adafruit$_{fingerprint}$

Release 1.0.0

Jan 12, 2020

Contents:

Other pages (online)

- project page on github

CHAPTER 1

Adafruit Fingerprint Core Library Documentation

1.1 Core

Core module for serial communication of module data package format

This module implements methods to read and write packets in the data package format. When communicating, the transferring and receiving of command/data/result are all wrapped in data package format. The packets take the shape of package to be sent and received as specified by the adafruit fingerprint module.

1.1.1 Classes

Package Contain methods for serial read and write of module package

class `adafruit_fingerprint.core.`**`Package`**(*port*)

Implemets data package format for adafruit fingerprint module

Contain methods to read and write module package data format to and from the serial buffer. Before every read and write, the package (packet to be read or written) has to be deconstructed and constructed respectively before the read/write operation, to be able to pick out the "package content" which is important to the AdafruitFingerprint class.

`port`

Instance of the Serial class from the serial module. The serial port passed down to allow serial communication (Default is None)

Type serial.Serial

`header`

Pakcage data header value (Default is 0xEF01)

Type int

`address`

Package data address value (Default is 0xFFFFFFFF)

Type int

`identifier`
Package data identifier value. Values can be 01H, 02H or 07H

Type int

`package_head`
a list containing package *header*, *address* and *identifier*

Type list

`read``()`
read package (packet) from the serial buffer

`write``(`*data*`)`
write package (data packet) from the serail buffer

`read_template``()`
read fingerprint template from the serial buffer

`write_template``(`*data*`)`
write fingerprint template to the serial buffer

`read``()`
read package data (packet) from the serial buffer

Returns A list of integers. Unpacked via a specified format from a string of hex bytes

Return type package

`read_template``()`
Read fiingerprint template from serial buffer

Returns

- **template** (*string*) – if fingerprint template is read successfully
- *None* – if no fingerprint template is read from serial buffer

`write``(`*data*`)`
write package data (data packet) from the serail buffer

`write_template``(`*data*`)`
Write fingerprint template to serial buffer

1.2 Interface (AdafruitFingerprint API)

This is the core interface API documentation. This and the *Responses* is all the documentation you need look at to implement a use case like the ones in *Example Codes*

AdafruitFingerprint core API

1.2.1 Classes

class AdafruitFingerprint Implements the core interface of the fingerprint module as methods

`class` `adafruit_fingerprint.interface.`**`AdafruitFingerprint`**`(`*port=None*`)`
Interface class for adafruit fingerprint module

This class implements the methods for interacting with the adafruit fingerprint module as is in the official datasheet

`__init__`(*port=None*)

Initialize class with serial port object

This sets up the *AdafruitFingerprint* class with the serial *port* object to be used for serial communication. The *port* object is passed down during initialization of the *Package* class from the *core* module (which composes this class) where it is actually used. The implementation of the type of serial object *port* must be is not strict. This is why it is left up to the user to select the type of serial object to be used, as can be seen in the examples section.

In development, the `pyserial` package is used. So the constraints are that the serial port object must implement two (2) methods (a `read` and a `write`); which reads and writes from the serial in and out-buffer of the specified port string used when creating the serial connection, and a property `in_waiting`; which checks the in-buffer for waiting (buffered) incoming data, as specified in the pyserial docs. We advise you simply go with the pyserial package for 100% compatibility. We refused to be strict on this, by abstracting away the serial connection entirely from the user (to the core module perhaps), this is so as to have similiarities with the actual adafruit fingerprint library implemented in arduino which accepts a serial port connection (a hardware or software serial).

Parameters **`port`** (*`serial.Serial`*) – Instance of the Serial class from the serial module. The serial port passed to allow serial communication (Default is None)

`package`

Instance of the *Package* class from the local *core* module that describes the complete format for communicating with the adafruit fingerprint module over serial communication. The format describes and composes, and receives and sends the complete read and raw write packets

Type *core.Package*

`delete_char`(*page_id*, *num=1*)

To delete a segment (n) of templates of Flash library started from the specified location (or Page_id).

Parameters

- **`page_id`** (*`int`*) – location in module flash library to start delete from
- **`num`** (*`int`*) – number of templates to be deleted from module flash library (Default is 1)

Raises

- `UnknownConfirmationCodeException` – if no valid confirmation code is received from module
- `SerialReadException` – if no serial data can be read from buffer (from module)

Returns Confirmation code (A response object)

Return type int

`down_char`(*buffer*, *template*)

To download character file or template from upper computer to the Specified buffer of Module.

Parameters

- **`buffer`** (*`int`*) – one of two module CharBuffers used for template storage
- **`template`** (*`str`*) – previously generated template passed down from upper computer

Raises

- `UnknownConfirmationCodeException` – if no valid confirmation code is received from module
- `SerialReadException` – if no serial data can be read from buffer (from module)

Returns Confirmation code (A response object)

Return type int

`empty`()
To delete all the templates in the Flash library

Raises

- `UnknownConfirmationCodeException` – if no valid confirmation code is received from module
- `SerialReadException` – if no serial data can be read from buffer (from module)

Returns Confirmation code (A response object)

Return type int

`gen_img`()
Try detecting finger and store the detected finger image in ImageBuffer while returning successful confirmation code; If there is no finger, returned confirmation code would be "can't detect finger".

Raises

- `UnknownConfirmationCodeException` – if no valid confirmation code is received from module
- `SerialReadException` – if no serial data can be read from buffer (from module)

Returns Confirmation code (A response object)

Return type int

`img_2Tz`(*buffer*)
To generate character file from the original finger image in ImageBuffer and store the file in CharBuffer1 or CharBuffer2.

Parameters **`buffer`** (*`int`*) – one of two module CharBuffers used for template storage

Raises

- `UnknownConfirmationCodeException` – if no valid confirmation code is received from module
- `SerialReadException` – if no serial data can be read from buffer (from module)

Returns Confirmation code (A response object)

Return type int

`reg_model`()
To combine information of character files from CharBuffer1 and CharBuffer2 and generate a template which is stroed back in both CharBuffer1 and CharBuffer2.

Raises

- `UnknownConfirmationCodeException` – if no valid confirmation code is received from module
- `SerialReadException` – if no serial data can be read from buffer (from module)

Returns Confirmation code (A response object)

Return type int

`search`(*buffer*, *page_start=0*, *page_num=255*)
To search the whole finger library or a protion of it for the template that matches the One in CharBuffer1 or CharBuffer2 When found, page_id will be returned.

