python-kasa

python-kasa developers

Dec 04, 2023

CONTENTS

1 Discovering devices **3**

2 Basic controls **5**

3 Energy meter **7**

4 Bulb-specific commands **9**

5 Library usage **11**
- 5.1 Contributing . . . 11
- 5.2 Supported devices . . . 12
- 5.3 Resources . . . 14

6 Command-line usage **15**
- 6.1 Discovery . . . 15
- 6.2 Provisioning . . . 15
- 6.3 `kasa --help` . . . 16

7 Discovering devices **17**
- 7.1 API documentation . . . 17

8 Common API **19**
- 8.1 SmartDevice class . . . 19
- 8.2 Energy Consumption and Usage Statistics . . . 20
- 8.3 API documentation . . . 21

9 Library Design & Modules **27**
- 9.1 Initialization . . . 27
- 9.2 Update Cycle . . . 27
- 9.3 Modules . . . 28
- 9.4 API documentation for modules . . . 28

10 Bulbs **39**
- 10.1 Supported features . . . 39
- 10.2 Currently unsupported . . . 39
- 10.3 Transitions . . . 40
- 10.4 Command-line usage . . . 40
- 10.5 API documentation . . . 40

11 Plugs **49**
- 11.1 API documentation . . . 49

12 Dimmers **55**
12.1 API documentation 55

13 Smart strips **61**
13.1 Command-line usage 61
13.2 API documentation 61

14 Light strips **67**
14.1 API documentation 67

Python Module Index **75**

Index **77**

python-kasa is a Python library to control TPLink's kasa-branded smart home devices (plugs, wall switches, power strips, and bulbs) using asyncio.

This is a voluntary, community-driven effort and is not affiliated, sponsored, or endorsed by TPLink.

You can install the most recent release using pip:

```
pip install python-kasa
```

If you are using cpython, it is recommended to install with `[speedups]` to enable orjson (faster json support):

```
pip install python-kasa[speedups]
```

With `[speedups]`, the protocol overhead is roughly an order of magnitude lower (benchmarks available in devtools).

Alternatively, you can clone this repository and use poetry to install the development version:

```
git clone https://github.com/python-kasa/python-kasa.git
cd python-kasa/
poetry install
```

CHAPTER

ONE

DISCOVERING DEVICES

After installation, the devices can be discovered either by using `kasa discover` or by calling `kasa` without any parameters.

```
$ kasa
No --bulb nor --plug given, discovering..
Discovering devices for 3 seconds
== My Smart Plug - HS110(EU) ==
Device state: ON
IP address: 192.168.x.x
LED state: False
On since: 2017-03-26 18:29:17.242219
== Generic information ==
Time:         1970-06-22 02:39:41
Hardware:     1.0
Software:     1.0.8 Build 151101 Rel.24452
MAC (rssi):   50:C7:BF:XX:XX:XX (-77)
Location:     {'latitude': XXXX, 'longitude': XXXX}
== Emeter ==
Current state: {'total': 133.082, 'power': 100.418681, 'current': 0.510967, 'voltage':␣
↪225.600477}
```

Use `kasa --help` to get list of all available commands, or alternatively, consult the documentation.

CHAPTER

TWO

BASIC CONTROLS

All devices support a variety of common commands, including:

- `state` which returns state information
- `on` and `off` for turning the device on or off
- `emeter` (where applicable) to return energy consumption information
- `sysinfo` to return raw system information

CHAPTER

THREE

ENERGY METER

Passing no options to `emeter` command will return the current consumption. Possible options include `--year` and `--month` for retrieving historical state, and reseting the counters is done with `--erase`.

```
$ kasa emeter
== Emeter ==
Current state: {'total': 133.105, 'power': 108.223577, 'current': 0.54463, 'voltage':␣
↪225.296283}
```

CHAPTER

FOUR

BULB-SPECIFIC COMMANDS

At the moment setting brightness, color temperature and color (in HSV) are supported depending on the device. The commands are straightforward, so feel free to check `--help` for instructions how to use them.

