Microhomie Documentation

Release 2.3.0

Microhomie Team

Jan 07, 2021

Contents

CHAPTER 1

Contents

1.1 Get started

Start here with hands-on examples.

1.2 How-to guides

Step-by-step guides for the developer covering key operations and procedures

1.3 Reference

Technical reference - tools, components and commands

1.4 Background

Explanation and discussion of key topics

CHAPTER 2

About this documentation

This documentation is the central hub of information for all things Microhomie.

CHAPTER 3

What is Microhomie?

Microhomie is a MicroPython framework for Homie, a lightweight MQTT convention for the IoT.

Main target for Microhomie is the ESP8266 device but has been well tested and used on ESP32 too.

Source is on GitHub.

CHAPTER 4

Notational Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “NOT RECOMMENDED”, “MAY”, and “OPTIONAL” are to be interpreted as described in [RFC 2119](http://tools.ietf.org/html/rfc2119).

The key words “unspecified”, “undefined”, and “implementation-defined” are to be interpreted as described in the [rationale for the C99 standard](http://www.open-std.org/jtc1/sc22/wg14/www/C99RationaleV5.10.pdf#page=18).

An implementation is not compliant if it fails to satisfy one or more of the MUST, MUST NOT, REQUIRED, SHALL, or SHALL NOT requirements for the protocols it implements. An implementation is compliant if it satisfies all the MUST, MUST NOT, REQUIRED, SHALL, and SHALL NOT requirements for the protocols it implements.

CHAPTER 5

Detailed table of contents

5.1 Get Started

First steps to understand Microhomie and get to know it.

Note: If you just started with MicroPython, a good start is the Getting started with MicroPython on the ESP8266 tutorial from the MicroPython documentation.

5.1.1 Prepare MQTT

Checkout mosquitto MQTT Server (https://mosquitto.org) if you want to host a server yourself or head over to IO Adafruit (https://io.adafruit.com), create an account and use their MQTT API (https://learn.adafruit.com/adafruit-io/mqtt-api).

5.1.2 Install Microhomie on the ESP8266

The first thing you need to do is to load the Microhomie firmware, a modified MicroPython firmware, onto your ESP8266 device.

You can download the Microhomie firmware from the GitHub release page and flash it like any MircoPython firmware to your ESP8266 device. I.E:

```
esptool --port PORT --baud 460800 write_flash --flash_size=detect --verify -fm dio␣
↪0x0 microhomie-esp8266-v2.2.0.bin
```

Continue with the *configuration* and the *Quick start with a simple LED node* sections.

5.1.3 Install Microhomie on the ESP32

Install MicroPython

For now, follow this tutorial (https://www.cnx-software.com/2017/10/16/esp32-micropython-tutorials/) to install MicroPython on the ESP. Please remember to use the latest version of the ESP software during the installation.

Get the latest firmware for ESP32 boards.

Important: On some boards, the installation of MicroPython will fail with an "connection timeout" if you have any wires attached to the board. This depends on your board and you have to detach all wires except power for the basic installation to work. This is only relevant for the installation of MicroPython and not Microhomie.

Once the installation is successful and you are able to execute python code on the ESP32, you can install Microhomie.

Install Microhomie

For the ESP32 you can just copy Microhomie with all requirements to your device.

Clone the Microhomie repository:

```
git clone https://github.com/microhomie/microhomie.git
```

and copy `lib` and `homie` from your host to the device. `homie` should be copied to the device `lib` directory:

```
lib/
├── aswitch.py
├── asyn.py
├── homie
│   ├── __init__.py
│   ├── constants.py
│   ├── device.py
│   ├── network.py
│   ├── node.py
│   ├── property.py
│   └── validator.py
├── primitives
│   ├── __init__.py
│   ├── delay_ms.py
│   ├── message.py
│   ├── pushbutton.py
│   └── switch.py
└── mqtt_as.py
```

For example we have an mpfshell script `esp32_install.mpf` to automate the deployment:

```
mpfshell ttyUSB0 -s esp32_install.mpf
```

Continue with the *configuration* and the *Quick start with a simple LED node* sections.

5.1.4 Configuration

To configure your Microhomie device create a `settings.py` file from the `settings.example.py` file, make your changes and copy the file to your ESP8266 device.