Parameters

- **`buffer`** (*`int`*) – one of two module CharBuffers used for template storage
- **`page_start`** (*`int, optional`*) – location in module flash library to start search from (Default is 0)
- **`page_num`** (*`int, optional`*) – location in module flash library to end search (Default is 255)

Raises

- `UnknownConfirmationCodeException` – if no valid confirmation code is received from module
- `SerialReadException` – if no serial data can be read from buffer (from module)

Returns

- *tuple* – On success. Confirmation code (A response object), *page_id* where template was fonud, and the confidence score
- *int* – On failure. Confirmation code (A response object)

`store`(*buffer*, *page_id*)

To store the template of specified buffer (Buffer1/Buffer2) at the designated Location (page) of Flash library.

Parameters

- **`buffer`** (*`int`*) – one of two module CharBuffers used for template storage
- **`page_id`** (*`int`*) – designated location in module flash library (0 - 255)

Raises

- `UnknownConfirmationCodeException` – if no valid confirmation code is received from module
- `SerialReadException` – if no serial data can be read from buffer (from module)

Returns Confirmation code (A response object)

Return type int

`template_num`()

To read the current valid template number of the Module

Raises

- `UnknownConfirmationCodeException` – if no valid confirmation code is received from module
- `SerialReadException` – if no serial data can be read from buffer (from module)

Returns

- *tuple* – On success. Confirmation code (A response object), template number count in flash library
- *int* – On failure. Confirmation code (A response object)

`up_char`(*buffer*)

To upload the character file or template of CharBuffer1 or CharBuffer2 to upper computer.

Parameters **`buffer`** (*`int`*) – one of two module CharBuffers used for template storage

Raises

- `UnknownConfirmationCodeException` – if no valid confirmation code is received from module
- `SerialReadException` – if no serial data can be read from buffer (from module)

Returns

- *tuple* – On success. Confirmation code (A response object) and fingerprint template *template*
- *int* – On failure. Confirmation code (A response object)

vfy_pwd()
Verify module's handshaking password

Raises

- `UnknownConfirmationCodeException` – if no valid confirmation code is received from module
- `SerialReadException` – if no serial data can be read from buffer (from module)

Returns Confirmation code (A response object)

Return type int

Note: This has to be the first method to be called on any created instance of this class. It also serves as a checker for proper hardware connections as it first tries to establish communication with the connected module

1.3 Exceptions

Internal Exception classes used by package

These classes subclass the base Exception class

1.3.1 Classes

MissingPortException(Exception) SerialReadException(Exception) UnknownConfirmationCodeException(Exception)

exception `adafruit_fingerprint.exceptions.`**MissingPortException**
Exception raised when the port param is missing when instantiating the AdafruitFingerprin class

exception `adafruit_fingerprint.exceptions.`**SerialReadException**
Exception raised when no data is read from the serial port

exception `adafruit_fingerprint.exceptions.`**UnknownConfirmationCodeException**
Exception raised when package content is an invalid response

1.4 Responses

This is the core response documentation. This and the *Interface (AdafruitFingerprint API)* is all the documentation you need look at to implement a use case like the ones in *Example Codes*.

The *Confirmation code (A response object)* found as most of the returns of the methods of the *AdafruitFingerprint* class implemented in the *Interface (AdafruitFingerprint API)* module are implemeneted here.

Implements fingerprint module confirmation codes as int constants

Defined for the methods of the AdafruitFingerprint class. This module defines the confirmation codes for interfaces of the adafruit fingerprint module. Codes are defined as integer constants.

H here simply means the values are hex values. E.g `0bH` means `0x0b`, and the integer value is 11.

`adafruit_fingerprint.responses.`**`FINGERPRINT_BADLOCATION = 11`**

(Value is 0bH)

addressing PageID is beyond the finger library (store)

fail to delete template from a location (delete_char)

A *store* and *delete_char* response

Type int

`adafruit_fingerprint.responses.`**`FINGERPRINT_ENROLLMISMATCH = 10`**

Fail to combine the character files. That's, the character files don't belong to one finger (Value is 0aH)

A *reg_model* response

Type int

`adafruit_fingerprint.responses.`**`FINGERPRINT_FEATUREFAIL = 7`**

Fail to generate character file due to lackness of character point or over-smallness of fingerprint image (Value is 07H)

A *img_2Tz* response

Type int

`adafruit_fingerprint.responses.`**`FINGERPRINT_FLASHER = 24`**

(Value is 18H)

error when writing to flash library

A *store* and *delete_char* response

Type int

`adafruit_fingerprint.responses.`**`FINGERPRINT_IMAGEFAIL = 3`**

Fail to collect finger (Value is 03H)

A *gen_img* response

Type int

`adafruit_fingerprint.responses.`**`FINGERPRINT_IMAGEMESS = 6`**

Fail to generate character file due to the over-disorderly fingerprint image (Value is 06H)

A *img_2Tz* response

Type int

`adafruit_fingerprint.responses.`**`FINGERPRINT_INVALIDIMAGE = 21`**

Fail to generate the image for the lackness of valid primary image (Value is 15H)

A *img_2Tz* response

Type int

`adafruit_fingerprint.responses.`**`FINGERPRINT_NOFINGER = 2`**
Can't detect finger (Value is 02H)

A *gen_img* response

Type int

`adafruit_fingerprint.responses.`**`FINGERPRINT_NOTFOUND = 9`**
No matching print in the library (both the PageID and matching score are 0) (Value is 09H)

A *search* response

Type int

`adafruit_fingerprint.responses.`**`FINGERPRINT_OK = 0`**
Success for all operations (Value is 00H)

Type int

`adafruit_fingerprint.responses.`**`FINGERPRINT_PACKETRECEIVER = 1`**
Error when receiving package (Value is 01H)

Type int

`adafruit_fingerprint.responses.`**`FINGERPRINT_PASSWORD_OK = 0`**
Correct password (Value is 00H)

A *vry_pwd* response

Type int

`adafruit_fingerprint.responses.`**`FINGERPRINT_TEMPLATECLEARALLFAIL = 17`**
Fail to clear finger library (Value is 11H)

A *empty* response

Type int

`adafruit_fingerprint.responses.`**`FINGERPRINT_TEMPLATEDELETEFAIL = 16`**
Fail to delete templates (Value is 10H)