CHAPTER

FIVE

LIBRARY USAGE

You can find several code examples in the API documentation.

5.1 Contributing

Contributions are very welcome! To simplify the process, we are leveraging automated checks and tests for contributions.

5.1.1 Setting up development environment

To get started, simply clone this repository and initialize the development environment. We are using poetry for dependency management, so after cloning the repository simply execute `poetry install` which will install all necessary packages and create a virtual environment for you.

5.1.2 Code-style checks

We use several tools to automatically check all contributions. The simplest way to verify that everything is formatted properly before creating a pull request, consider activating the pre-commit hooks by executing `pre-commit install`. This will make sure that the checks are passing when you do a commit.

You can also execute the checks by running either `tox -e lint` to only do the linting checks, or `tox` to also execute the tests.

5.1.3 Running tests

You can run tests on the library by executing `pytest` in the source directory. This will run the tests against contributed example responses, but you can also execute the tests against a real device:

```
pytest --ip <address>
```

Note that this will perform state changes on the device.

5.1.4 Analyzing network captures

The simplest way to add support for a new device or to improve existing ones is to capture traffic between the mobile app and the device. After capturing the traffic, you can either use the softScheck's wireshark dissector or the `parse_pcap.py` script contained inside the `devtools` directory.

5.2 Supported devices

In principle all devices that are locally controllable using the official Kasa mobile app should work with this library. The following lists merely the devices that have been manually verified to work. If your device is unlisted but working, please open a pull request to update the list and add a fixture file (generated by `devtools/dump_devinfo.py`).

5.2.1 Plugs

- HS100
- HS103
- HS105
- HS107
- HS110
- KP100
- KP105
- KP115
- KP125
- KP125M
- KP401
- EP10

5.2.2 Power Strips

- EP40
- HS300
- KP303
- KP200 (in wall)
- KP400
- KP405 (dimmer)

5.2.3 Wall switches

- ES20M
- HS200
- HS210
- HS220
- KS200M (partial support, no motion, no daylight detection)
- KS220M (partial support, no motion, no daylight detection)
- KS230

5.2.4 Bulbs

- LB100
- LB110
- LB120
- LB130
- LB230
- KL50
- KL60
- KL110
- KL120
- KL125
- KL130
- KL135

5.2.5 Light strips

- KL400
- KL420
- KL430

Contributions (be it adding missing features, fixing bugs or improving documentation) are more than welcome, feel free to submit pull requests!

5.3 Resources

5.3.1 Developer Resources

- softScheck's github contains lot of information and wireshark dissector
- TP-Link Smart Home Device Simulator
- Unofficial API documentation
- Another unofficial API documentation
- pyHS100 provides synchronous interface and is the unmaintained predecessor of this library.

5.3.2 Library Users

- Home Assistant
- MQTT access to TP-Link devices, using python-kasa

5.3.3 TP-Link Tapo support

- PyTapo - Python library for communication with Tapo Cameras
- Tapo P100 (Tapo P105/P100 plugs, Tapo L510E bulbs)
  - Home Assistant integration
- plugp100, another tapo library
  - Home Assistant integration

CHAPTER

SIX

COMMAND-LINE USAGE

The package is shipped with a console tool named `kasa`, refer to `kasa --help` for detailed usage. The device to which the commands are sent is chosen by `KASA_HOST` environment variable or passing `--host <address>` as an option. To see what is being sent to and received from the device, specify option `--debug`.

To avoid discovering the devices when executing commands its type can be passed as an option (e.g., `--type plug` for plugs, `--type bulb` for bulbs, ..). If no type is manually given, its type will be discovered automatically which causes a short delay.

If no command is given, the `state` command will be executed to query the device state.

