seconds, or an `int` in milliseconds. If `None` then uses the current time of the Raspberry Pi.

temperature() → float

Returns the current temperature.

Returns
`float` – The current temperature [degree C].

temperature_humidity() → Tuple[float, float]

Returns the current temperature and humidity.

Returns

- `float` – The current temperature [degree C].
- `float` – The current humidity [%RH].

temperature_notifications_enabled() → bool

Returns whether temperature notifications are enabled.

Returns
`bool` – Whether temperature notifications are enabled.

class smartgadget.sht3x.**SHT3XService**(*interface=None*)

Bases: *SmartGadgetService*

The `Service` for a *SHT3X* Smart Gadget.

Parameters
interface (`int`, optional) – The Bluetooth interface to use for the connection. For example, 0 or `None` means `/dev/hci0`, 1 means `/dev/hci1`.

disable_humidity_notifications(*mac_address*)

Disable humidity notifications.

Parameters
mac_address (`str`) – The MAC address of the Smart Gadget.

disable_temperature_notifications(*mac_address*)

Disable temperature notifications.

Parameters
mac_address (`str`) – The MAC address of the Smart Gadget.

enable_humidity_notifications(*mac_address*)

Enable humidity notifications.

Parameters
mac_address (str) – The MAC address of the Smart Gadget.

enable_temperature_notifications(*mac_address*)

Enable temperature notifications.

Parameters
mac_address (str) – The MAC address of the Smart Gadget.

fetch_logged_data(*mac_address*, *, *enable_temperature=True*, *enable_humidity=True*, *sync=None*, *oldest=None*, *newest=None*, *as_datetime=False*, *num_iterations=1*) → Tuple[list, list]

Returns the logged temperature and humidity values.

The maximum number of temperature and humidity values that can be logged is 15872 (for each).

It can take approximately 80 seconds per iteration to fetch the maximum amount of data that can be saved in the internal memory of the Smart Gadget.

The data is returned as an N x 2 `list`:

- The first column is the timestamp → `int` or `datetime`
- The second column is the value → `float` or `None` (if there was an error downloading the value)

Parameters

- **mac_address** (str) – The MAC address of the Smart Gadget.
- **enable_temperature** (bool, optional) – Whether to download the temperature values.
- **enable_humidity** (bool, optional) – Whether to download the humidity values.
- **sync** – Passed to *SHT3X.set_sync_time()*.
- **oldest** – Passed to *SHT3X.set_oldest_timestamp()*.
- **newest** – Passed to *SHT3X.set_newest_timestamp()*.
- **as_datetime** (bool) – If `True` then return the timestamps as `datetime` objects otherwise return the timestamps as an `int` in milliseconds. If you are calling this method from a remote computer (i.e., you are not running your script on a Raspberry Pi) then you **must** keep this value as `False` otherwise you will get the following error:

  ```
  TypeError: Object of type datetime is not JSON
  serializable
  ```

  You can convert the timestamps after getting the data from the Raspberry Pi by calling *milliseconds_to_datetime()* on each timestamp.
- **num_iterations** (int, optional) – Bluetooth does not guarantee that all data packets are received by default, its connection principles are equivalent

to the same ones as UDP for computer networks. You can specify the number of times to download the data to fix missing packets.

Returns

- `list` – The logged temperature values [degree C].
- `list` – The logged humidity values [%RH].

humidity_notifications_enabled(*mac_address*) → bool

Returns whether humidity notifications are enabled.

Parameters
mac_address (`str`) – The MAC address of the Smart Gadget.

Returns
`bool` – Whether humidity notifications are enabled.

logger_interval(*mac_address*) → int

Returns the data logger interval.

Parameters
mac_address (`str`) – The MAC address of the Smart Gadget.

Returns
`int` – The time between log events [milliseconds].

newest_timestamp(*mac_address*) → int

Returns the newest timestamp of the data logger.

Parameters
mac_address (`str`) – The MAC address of the Smart Gadget.