A *delete_char* response

Type int

`adafruit_fingerprint.responses.`**`FINGERPRINT_TEMPLATEDOWNLOADFAIL = 14`**
error when downloading template (Value is 0eH)

A *down_char* response

Type int

`adafruit_fingerprint.responses.`**`FINGERPRINT_TEMPLATEUPLOADFAIL = 13`**
error when uploading template (Value is 0dH)

A *up_char* response

Type int

`adafruit_fingerprint.responses.`**`FINGERPRINT_WRONG_PASSWORD = 19`**
Wrong Password (Value is 13H)

A *vry_pwd* response

Type int

1.5 Utils

Uitility functions used by core interface

This module contains functions that are unnecessary to go in main core classes and perform specific functions

1.5.1 Functions

hexbyte_2integer_normalizer(first_int_byte, second_int_btye) Function to normalize integer bytes to a single byte

`adafruit_fingerprint.utils.`**`hexbyte_2integer_normalizer`**(*first_int_byte*, *second_int_btye*)

Function to normalize integer bytes to a single byte

Transform two integer bytes to their hex byte values and normalize their values to a single integer

Parameters **`second_int_byte`** (*`first_int_byte,`*) – integer values to normalize (0 to 255)

Returns **integer** – Single normalized integer

Return type int

This part documents the library itself.

Now, the *interface* and *responses* module are most likely the only ones you'll have to deal with to implement code for your use case.

The *Core*, *Exceptions* and *Utils* modules are internal modules used internally by the *adafruit_fingerprint* package itself to build up the interface. The interface exposes the `AdafruitFingerprint` class, which can be imported directly from the package like so `from adafruit_fingerprint import AdafruitFingerprint`. This class is the only object exposed by the package's *__init__* file.

And the responses can be imported like so `from adafruit_fingerprint.responses import *`. You can also decide to import only a particular response like so `from adafruit_fingerprint.responses import FINGERPRINT_OK`.

And this is basically all you need from the library to get started. See the *Example Codes* section to get an idea of how this works.

CHAPTER 2

Example Codes

Below are some popular use cases of the adafruit fingerprint module and their sample codes using this library

2.1 Enroll To Flash Library

This sample code shows how to enroll a fingerprint and store it in a particular location in the module flash library

```
# Standard library imports
import sys
from time import sleep

# Third party imports
import serial

# Adafruit package imports
from adafruit_fingerprint import AdafruitFingerprint
from adafruit_fingerprint.responses import *


def main():
    # Attempt to connect to serial port
    try:
        port = '/dev/ttyUSB0'  # USB TTL converter port
        baud_rate = '57600'
        serial_port = serial.Serial(port, baud_rate)
    except Exception as e:
        print(e)
        sys.exit()

    # Initialize sensor library with serial port connection
    finger = AdafruitFingerprint(port=serial_port)

    response = finger.vfy_pwd()
```

(continues on next page)

(continued from previous page)

```
    if response is not FINGERPRINT_PASSWORD_OK:
        print('Did not find fingerprint sensor :(')
        sys.exit()
    print('Found Fingerprint Sensor!\n')

    while True:
        print('\nReady to enroll a fingerprint!\n')
        print('Please type in the ID # (from 1 to 255) you want to save this finger␣
↪as...')
        id = read_number()
        print(f'Enrolling id #{id}\n')
        while not enroll_to_flash_library(finger=finger, id=id):
            break


def read_number():
    num = 0
    while num < 1 or num > 255:
        try:
            num = int(input())
        except ValueError:
            print('Please provide an integer')
        else:
            if num < 1 or num > 255:
                print('Please provide an integer in the above range')

    return num


def enroll_to_flash_library(finger, id):
    CHAR_BUFF_1 = 0x01
    CHAR_BUFF_2 = 0x02

    print('Waiting for a valid finger to enroll\n')
    sys.stdout.flush()

    # Read finger the first time
    response = -1
    while response is not FINGERPRINT_OK:
        response = finger.gen_img()
        if response is FINGERPRINT_OK:
            print('Image taken')
            sys.stdout.flush()
        elif response is FINGERPRINT_NOFINGER:
            print('waiting...')
            sys.stdout.flush()
        elif response is FINGERPRINT_PACKETRECEIVER:
            print('Communication error')
            sys.stdout.flush()
        elif response is FINGERPRINT_IMAGEFAIL:
            print('Imaging Error')
            sys.stdout.flush()
        else:
            print('Unknown Error')
            sys.stdout.flush()

    response = finger.img_2Tz(buffer=CHAR_BUFF_1)
```

(continues on next page)

(continued from previous page)

```
    if response is FINGERPRINT_OK:
        print('Image Converted')
        sys.stdout.flush()
    elif response is FINGERPRINT_IMAGEMESS:
        print('Image too messy')
        return response
    elif response is FINGERPRINT_PACKETRECEIVER:
        print('Communication error')
        return response
    elif response is FINGERPRINT_FEATUREFAIL:
        print('Could not find fingerprint features')
        return response
    elif response is FINGERPRINT_INVALIDIMAGE:
        print('Could not find fingerprint features')
        return response
    else:
        print('Unknown Error')
        return response

    # Ensure finger has been removed
    print('Remove finger')
    sleep(1)
    response = -1
    while (response is not FINGERPRINT_NOFINGER):
        response = finger.gen_img()

    print('\nPlace same finger again')
    sys.stdout.flush()

    # Read finger the second time
    response = -1
    while response is not FINGERPRINT_OK:
        response = finger.gen_img()
        if response is FINGERPRINT_OK:
            print('Image taken')
            sys.stdout.flush()
        elif response is FINGERPRINT_NOFINGER:
            print('waiting...')
            sys.stdout.flush()
        elif response is FINGERPRINT_PACKETRECEIVER:
            print('Communication error')
            sys.stdout.flush()
        elif response is FINGERPRINT_IMAGEFAIL:
            print('Imaging Error')
            sys.stdout.flush()
        else:
            print('Unknown Error')
            sys.stdout.flush()

    response = finger.img_2Tz(buffer=CHAR_BUFF_2)
    if response is FINGERPRINT_OK:
        print('Image Converted')
        sys.stdout.flush()
    elif response is FINGERPRINT_IMAGEMESS:
        print('Image too messy')
        return response
    elif response is FINGERPRINT_PACKETRECEIVER:
```

(continues on next page)