Note: Some commands (such as reading energy meter values, changing bulb settings, or accessing individual sockets on smart strips) additional parameters are required, which you can find by adding `--help` after the command, e.g. `kasa --type emeter --help` or `kasa --type hsv --help`. Refer to the device type specific documentation for more details.

6.1 Discovery

The tool can automatically discover supported devices using a broadcast-based discovery protocol. This works by sending an UDP datagram on port 9999 to the broadcast address (defaulting to `255.255.255.255`).

On multihomed systems, you can use `--target` option to specify the broadcsat target. For example, if your devices reside in network `10.0.0.0/24` you can use `kasa --target 10.0.0.255 discover` to discover them.

Note: When no command is specified when invoking `kasa`, a discovery is performed and the `state` command is executed on each discovered device.

6.2 Provisioning

You can provision your device without any extra apps by using the `kasa wifi` command:

1. If the device is unprovisioned, connect to its open network
2. Use `kasa discover` (or check the routes) to locate the IP address of the device (likely 192.168.0.1, if unprovisioned)
3. Scan for available networks using `kasa --host 192.168.0.1 wifi scan` see which networks are visible to the device

4. Join/change the network using `kasa --host 192.168.0.1 wifi join <network to join>`

As with all other commands, you can also pass `--help` to both `join` and `scan` commands to see the available options.

6.3 kasa --help

```
Usage: kasa [OPTIONS] COMMAND [ARGS]...

  A tool for controlling TP-Link smart home devices.

Options:
  --host TEXT                     The host name or IP address of the device to
                                  connect to.
  --port INTEGER                  The port of the device to connect to.
  --alias TEXT                    The device name, or alias, of the device to
                                  connect to.
  --target TEXT                   The broadcast address to be used for
                                  discovery.  [default: 255.255.255.255]
  -d, --debug
  --type [plug|bulb|strip|dimmer|lightstrip|tapoplug]
  --json                          Output raw device response as JSON.
  --timeout INTEGER               Timeout for device communications.
  --discovery-timeout INTEGER     Timeout for discovery.
  --username TEXT                 Username/email address to authenticate to
                                  device.
  --password TEXT                 Password to use to authenticate to device.
  --version                       Show the version and exit.
  --help                          Show this message and exit.

Commands:
  alias             Get or set the device (or plug) alias.
  brightness        Get or set brightness.
  discover          Discover devices in the network.
  effect            Set an effect.
  emeter            Query emeter for historical consumption.
  hsv               Get or set color in HSV.
  led               Get or set (Plug's) led state.
  off               Turn the device off.
  on                Turn the device on.
  presets           List and modify bulb setting presets.
  raw-command       Run a raw command on the device.
  reboot            Reboot the device.
  schedule          Scheduling commands.
  state             Print out device state and versions.
  sysinfo           Print out full system information.
  temperature       Get or set color temperature.
  time              Get the device time.
  toggle            Toggle the device on/off.
  turn-on-behavior  Modify bulb turn-on behavior.
  usage             Query usage for historical consumption.
  wifi              Commands to control wifi settings.
```

CHAPTER

SEVEN

DISCOVERING DEVICES

Contents

- *API documentation*

7.1 API documentation

class kasa.**Discover**

Discover TPLink Smart Home devices.

The main entry point for this library is *Discover.discover()*, which returns a dictionary of the found devices. The key is the IP address of the device and the value contains ready-to-use, SmartDevice-derived device object.

discover_single() can be used to initialize a single device given its IP address. If the type of the device and its IP address is already known, you can initialize the corresponding device class directly without this.

The protocol uses UDP broadcast datagrams on port 9999 for discovery.