Returns
`int` – The newest timestamp [milliseconds]. See also *milliseconds_to_datetime()*.

oldest_timestamp(*mac_address*) → int

Returns the oldest timestamp of the data logger.

Parameters
mac_address (`str`) – The MAC address of the Smart Gadget.

Returns
`int` – The oldest timestamp [milliseconds]. See also *milliseconds_to_datetime()*.

set_logger_interval(*mac_address*, *milliseconds*)

Set the data logger interval.

> **Attention:** This will clear all values that are currently saved in memory.

Parameters

- **mac_address** (`str`) – The MAC address of the Smart Gadget.
- **milliseconds** (`int`) – The time between log events [milliseconds].

set_newest_timestamp(*mac_address*, *timestamp*)

Set the newest timestamp of the data logger.

Parameters

- **mac_address** (str) – The MAC address of the Smart Gadget.
- **timestamp** – Can be a datetime object, an ISO-8601 formatted str, a float in seconds, or an int in milliseconds.

set_oldest_timestamp(*mac_address*, *timestamp*)

Set the oldest timestamp of the data logger.

Parameters

- **mac_address** (str) – The MAC address of the Smart Gadget.
- **timestamp** – Can be a datetime object, an ISO-8601 formatted str, a float in seconds, or an int in milliseconds.

set_sync_time(*mac_address*, *timestamp=None*)

Sync the timestamps of the data logger.

Parameters

- **mac_address** (str) – The MAC address of the Smart Gadget.
- **timestamp** – Can be a datetime object, an ISO-8601 formatted str, a float in seconds, or an int in milliseconds. If None then uses the current time of the Raspberry Pi.

temperature_notifications_enabled(*mac_address*) → bool

Returns whether temperature notifications are enabled.

Parameters
mac_address (str) – The MAC address of the Smart Gadget.

Returns
bool – Whether temperature notifications are enabled.

smartgadget.shtc1 module

class smartgadget.shtc1.**SHTC1**(*device*, *interface=None*)

Bases: *SmartGadget*

Base class for a Smart Gadget.

Parameters

- **device** – A MAC address as a str or a ScanEntry object.
- **interface** (int, optional) – The Bluetooth interface to use for the connection. For example, 0 or None means /dev/hci0, 1 means /dev/hci1.

DEVICE_NAME = 'SHTC1 smart gadget'

TEMPERATURE_HUMIDITY_CHARACTERISTIC_UUID = <bluepy.btle.UUID object>

battery() → int

Returns the battery level.

Returns
`float` – The current battery level [%].

humidity() → float

Returns the current humidity.

Returns
`float` – The current humidity [%RH].

info() → dict

Returns all available information from the Smart Gadget.

Returns
`dict` – Includes information such as the firmware, hardware and software version numbers, the battery level, the temperature, humidity and dew point values.

temperature() → float

Returns the current temperature.

Returns
`float` – The current temperature [degree C].

temperature_humidity() → Tuple[float, float]

Returns the current temperature and humidity.

Returns

- `float` – The current temperature [degree C].
- `float` – The current humidity [%RH].

class smartgadget.shtc1.**SHTC1Service**(*interface=None*)

Bases: *SmartGadgetService*

The `Service` for a *SHTC1* Smart Gadget.

Parameters
interface (`int`, optional) – The Bluetooth interface to use for the connection. For example, 0 or `None` means `/dev/hci0`, 1 means `/dev/hci1`.

smartgadget.smart_gadget module

class smartgadget.smart_gadget.**SmartGadget**(*device*, *interface=None*)

Bases: `Peripheral`

Base class for a Smart Gadget.

Parameters

- **device** – A MAC address as a `str` or a ScanEntry object.
- **interface** (`int`, optional) – The Bluetooth interface to use for the connection. For example, 0 or `None` means `/dev/hci0`, 1 means `/dev/hci1`.