(continued from previous page)

```
        print('Communication error')
        return response
    elif response is FINGERPRINT_FEATUREFAIL:
        print('Could not find fingerprint features')
        return response
    elif response is FINGERPRINT_INVALIDIMAGE:
        print('Could not find fingerprint features')
        return response
    else:
        print('Unknown Error')
        return response

    print('Remove finger')
    print('\nChecking both prints...\n')
    sys.stdout.flush()

    # Register model
    response = finger.reg_model()
    if response is FINGERPRINT_OK:
        print('Prints matched')
        sys.stdout.flush()
    elif response is FINGERPRINT_PACKETRECEIVER:
        print('Communication error')
        return response
    elif response is FINGERPRINT_ENROLLMISMATCH:
        print('Prints did not match')
        return response
    else:
        print('Unknown Error')
        return response

    response = finger.store(buffer=CHAR_BUFF_2, page_id=id)
    if response is FINGERPRINT_OK:
        print(f'Print stored in id #{id} of flash library\n')
        sys.stdout.flush()
        return response
    if response is FINGERPRINT_PACKETRECEIVER:
        print('Communication error')
        sys.stdout.flush()
        return response
    if response is FINGERPRINT_BADLOCATION:
        print('Could not store in that location')
        sys.stdout.flush()
        return response
    if response is FINGERPRINT_FLASHER:
        print('Error writing to flash')
        sys.stdout.flush()
        return response


# Expose only enroll function from module
__all__ = ['enroll_to_flash_library']


if __name__ == '__main__':
    main()
```

2.2 Enroll To Upper Computer

This sample code shows how to enroll a fingerprint, but instead of storing the template in a particular location in the module flash library, we return the template back to upper computer.

The template could then be stored in a database or something, and could be retrieved later into the module for verification. There's an example that shows how to store templates from upper computer back into a location in the flash library in order to perform a search.

```
# Standard library imports
import sys
from time import sleep

# Third party imports
import serial

# Adafruit package imports
from adafruit_fingerprint import AdafruitFingerprint
from adafruit_fingerprint.responses import *


def main():
    # Attempt to connect to serial port
    try:
        port = '/dev/ttyUSB0'  # USB TTL converter port
        baud_rate = '57600'
        serial_port = serial.Serial(port, baud_rate)
    except Exception as e:
        print(e)
        sys.exit()

    # Initialize sensor library with serial port connection
    finger = AdafruitFingerprint(port=serial_port)

    response = finger.vfy_pwd()
    if response is not FINGERPRINT_PASSWORD_OK:
        print('Did not find fingerprint sensor :(')
        sys.exit()
    print('Found Fingerprint Sensor!\n')

    while True:
        print('\nReady to enroll a fingerprint!\n')
        template = enroll_to_upper_computer(finger=finger)
        if template:
            print(f'Template:: {template}')
        else:
            print('Failed to return template')


def enroll_to_upper_computer(finger):
    # Buffer constants
    _CHAR_BUFF_1 = 0x01
    _CHAR_BUFF_2 = 0x02

    '''
    Enrolls fingerprint, but returns template to upper computer instead
    Of flash library
```

(continues on next page)

(continued from previous page)

```
    '''
    print('Waiting for a valid finger to enroll\n')
    sys.stdout.flush()

    # Read finger the first time
    response = -1
    while response is not FINGERPRINT_OK:
        response = finger.gen_img()
        if response is FINGERPRINT_OK:
            print('Image taken')
            sys.stdout.flush()
        elif response is FINGERPRINT_NOFINGER:
            print('waiting...')
            sys.stdout.flush()
        elif response is FINGERPRINT_PACKETRECEIVER:
            print('Communication error')
            sys.stdout.flush()
        elif response is FINGERPRINT_IMAGEFAIL:
            print('Imaging Error')
            sys.stdout.flush()
        else:
            print('Unknown Error')
            sys.stdout.flush()

    response = finger.img_2Tz(buffer=_CHAR_BUFF_1)
    if response is FINGERPRINT_OK:
        print('Image Converted')
        sys.stdout.flush()
    elif response is FINGERPRINT_IMAGEMESS:
        print('Image too messy')
        return False
    elif response is FINGERPRINT_PACKETRECEIVER:
        print('Communication error')
        return False
    elif response is FINGERPRINT_FEATUREFAIL:
        print('Could not find fingerprint features')
        return False
    elif response is FINGERPRINT_INVALIDIMAGE:
        print('Could not find fingerprint features')
        return False
    else:
        print('Unknown Error')
        return False

    # Ensure finger has been removed
    print('Remove finger')
    sleep(1)
    response = -1
    while (response is not FINGERPRINT_NOFINGER):
        response = finger.gen_img()

    print('\nPlace same finger again')
    sys.stdout.flush()

    # Read finger the second time
    response = -1
    while response is not FINGERPRINT_OK:
```

(continues on next page)

(continued from previous page)

```
        response = finger.gen_img()
        if response is FINGERPRINT_OK:
            print('Image taken')
            sys.stdout.flush()
        elif response is FINGERPRINT_NOFINGER:
            print('waiting...')
            sys.stdout.flush()
        elif response is FINGERPRINT_PACKETRECEIVER:
            print('Communication error')
            sys.stdout.flush()
        elif response is FINGERPRINT_IMAGEFAIL:
            print('Imaging Error')
            sys.stdout.flush()
        else:
            print('Unknown Error')
            sys.stdout.flush()

    response = finger.img_2Tz(buffer=_CHAR_BUFF_2)
    if response is FINGERPRINT_OK:
        print('Image Converted')
        sys.stdout.flush()
    elif response is FINGERPRINT_IMAGEMESS:
        print('Image too messy')
        return False
    elif response is FINGERPRINT_PACKETRECEIVER:
        print('Communication error')
        return False
    elif response is FINGERPRINT_FEATUREFAIL:
        print('Could not find fingerprint features')
        return False
    elif response is FINGERPRINT_INVALIDIMAGE:
        print('Could not find fingerprint features')
        return False
    else:
        print('Unknown Error')
        return False

    print('Remove finger')
    print('\nChecking both prints...\n')
    sys.stdout.flush()

    # Register model
    response = finger.reg_model()
    if response is FINGERPRINT_OK:
        print('Prints matched')
        sys.stdout.flush()
    elif response is FINGERPRINT_PACKETRECEIVER:
        print('Communication error')
        return False
    elif response is FINGERPRINT_ENROLLMISMATCH:
        print('Prints did not match')
        return False
    else:
        print('Unknown Error')
        return False

    # Return template to upper computer
```

(continues on next page)

(continued from previous page)

```python
    response = finger.up_char(buffer=_CHAR_BUFF_2)
    if isinstance(response, tuple) and len(response) == 2 and response[0] is␣
↪FINGERPRINT_OK:
        print('Template created successfully!')
        print('Enrollment done!\n')
        sys.stdout.flush()
        return response[1]
    if response is FINGERPRINT_PACKETRECEIVER:
        print('Communication error')
        return False
    if response is FINGERPRINT_TEMPLATEUPLOADFAIL:
        print('Template upload error')
        return False


# Expose only enroll function from module
__all__ = ['enroll_to_upper_computer']


if __name__ == '__main__':
    main()
```

2.3 Store From Upper Computer

This sample code shows how to store a fingerprint from upper computer into a location in the flash library of the module.