Reference: *settings — Homie Settings*

5.1.5 Quick start with a simple LED node

The LED example in this guide use the on-board LED, so you don't need to do any wiring to get started.

Copy the `main.py` file from the `examples/led` directory to your ESP8266, reset the device and watch the incoming MQTT messages.

If everything is setup correctly, the data will then be pushed to the MQTT server in the homie format. Take a look at the homie specification (https://homieiot.github.io/specification/#) to get an idea of the possibilities.

```
homie/2fe65700/$homie 4.0.0
homie/2fe65700/$name LED test
homie/2fe65700/$state init
homie/2fe65700/$implementation esp8266
homie/2fe65700/$nodes led
homie/2fe65700/led/$name Onboard LED
homie/2fe65700/led/$type LED
homie/2fe65700/led/$properties power
homie/2fe65700/led/power/$name LED Power
homie/2fe65700/led/power/$datatype boolean
homie/2fe65700/led/power/$settable true
homie/2fe65700/$extensions org.homie.legacy-firmware:0.1.1:[4.x],org.homie.legacy-
↪stats:0.1.1:[4.x]
homie/2fe65700/$localip 10.42.0.3
homie/2fe65700/$mac 80:7d:3a:bb:c7:8a
homie/2fe65700/$fw/name Microhomie
homie/2fe65700/$fw/version 2.2.0
homie/2fe65700/$stats/interval 60
homie/2fe65700/led/power true
homie/2fe65700/$stats/uptime 0
homie/2fe65700/$stats/freeheap 15520
homie/2fe65700/$state ready
```

To turn the on-board LED from your ESP8266 on or off send `true` or `false` to the property topic. For example:

```
mosquitto_pub -h HOST -u USER -P PASSWORD --qos 1 -t "homie/DEVICE-ID/led/power/set" -
↪m false
```

5.2 How-to guides

5.2.1 Build your own Microhomie ESP8266 firmware

If you want to build your own Microhomie firmware, maybe for helping us with development, learning or just for the fun to build your own firmware, follow the next steps.

First clone the Microhomie repository:

```
git clone https://github.com/microhomie/microhomie.git
```

The next step is to setup the build environment, build the esp-open-sdk (Requirements and Dependencies), get the MicroPython source, prepare it for the Microhomie firmware and download the required MicroPython modules:

```
cd microhomie
make bootstrap
```

Now you can build your Microhomie firmware and load it to your ESP8266:

```
make
```

Erase and flash:

```
make delpoy PORT=/dev/ttyUSBX
```

Just flash:

```
make flash PORT=/dev/ttyUSBX
```

If you want to help with development, please use our linting:

```
make lint
```

5.3 Reference

Technical reference

5.3.1 `homie.device` — Homie Device

This module provides an interface to the Homie device definition.

Consider to read the Homie convention for details.

Decorator

homie.device.**get_unique_id**()
: Returns a device unique id.

homie.device.**await_ready_state**(*func*)
: Is a async decorator to block coroutines as long as the device has published all device topics and announced itself as ready.

  Decorate methods with this if the method should wait until the device is ready.

class HomieDevice()

The HomieDevice object is the core that handles all incoming and outgoing messages in the way the homie convention is defined.

One HomieDevice object can have multiple HomieNode objects. Microhomie can run with only the device object without nodes but the homie convention requires as minimum one node per device.

Usage Model:

```
import settings

from mynode import MyNode
from homie.node import HomieNode


homie = HomieDevice(settings)
homie.add_node(MyNode)
homie.run_forever()
```

Constructor

class homie.device.**HomieDevice**(*settings*)
Construct a Homie device object. The arguments are:

- `settings` is the settings module from the settings.py file.

Methods

HomieDevice.**add_node**(*self*, *node*)
This method is used to register a HomieNode object to the device.

The arguments are:

- `node` is the HomieNode object.

HomieDevice.**all_properties**(*self*, *func*, *tup_args*)
Run a property class function on all registered property objects

The arguments are:

- `func` is the homie.property.HomieProperty function which should run
- `tup_args` function args as a tuple

HomieDevice.**subscribe**(*self*, *topic*)
Async method to subscribes to the given topic.

The arguments are:

- `topic` is the topic that should be subscribed to.

HomieDevice.**unsubscribe**(*self*, *topic*)
Async method to unsubscribe from the given topic.

The arguments are:

- `topic` is the topic that should be unsubscribed.