APPEARANCE_CHARACTERISTIC_UUID = <bluepy.btle.UUID object>

BATTERY_LEVEL_CHARACTERISTIC_UUID = <bluepy.btle.UUID object>

DEVICE_NAME_CHARACTERISTIC_UUID = <bluepy.btle.UUID object>

FIRMWARE_REVISION_STRING_CHARACTERISTIC_UUID = <bluepy.btle.UUID object>

HARDWARE_REVISION_STRING_CHARACTERISTIC_UUID = <bluepy.btle.UUID object>

MANUFACTURER_NAME_STRING_CHARACTERISTIC_UUID = <bluepy.btle.UUID object>

MODEL_NUMBER_STRING_CHARACTERISTIC_UUID = <bluepy.btle.UUID object>

PERIPHERAL_PREFERRED_CONNECTION_PARAMETERS_CHARACTERISTIC_UUID = <bluepy.btle.UUID object>

SERIAL_NUMBER_STRING_CHARACTERISTIC_UUID = <bluepy.btle.UUID object>

SOFTWARE_REVISION_STRING_CHARACTERISTIC_UUID = <bluepy.btle.UUID object>

SYSTEM_ID_CHARACTERISTIC_UUID = <bluepy.btle.UUID object>

battery() → int

Returns the temperature [degree C].

> **Attention:** The subclass must override this method.

dewpoint(*temperature=None*, *humidity=None*) → float

Returns the dew point for the specified MAC address.

Parameters

- **temperature** (float, optional) – The temperature [degree C]. If None then reads the current temperature value from the Smart Gadget.
- **humidity** (float, optional) – The humidity [%RH]. If None then reads the current humidity value from the Smart Gadget.

Returns

float – The dew point [degree C].

humidity() → float

Returns the temperature [degree C].

> **Attention:** The subclass must override this method.

info() → dict

Returns the temperature [degree C].

> **Attention:** The subclass must override this method.

rssi() → Optional[int]

Returns the Received Signal Strength Indication (RSSI) for the last received broadcast from the device.

This is an integer value measured in dB, where 0 dB is the maximum (theoretical) signal strength, and more negative numbers indicate a weaker signal.

Returns

`int` or `None` – The RSSI value if the *SmartGadget* was initialized with a ScanEntry object. Otherwise returns `None`.

temperature() → float

Returns the temperature [degree C].

> **Attention:** The subclass must override this method.

temperature_humidity() → Tuple[float, float]

Returns the temperature [degree C].

> **Attention:** The subclass must override this method.

temperature_humidity_dewpoint() → Tuple[float, float, float]

Returns the current temperature, humidity and dew point.

Returns

- `float` – The temperature [degree C].
- `float` – The humidity [%RH].
- `float` – The dew point [degree C].

smartgadget.update_bluez module

BlueZ is a program that is used to communicate with Bluetooth devices on Linux.

At the time of writing this script (2020-03-31) the latest version of BlueZ was 5.54 and therefore this is the default version when running the `bluez-update` console script (see setup.py).

smartgadget.update_bluez.**run**()

Run the updater.

Meant to be invoked via the `bluez-update` command on the terminal.

See *Update BlueZ* for more details.

1.5 Update BlueZ

BlueZ is the program that is used by a Raspberry Pi to communicate with Bluetooth devices. **RPi-SmartGadget** has been tested with versions 5.43, 5.44, 5.47, 5.50, 5.52 and 5.54. Other versions may work as well.

A script is included with **RPi-SmartGadget** that will update the version of BlueZ. Since the **RPi-SmartGadget** package is *installed* in a virtual environment called `shtenv` on the Raspberry Pi we must first activate the virtual environment

From the home directory run

```
source shtenv/bin/activate
```

and then execute

```
bluez-update
```

This will update BlueZ to version 5.54 *(the latest version at the time of writing the bluez-update script)*. To install a specific version of BlueZ (e.g., version 5.47) run

```
bluez-update 5.47
```

1.6 License

```
MIT License

Copyright (c) 2019 - 2022, Measurement Standards Laboratory of New Zealand

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

1.7 Developers

- Joseph Borbely <joseph.borbely@measurement.govt.nz>

1.8 Release Notes

1.8.1 Version 0.1.0 (in development)

PYTHON MODULE INDEX

S

INDEX