The first part of the code makes use of *enroll_to_upper_computer* function exposed in the sample code from the *Enroll to upper computer* example, as seen in line *13* where it is imported and line *37* where it is used.

This saves us the stress of having to always copy and paste a previously stored template into the command line when the program is been ran.

```python
1   # Standard library imports
2   import sys
3   from time import sleep
4   
5   # Third party imports
6   import serial
7   
8   # Adafruit package imports
9   from adafruit_fingerprint import AdafruitFingerprint
10  from adafruit_fingerprint.responses import *
11  
12  # Example module imports
13  from examples.enroll_to_upper_computer import enroll_to_upper_computer
14  
15  
16  def main():
17      # Attempt to connect to serial port
18      try:
19          port = '/dev/ttyUSB0'  # USB TTL converter port
20          baud_rate = '57600'
21          serial_port = serial.Serial(port, baud_rate)
```

(continues on next page)

(continued from previous page)

```
    except Exception as e:
        print(e)
        sys.exit()

    # Initialize sensor library with serial port connection
    finger = AdafruitFingerprint(port=serial_port)

    response = finger.vfy_pwd()
    if response is not FINGERPRINT_PASSWORD_OK:
        print('Did not find fingerprint sensor :(')
        sys.exit()
    print('Found Fingerprint Sensor!\n')

    while True:
        print('\nReady to enroll a fingerprint!\n')
        template = enroll_to_upper_computer(finger)
        if template:
            print(f'Template:: {template}')
            print(
                '\nPlease type in the ID # (from 1 to 255) you want to save this␣
↪finger as...')
            id = read_number()
            print(f'Storing template to flash library, with id #{id}\n')
            if store_from_upper_computer(finger=finger, template=template, page_
↪id=id):
                print('Finished storing\n')
        else:
            print('Failed to return template')


def read_number():
    num = 0
    while num < 1 or num > 255:
        try:
            num = int(input())
        except ValueError:
            print('Please provide an integer')
        else:
            if num < 1 or num > 255:
                print('Please provide an integer in the above range')

    return num


def store_from_upper_computer(finger, template, page_id):
    # Buffer constants
    CHAR_BUFF_1 = 0x01
    CHAR_BUFF_2 = 0x02

    response = finger.down_char(buffer=CHAR_BUFF_1, template=template)
    if response is FINGERPRINT_OK:
        print('Template downloaded successfully!')
        sys.stdout.flush()
    if response is FINGERPRINT_PACKETRECEIVER:
        print('Communication error')
        return False
    if response is FINGERPRINT_TEMPLATEDOWNLOADFAIL:
```

(continues on next page)

(continued from previous page)

```
            print('Template download error')
            return False

    response = finger.store(buffer=CHAR_BUFF_1, page_id=page_id)
    if response is FINGERPRINT_OK:
        print('Template stored successfully!')
        sys.stdout.flush()
        return page_id
    if response is FINGERPRINT_PACKETRECEIVER:
        print('Communication error')
        return False
    if response is FINGERPRINT_BADLOCATION:
        print('Could not store in that location')
        return False
    if response is FINGERPRINT_FLASHER:
        print('Error writing to flash')
        return False


# Expose only store function from module
__all__ = ['store_from_upper_computer']


if __name__ == '__main__':
    main()
```

2.4 Search Flash Library

This sample code shows how to search the module flash library for a print (template) match. It is based on a confidence score. The search could be for a previously generated template *enrolled directly in the flash library* or one *stored from upper computer* to module flash library.

The template could then be stored in a database or something, and could be retrieved later into the module for verification. There's an example that shows how to store templates from upper computer back into a location in the flash library in order to perform a search.

```
# Standard library imports
import sys
from time import sleep

# Third party imports
import serial

# Adafruit package imports
from adafruit_fingerprint import AdafruitFingerprint
from adafruit_fingerprint.responses import *


def main():
    # Attempt to connect to serial port
    try:
        port = '/dev/ttyUSB0'  # USB TTL converter port
        baud_rate = '57600'
        serial_port = serial.Serial(port, baud_rate)
```

(continues on next page)

(continued from previous page)

```
    except Exception as e:
        print(e)
        sys.exit()

    # Initialize sensor library with serial port connection
    finger = AdafruitFingerprint(port=serial_port)

    response = finger.vfy_pwd()
    if response is not FINGERPRINT_PASSWORD_OK:
        print('Did not find fingerprint sensor :(')
        sys.exit()
    print('Found Fingerprint Sensor!\n')

    print('\nWaiting for valid finger!\n')
    while True:
        response = search(finger=finger, page_id=1, page_num=255)
        if response:
            id, confidence = response
            print(f'Found ID #{id}', end='')
            print(f' with confidence of {confidence}\n')
        sleep(0.1)  # Don't run at full speed


def search(finger, page_id, page_num):
    # Buffer constants
    CHAR_BUFF_1 = 0x01
    CHAR_BUFF_2 = 0x02

    # Read finger the first time
    response = -1
    while response is not FINGERPRINT_OK:
        response = finger.gen_img()
        if response is FINGERPRINT_OK:
            print('Image taken')
            sys.stdout.flush()
        elif response is FINGERPRINT_NOFINGER:
            print('waiting...')
            sys.stdout.flush()
        elif response is FINGERPRINT_PACKETRECEIVER:
            print('Communication error')
            return False
        elif response is FINGERPRINT_IMAGEFAIL:
            print('Imaging Error')
            return False
        else:
            print('Unknown Error')
            return False

    response = finger.img_2Tz(buffer=CHAR_BUFF_1)
    if response is FINGERPRINT_OK:
        print('Image Converted')
        sys.stdout.flush()
    elif response is FINGERPRINT_IMAGEMESS:
        print('Image too messy')
        return False
    elif response is FINGERPRINT_PACKETRECEIVER:
        print('Communication error')
```