HomieDevice.**connection_handler**(*self*, *client*)
Internal async method that gets called when the mqtt connection is established. This method subscribes to all the topics.

The arguments are:

- `client` is the mqtt_as client object.

HomieDevice.**sub_cb**(*self*, *topic*, *payload*, *retained=True*)
This method is the base callback method for arriving messages. Every message arrives on a subscribed topic calls this method.

This method test if the topic is a broadcast topic and pass the message to all nodes broadcast_callback method.

Else the payload will be passed to the node that has subscribed to the messages topic.

The arguments are:

- `topic` is the topic the message has arrived on.
- `payload` is a binary string with the message payload.
- `retained` indicates if the messages is retained on the broker. For Homie this is per default ‘‘True‘´.

`HomieDevice.`**`publish`**(*self*, *topic*, *payload*, *retained=True*)

This async method is used to publish data. Topics will be prefixed with the device base topic.

The arguments are:

- `topic` the sub-topic the payload should be published to.
- `payload` is the payload.
- `retained` indicates if the message should be retained on the broker. Convention default is True.

`HomieDevice.`**`broadcast`**(*self*, *payload*, *level=None*)

This async method can be used to send payload to the Homie broadcast topic. If the level argument is not None, it will be attached as a sub-topic to the broadcast topic.

The arguments are:

- `payload` the payload to send.
- `level` is the broadcast level for the payload. Default is no level.

`HomieDevice.`**`broadcast_callback`**(*self*, *topic*, *payload*, *retained*)

Gets called when the broadcast topic receives a message. Implement

The arguments are:

- `topic` the sub-topic the payload should be published to.
- `payload` is the payload.
- `retained` indicates if the message should be retained on the broker. Convention default is True.

`HomieDevice.`**`publish_properties`**(*self*)

This async method publish the device properties as defined in the Homie convention.

`HomieDevice.`**`publish_stats`**(*self*)

This is a async coroutine to publish device stats as in the homie extension `org.homie.legacy-stats:0.1.1:[4.x]`.

`HomieDevice.`**`run`**(*self*)

This async method is the main loop. It handles the mqtt connection and tries to reconnect if there is an error.

If the RTC is set to `webrepl`, the mainloop will not start to not block the WebREPL.

`HomieDevice.`**`run_forever`**(*self*)

This method should be called from main to start the device.

`HomieDevice.wdt(self):`

This async method is a loop that feeds a watch dog timer. To disable the WDT set DEBUG to True in the settings.py file.

`HomieDevice.`**`dprint`**(*self*)

This method will print to stdout if DEBUG is enabled.

HomieDevice.**setup_wifi**(*self*)
Method that try to connect to a wifi nearby if multible wifi credentials are present in `settings.py`.

5.3.2 `homie.node` — Homie Node

This module provides an interface to the Homie node definition.

class HomieNode()

The HomieNode object defines the Node as in the Homie convention and should be sub-classed to make nodes. A node must be attached to a Homie device object and can has multiple HomieNodeProperty's. A node object can have multiple property objects.

Usage Model:

```
from machine import Pin

from homie.node import HomieNode
from homie.constants import FALSE; TRUE, BOOLEAN
from homie.property import HomieProperty

# reversed values for the esp8266 boards onboard led
ONOFF = {FALSE: 1, TRUE: 0, 1: FALSE, 0: TRUE}

class LEDTestNode(HomieNode):
    def __init__(self, id="led", name="LED test node", type="LED")
        super().__init__(id, name, type)

        # create the esp8266 on-board led object
        self.led = Pin(2, Pin.OUT, value=0)

        # add a homie node property
        self.p_power = HomieProperty(
            id="power",
            name="LED power",
            settable=True,
            datatype=BOOLEAN,
            default=TRUE,
        )
        self.add_property(self.power_property, self.on_power_msg)

    def on_power_message(self, topic, payload, retained):
        self.led(ONOFF[payload]
```

Properties

HomieNode.**device**
This property will be set to the device object, when a node is registered to the device. With this the methods in the device object can be called from the node object.

HomieNode.**properties**
Array that contains all the properties associated to the node.

Constructor

class homie.node.**HomieNode**(*id*, *name*, *type*)
Construct a HomieNode object. The arguments are:

- `id` is an unique id for the node.
- `name` is ne human readable node name.
- `type` is the node type.