(continues on next page)

(continued from previous page)

```
        return False
    elif response is FINGERPRINT_FEATUREFAIL:
        print('Could not find fingerprint features')
        return False
    elif response is FINGERPRINT_INVALIDIMAGE:
        print('Could not find fingerprint features')
        return False
    else:
        print('Unknown Error')
        return False

    response = finger.search(
        buffer=CHAR_BUFF_1, page_start=page_id, page_num=page_num)
    if isinstance(response, tuple) and len(response) == 3 and response[0] is␣
↪FINGERPRINT_OK:
        print('Found a print match!\n')
        return response[1], response[2]
    if response is FINGERPRINT_PACKETRECEIVER:
        print('Communication error\n')
        return False
    if response is FINGERPRINT_NOTFOUND:
        print('Did not find a match\n')
        return False


# Expose only search from module
__all__ = ['search']


if __name__ == '__main__':
    main()
```

2.5 Delete From Flash Library

This sample code shows how to delete a fingerprint template or a segment of templates from flash library.

```
# Standard library imports
import sys

# Third party imports
import serial

# Adafruit package imports
from adafruit_fingerprint import AdafruitFingerprint
from adafruit_fingerprint.responses import *


def main():
    # Attempt to connect to serial port
    try:
        port = '/dev/ttyUSB0'  # USB TTL converter port
        baud_rate = '57600'
        serial_port = serial.Serial(port, baud_rate)
    except Exception as e:
```

(continues on next page)

(continued from previous page)

```
        print(e)
        sys.exit()

    # Initialize sensor library with serial port connection
    finger = AdafruitFingerprint(port=serial_port)

    response = finger.vfy_pwd()
    if response is not FINGERPRINT_PASSWORD_OK:
        print('Did not find fingerprint sensor :(')
        sys.exit()
    print('Found Fingerprint Sensor!\n')

    while True:
        print('\nPlease type in the ID # (from 1 to 255) you want to delete...\n')
        id = read_number()
        print(f'Deleting ID #{id}\n')
        if delete(finger=finger, page_id=id, num=1):
            print(f'Fingerprint at ID #{id} has been successfully deleted.')


def read_number():
    num = 0
    while num < 1 or num > 255:
        try:
            num = int(input())
        except ValueError:
            print('Please provide an integer')
        else:
            if num < 1 or num > 255:
                print('Please provide an integer in the above range')

    return num


def delete(finger, page_id, num):
    response = -1

    response = finger.delete_char(page_id=page_id, num=num)
    if response is FINGERPRINT_OK:
        print('Deleted')
        sys.stdout.flush()
        return page_id
    elif response is FINGERPRINT_PACKETRECEIVER:
        print('Communication error')
    elif response is FINGERPRINT_TEMPLATEDELETEFAIL:
        print('Could not delete')
    elif response is FINGERPRINT_BADLOCATION:
        print('Could not delete in that location')
    elif response is FINGERPRINT_FLASHER:
        print('Error writing to flash')
    else:
        print('Unknown Error')

    return False


__all__ = ['delete']
```

(continues on next page)

(continued from previous page)

```python


if __name__ == '__main__':
    main()
```

2.6 Empty Flash Library

This sample code shows how to delete all fingerprint templates from the flash library.

```python
# Standard library imports
import sys

# Third party imports
import serial

# Adafruit package imports
from adafruit_fingerprint import AdafruitFingerprint
from adafruit_fingerprint.responses import *


def main():
    # Attempt to connect to serial port
    try:
        port = '/dev/ttyUSB0'  # USB TTL converter port
        baud_rate = '57600'
        serial_port = serial.Serial(port, baud_rate)
    except Exception as e:
        print(e)
        sys.exit()

    # Initialize sensor library with serial port connection
    finger = AdafruitFingerprint(port=serial_port)

    response = finger.vfy_pwd()
    if response is not FINGERPRINT_PASSWORD_OK:
        print('Did not find fingerprint sensor :(')
        sys.exit()
    print('Found Fingerprint Sensor!\n')

    while True:
        print('\nPlease type in the ID # (from 1 to 255) you want to delete...\n')
        id = read_number()
        print(f'Deleting ID #{id}\n')
        if delete(finger=finger, page_id=id, num=1):
            print(f'Fingerprint at ID #{id} has been successfully deleted.')


def read_number():
    num = 0
    while num < 1 or num > 255:
        try:
            num = int(input())
        except ValueError:
            print('Please provide an integer')
```

(continues on next page)

(continued from previous page)

```
        else:
            if num < 1 or num > 255:
                print('Please provide an integer in the above range')

    return num


def delete(finger, page_id, num):
    response = -1

    response = finger.delete_char(page_id=page_id, num=num)
    if response is FINGERPRINT_OK:
        print('Deleted')
        sys.stdout.flush()
        return page_id
    elif response is FINGERPRINT_PACKETRECEIVER:
        print('Communication error')
    elif response is FINGERPRINT_TEMPLATEDELETEFAIL:
        print('Could not delete')
    elif response is FINGERPRINT_BADLOCATION:
        print('Could not delete in that location')
    elif response is FINGERPRINT_FLASHER:
        print('Error writing to flash')
    else:
        print('Unknown Error')

    return False


__all__ = ['delete']


if __name__ == '__main__':
    main()
```

2.7 Get Template Num Count

This sample code shows how to get the number of templates registered in the module flash library. The flash library has a total of 255 locations to store templates. The examples code searches all 255 locations and returns the total number of locations that has a template stored in them. Good for knowing how many fingerprints have been *enrolled* into the flash library.

```
# Standard library imports
import sys

# Third party imports
import serial

# Adafruit package imports
from adafruit_fingerprint import AdafruitFingerprint
from adafruit_fingerprint.responses import *


def main():
```

(continues on next page)

(continued from previous page)

```
    # Attempt to connect to serial port
    try:
        port = '/dev/ttyUSB0'  # USB TTL converter port
        baud_rate = '57600'
        serial_port = serial.Serial(port, baud_rate)
    except Exception as e:
        print(e)
        sys.exit()

    # Initialize sensor library with serial port connection
    finger = AdafruitFingerprint(port=serial_port)

    response = finger.vfy_pwd()
    if response is not FINGERPRINT_PASSWORD_OK:
        print('Did not find fingerprint sensor :(')
        sys.exit()
    print('Found Fingerprint Sensor!\n')

    response = get_template_num_count(finger=finger)
    if response:
        _, template_num_count = response
        print(f'Total number of templates stored is #{template_num_count}')


def get_template_num_count(finger):
    response = -1

    response = finger.template_num()
    if isinstance(response, tuple) and len(response) == 2 and response[0] is␣
↪FINGERPRINT_OK:
        return True, response[1]

    if response is FINGERPRINT_PACKETRECEIVER:
        print('Communication error')
    else:
        print('Unknown Error')

    return False


__all__ = ['get_template_num_count']


if __name__ == '__main__':
    main()
```

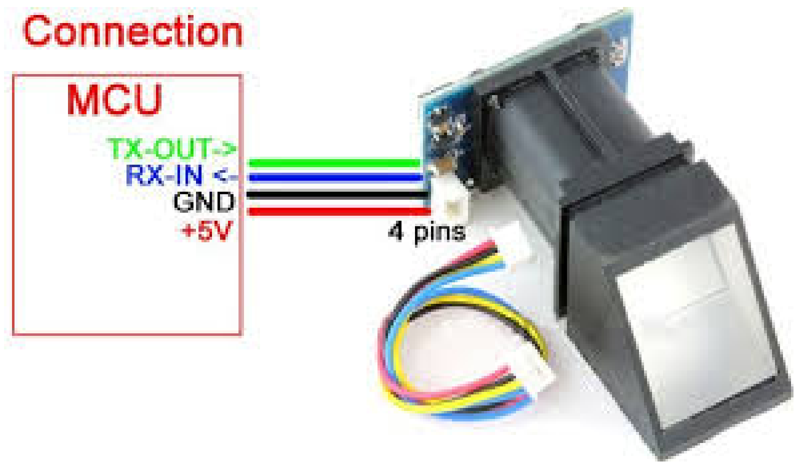
CHAPTER 3

Hardware

This section lists and explains the minimum hardware and connection requirements needed to work with this library.

3.1 Requirements

- **r305 fingerprint module**

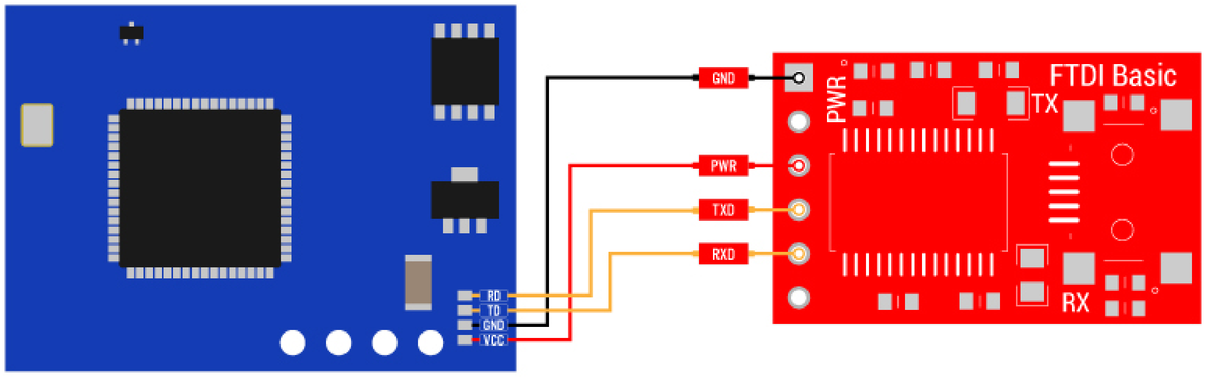


- **USB to TTL converter**

3.2 Connections

The minimum connection to get started is to:

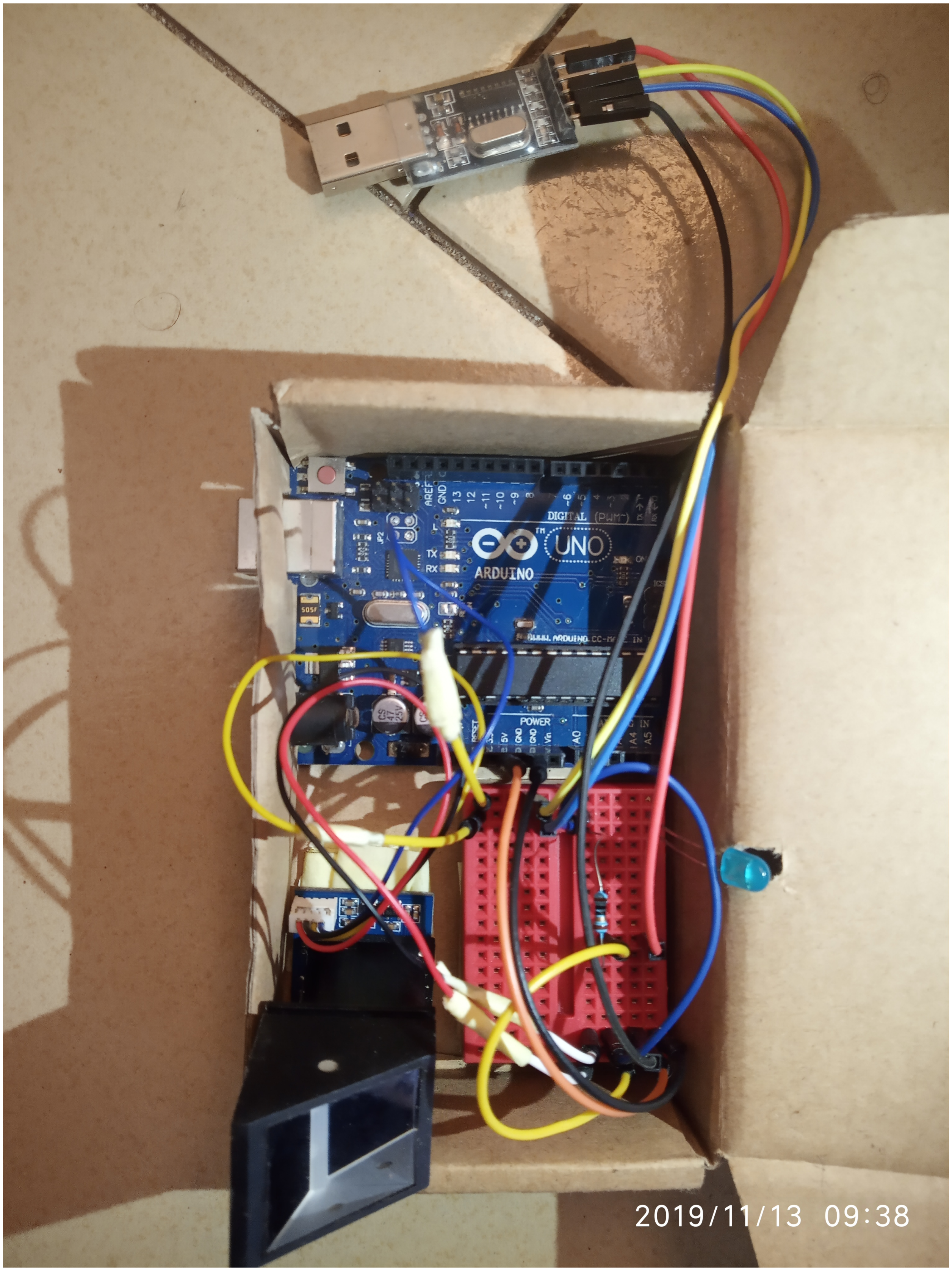
1. Connect the **RX-IN** of the r305 to the **TXD** of the converter
2. Connect the **TX-OUT** of r305 to the **RXD** of the converter
3. Connect both grounds (**GND**), and
4. Connect both powers (**VCCs**)