Methods

HomieNode.**set_topic**(*self*)
Set the node base topic. This method will be called from the HomieDevice class on device setup.

HomieNode.**add_property**(*self*, *p*, *cb=None*)
This method adds HomieNodeProperty objects to the node object. The arguments are:

- `p` is a HomieNodeProperty object.
- `cb` can be an optional message handler that gets called if the property topic receive a message.

HomieNode.**publish_properties**(*self*)
This method gets called from the device object on device start and publish all properties registered with the node to MQTT.

This is an async method.

5.3.3 homie.property — Homie Property

This module provides an interface to the Homie Node Property definition.

class HomieProperty()

A property object is used to define a homie node property. A property object must be attached to a node object.

Usage Model:

```
from homie.property import HomieProperty
from homie.constants import BOOLEAN, FALSE, TRUE

power_property = HomieProperty(
    id="power",
    name="Power",
    settable=True,
    datatype=BOOLEAN,
    default=TRUE,
)
```

Constructor

class homie.property.**HomieNodeProperty**(*id*, *name=None*, *settable=False*, *retained=True*, *unit=None*, *datatype=STRING*, *format=None*, *default=None*, *restore=True*)
The arguments id, name, settable, retained, unit, datatype and format are arguments from the Homie convention

definition for property attributes with the same defaults.

The arguments default, restore, on_message are Microhomie specific.

The arguments are:

- `id` is mandatory and must be a unique property ID on a per-node basis.
- `name` specifies the a friendly name of the property.
- `settable` allows a property to be settable and enables the /set topic. Default is False.
- `retained` specifies if the property is a retained message. Default is True.
- `unit` optional unit of this proeprty.
- `datatype` specifies the data type as string. Allowed data types are string, integer, float, boolean, enum, color. This types can be importet from *homie.constants*. Default is STRING.
- `format` specifies restrictions or options for the given data type. Default is None.
  - For integer and float: Describes a range of payloads e.g. 10:15
  - For enum: payload,payload,payload for enumerating all valid payloads.
  - **For color:**
    - `rgb` to provide colors in RGB format e.g. 255,255,0 for yellow.
    - `hsv` to provide colors in HSV format e.g. 60,100,100 for yellow.
- `default` set the default message payload. Default is None.
- `restore` restore property payload from mqtt retained message. Default is True.
- `on_message` callback method when the property receives new data.
- `pub_on_upd` publish or not publish when the proptery value changes, default is to publish on each value assignment.

Properties

`HomieProperty.`**`value`**
This is where the property data/payload is stored. The value will be auto published if assigned and `self.pub_on_upd` is `True`. Set `self.pub_on_upd` to `False` to only publish the value when the value changes.

Methods

`HomieProperty.`**`set_topic`**`(`*`self`*`)`
This method generate the homie property topic and will be called when the node is added to the device.

`HomieProperty.`**`publish`**`(`*`self`*`)`
This method publishes the current property value to mqtt. `None` values will not be published.

`HomieProperty.`**`subscribe`**`(`*`self`*`)`
Subscribe to the property topics.

`HomieProperty.`**`restore_handler`**`(`*`self`*, *`topic`*, *`payload`*, *`retained`*`)`
Gets called when the property should be restored from mqtt.

After called, this method removes the restore_handler callback and un-subscribe from the topic. When the restored value is valid it will be assigned to the `value` attribute without publishing the change to mqtt.

The arguments are:

- `topic` the message topic.
- `payload` the message payload.
- `retained` specifies if the payload is retained.

`HomieProperty.`**`message_handler`**(*self*, *topic*, *payload*, *retained*)
Retained messages are not allowed on this topic, if retained message the function will return early.

This method handles incoming payload for the property. Per default this method validates the payload and updates the object value with the new payload.

To overwrite the default handler set a *on_message* handler when adding the property to the node. See HomieNode.add_property().

The arguments are:

- `topic` the message topic.
- `payload` the message payload.
- `retained` specifies if the payload is retained.

`HomieProperty.`**`publish_properties`**(*self*)
This method publishes all homie property attributes to mqtt on device init.

Useful constants

The following constants can be used for the *datatype* argument.

`homie.constants.`**`STRING`**
`homie.constants.`**`BOOLEAN`**
`homie.constants.`**`INTEGER`**
`homie.constants.`**`FLOAT`**
`homie.constants.`**`ENUM`**
`homie.constants.`**`COLOR`**

5.3.4 `homie.utils` — Homie Utils

This module provides helper functions for networking.

`homie.utils.`**`enable_ap`**()
Start the Wifi Access Point with the default configuration.

SSID = Microhomie-<MAC> Secret = microhomiE

`homie.utils.`**`disable_ap`**()
Function to disable the device Wifi Access Point. For the ESP8266 Microhomie firmwares this function will be excecuted on boot.

`homie.utils.`**`get_local_ip`**()
Return the device IP address, if possible.

`homie.utils.`**`get_local_mac`**()
Return the device MAC address, if possible.

`homie.utils.`**`get_wifi_credentials`**(*wifi*)
This function tries to find and a know wifi defined in `settings.py` and returns the credentials.

homie.utils.**payload_is_valid**()
Helper function to validate payload for node properties.

This function does only test if the payload match the datatype and not the format the payload has to be in, for now.

Returns `True` or `False`.

5.3.5 `homie.utils` — Homie Utils

This module provides a homie value validator.

validator.**payload_is_valid**(*cls*, *payload*)
Validate the payload to the datatype and format defined in the property class.

The arguments are:

- `cls` property class object
- `payload` the payload to validate

5.3.6 `homie.constants` — Homie Constants

Node property

Datatype

Constants used for datatypes in NodeProperty.

homie.constants.**STRING**

homie.constants.**ENUM**

homie.constants.**BOOLEAN**

homie.constants.**INTEGER**

homie.constants.**FLOAT**

homie.constants.**COLOR**

Format

homie.constants.**RGB**

homie.constants.**HSV**

General

Contstants used for payload.

homie.constants.**ON**

homie.constants.**OFF**

homie.constants.**TRUE**

homie.constants.**FALSE**

homie.constants.**LOCKED**

homie.constants.**UNLOCKED**

homie.constants.**UTF8**

homie.constants.**SET**

homie.constants.**SLASH**

homie.constants.**UNDERSCORE**

Device

homie.constants.**QOS**
Homie convention specifies QOS to `1`.

homie.constants.**MAIN_DELAY**
This is the delay for the main coro.

homie.constants.**STATS_DELAY**
This is the delay for the stats coro set to `60000`.

homie.constants.**WDT_DELAY**
Feed the WDT every ‘‘100‘‘ms.

homie.constants.**DEVICE_STATE**
Name for the subtobic for device state.

Device states

homie.constants.**STATE_OTA**

homie.constants.**STATE_INIT**

homie.constants.**STATE_READY**

homie.constants.**STATE_RECOVER**

homie.constants.**STATE_WEBREPL**

(Sub-) Tobics

homie.constants.**DEVICE_STATE**
Device state topic `$state`

homie.constants.**T_BC**
Homie broadcast topic `$broadcast`

homie.constants.**T_MPY**
Microhomie extension topic `$mpy`

homie.constants.**T_SET**
`/set` topic

Extensions

homie.constants.**EXT_MPY**
    org.microhomie.mpy:0.1.0:[4.x]

homie.constants.**EXT_FW**
    org.homie.legacy-firmware:0.1.1:[4.x]

homie.constants.**EXT_STATS**
    org.homie.legacy-stats:0.1.1:[4.x]

5.3.7 `settings` — Homie Settings

This module is to configure the home device. For reference see the example settings file in the repository.

Mandatory

All mandatory settings must be set in the settings.py config file on the device.

Wifi settings

settings.**WIFI_SSID**
    SSID for the wifi to connect with. Value must be string.

settings.**WIFI_PASSWORD**
    The password for the wifi to connect with. Value must be string.

Multiple WiFi credentials

settings.**WIFI_CREDENTIALS**
    Microhomie can connect to a knwon wifi nearby. For this feature the `WIFI_CREDENTIALS` dictionary can contain multible wifi credentials in the format `"ssid": "secret"`.

MQTT broker

settings.**MQTT_BROKER**
    The MQTT broker IP address or hostname. Value must be string.

Optional settings

Optional settings have a default value an can be overwritten in the settings file.

Debug

settings.**DEBUG**
    Set DEBUG to `True` to enable debug log output and to disable the WDT. Default ist `False`.

MQTT settings

settings.**MQTT_PORT**
Default port is 1883. Value must be integer.

settings.**MQTT_USERNAME**
The username to connect with. Default is None. Value must be string.