3.2.1 A Few Checks

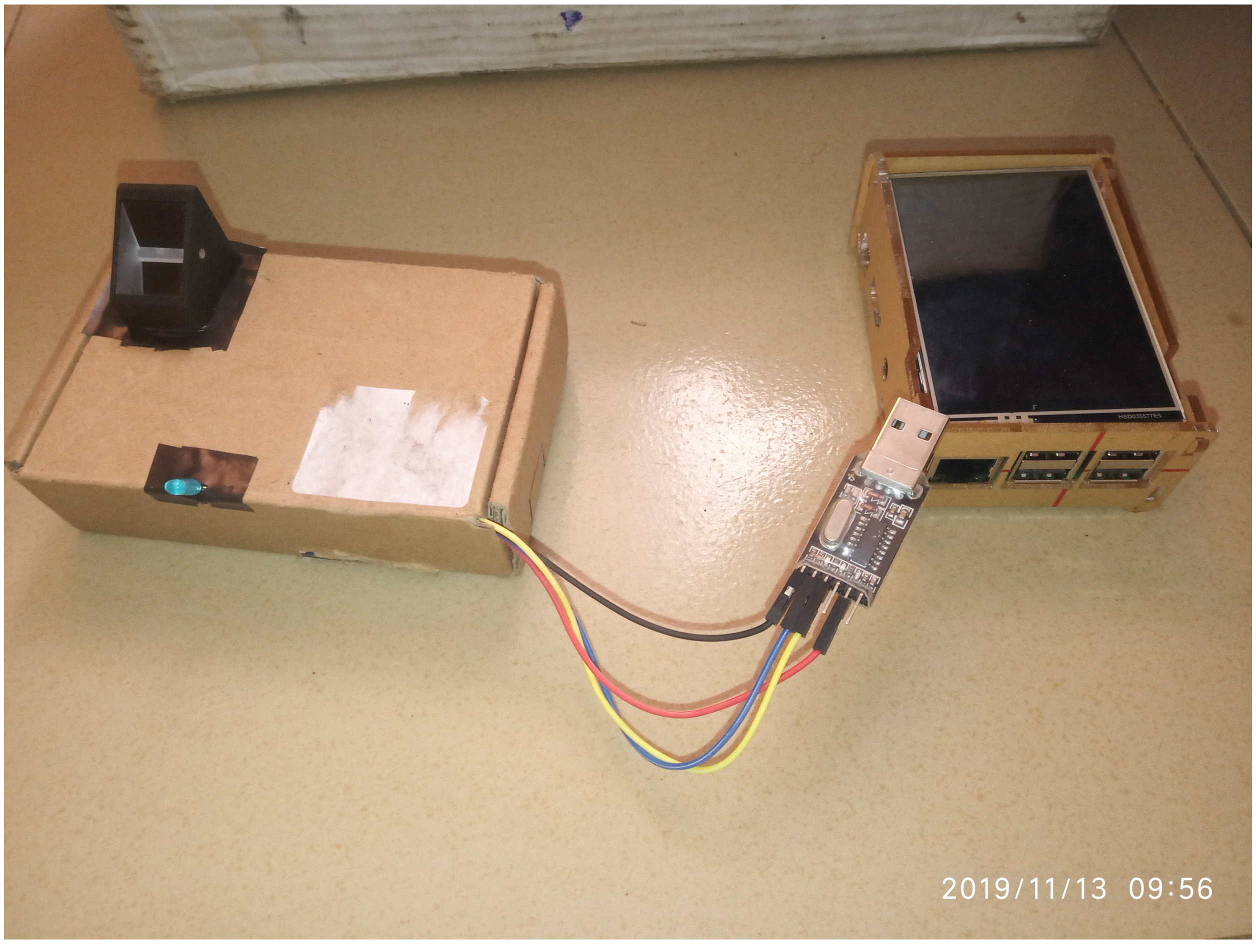
The USB TTL receives power once plugged in. The power from the USB TTL converter may not be enough, depending on your usb port on upper computer (Raspberry or Laptop), to power the r305. Hence you may need to power the r305 with an external power source of 5V.

In the image below, we power the r305 module with an arduino instead, via the 5V VCC on the arduino UNO board, and that's all the arduino is doing there.



The other connections you may see, like the LED and resistor are just connections to indicate that the module is powered, to give a visual on when the whole system is powered on or off.

Below is a complete package using Raspberry Pi 3 as upper computer.



Note: Once the USB TTL converter is connected to upper computer, and you run your program (or e.g any one of the *Example Codes*), depending on your systems settings, you may need to change the permissions of the port to allow serial connection and communication. You might find yourself having to this everytime you unplug and plug back the converter, so you could find a way to automate this.

And that'll be all the hardware and it's connection you need to set up.

It's fair to say that the implementation of this library does not cover all the functionalities of the r305, it does cover a good number of the fingerprint processing insturctions for most use cases. In that regard, if you would want to know more about the r305, its operations, extra capabilities and maybe lower level implementation details of the library, you can have a look in the datasheet, we have a copy on the github repo here.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index