Do not set username for anonymouse auth.

settings.**MQTT_PASSWORD**
The password for the username to connect with. Default is None. Value must be string.

settings.**MQTT_KEEPALIVE**
Default keepalive in seconds is 30. Value must be interger.

settings.**MQTT_SSL**
My only work on ESP32. Default is False. Set to True to enable SSL.

settings.**MQTT_SSL_PARAMS**
Aditional SSL params as dict(). Default is set to {"do_handshake": True}.

settings.**MQTT_BASE_TOPIC**
The base topic for the homie device. Default is "homie". Value must be string.

Device settings

settings.**DEVICE_ID**
The device ID for registration at the broker. The device id is also the base topic of the device and must be unique. Default is to use a generated ID with homie.utils.get_unique_id().

Value must be string and unique.

settings.**DEVICE_NAME**
Friendly name of the device. Value must be string.

settings.**DEVICE_STATS_INTERVAL**
Time in seconds the stats coro publish updates. Default is 60 seconds.

settings.**BROADCAST**
Subscribe to broadcast topic is enabled by default. To disable broadcast messages set BROADCAST to False.

Extensions

settings.**EXTENSIONS**
Default is a empty list() for no extensions. Microhomie currently supports the two legacy extensions and a microhomie extension. Add the extensions to the list to activate them. Items in the list() must be string.

- constants.EXT_MPY for org.microhomie.mpy:0.1.0:[4.x]
- constants.EXT_FW for org.homie.legacy-firmware:0.1.1:[4.x]
- constants.EXT_STATS for org.homie.legacy-stats:0.1.1:[4.x]

Example:

```
EXTENSIONS = [
    EXT_MPY,
    EXT_FW,
    EXT_STATS,
]
```

5.4 Background

5.4.1 External libraries

asyncio primitives

To get you startet the ESP8266 firmware has asyncio primitives from Peter Hinch included

`uasyncio`, `uasyncio.core`, `mqtt_as.py` and `asyn.py` are requiered for Microhomie.

switch

switch.py asyncio switch class

class primitives.switch.**Switch**(*pin*)
Simple debounced switch class for normally open grounded switch.

pushbutton

pushbutton.py asyncio pushbutton class

class primitives.pushbutton.**Pushbutton**(*pin*, *suppress=False*)
Extend the Switch class to support logical state, long press and double-click events

Author: Peter Hinch

mqtt_as.py

mqtt_as.py is a "resilient" asynchronous non-blocking MQTT driver. In Microhomie we use the patched version from Kevin Köck. Kevins version use keywords to initialize the mqtt_as object, support for "unsubscribe" and support for the unix port of MicroPython.

Author: Peter Hinch, Kevin Köck

5.4.2 Recommended input pins

Recommended ESP8266 input pins

The following ESP8266 GPIO pins are recommended for as input pins.

- GPIO4
- GPIO5
- GPIO12

- GPIO13
- GPIO14
- GPIO16

The following ESP8266 GPIO pins should be used with caution. There is a risk that the state of the pins can affect the boot sequence. When possible, use other GPIO pins.

- GPIO0 - used to detect boot-mode. Bootloader runs when pin is low during powerup.
- GPIO2 - used to detect boot-mode. Attached to pull-up resistor.
- GPIO15 - used to detect boot-mode. Attached to pull-down resistor.

One pin does not support interrupts.

- GPIO16 - does not support interrupts.

Recommended ESP32 input pins

The following ESP32 GPIO pins should be used with caution. There is a risk that the state of the pins can affect the boot sequence. When possible, use other GPIO pins.

- GPIO0 - used to detect boot-mode. Bootloader runs when pin is low during powerup. Internal pull-up resistor.
- GPIO2 - used to enter serial bootloader. Internal pull-down resistor.
- GPIO4 - technical reference indicates this is a strapping pin, but usage is not described. Internal pull-down resistor.
- GPIO5 - used to configure SDIO Slave. Internal pull-up resistor.
- GPIO12 - used to select flash voltage. Internal pull-down resistor.
- GPIO15 - used to configure silencing of boot messages. Internal pull-up resistor.

5.5 Indices and tables

- genindex
- search

Python Module Index

h

s

Index

L

M

O

P

Q

R

S

T

U

V

W