Examples: Discovery returns a list of discovered devices:

```
>>> import asyncio
>>> found_devices = asyncio.run(Discover.discover())
>>> [dev.alias for dev in found_devices]
['TP-LINK_Power Strip_CF69']
```

Discovery can also be targeted to a specific broadcast address instead of the default 255.255.255.255:

```
>>> asyncio.run(Discover.discover(target="192.168.8.255"))
```

It is also possible to pass a coroutine to be executed for each found device:

```
>>> async def print_alias(dev):
>>>     print(f"Discovered {dev.alias}")
>>> devices = asyncio.run(Discover.discover(on_discovered=print_alias))
```

DISCOVERY_PORT = 9999

DISCOVERY_PORT_2 = 20002

DISCOVERY_QUERY = {'system': {'get_sysinfo': None}}

DISCOVERY_QUERY_2 = b'\x02\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00F<\xb5\xd3'

async static discover(*, target='255.255.255.255', on_discovered=None, timeout=5, discovery_packets=3, interface=None, on_unsupported=None, credentials=None) → Dict[str, kasa.smartdevice.SmartDevice]

Discover supported devices.

Sends discovery message to 255.255.255.255:9999 in order to detect available supported devices in the local network, and waits for given timeout for answers from devices. If you have multiple interfaces, you can use *target* parameter to specify the network for discovery.

If given, *on_discovered* coroutine will get awaited with a SmartDevice-derived object as parameter.

The results of the discovery are returned as a dict of SmartDevice-derived objects keyed with IP addresses. The devices are already initialized and all but emeter-related properties can be accessed directly.

Parameters

- **target** – The target address where to send the broadcast discovery queries if multi-homing (e.g. 192.168.xxx.255).
- **on_discovered** – coroutine to execute on discovery
- **timeout** – How long to wait for responses, defaults to 5
- **discovery_packets** – Number of discovery packets to broadcast
- **interface** – Bind to specific interface

Returns dictionary with discovered devices

async static discover_single(*host: str*, *, *port: Optional[int] = None*, *timeout=5*, *credentials: Optional[kasa.credentials.Credentials] = None*, *update_parent_devices: bool = True*) → *kasa.smartdevice.SmartDevice*

Discover a single device by the given IP address.

It is generally preferred to avoid *discover_single()* and use connect_single() instead as it should perform better when the WiFi network is congested or the device is not responding to discovery requests.

Parameters

- **host** – Hostname of device to query
- **port** – Optionally set a different port for the device
- **timeout** – Timeout for discovery
- **credentials** – Credentials for devices that require authentication
- **update_parent_devices** – Automatically call device.update() on devices that have children

Return type *SmartDevice*

Returns Object for querying/controlling found device.

CHAPTER

EIGHT

COMMON API

Contents

- *SmartDevice class*
- *Energy Consumption and Usage Statistics*
  - *Energy Consumption*
  - *Usage statistics*
- *API documentation*

8.1 SmartDevice class

The basic functionalities of all supported devices are accessible using the common *SmartDevice* base class.

The property accesses use the data obtained before by awaiting *SmartDevice.update()*. The values are cached until the next update call. In practice this means that property accesses do no I/O and are dependent, while I/O producing methods need to be awaited. See *Library Design & Modules* for more detailed information.

Note: The device instances share the communication socket in background to optimize I/O accesses. This means that you need to use the same event loop for subsequent requests. The library gives a warning ("Detected protocol reuse between different event loop") to hint if you are accessing the device incorrectly.

Methods changing the state of the device do not invalidate the cache (i.e., there is no implicit *SmartDevice.update()* call made by the library). You can assume that the operation has succeeded if no exception is raised. These methods will return the device response, which can be useful for some use cases.

Errors are raised as `SmartDeviceException` instances for the library user to handle.

Simple example script showing some functionality:

```
import asyncio
from kasa import SmartPlug

async def main():
    p = SmartPlug("127.0.0.1")

    await p.update()  # Request the update
```

(continues on next page)

(continued from previous page)

```
    print(p.alias)  # Print out the alias
    print(p.emeter_realtime)  # Print out current emeter status

    await p.turn_off()  # Turn the device off

if __name__ == "__main__":
    asyncio.run(main())
```

If you want to perform updates in a loop, you need to make sure that the device accesses are done in the same event loop:

```
import asyncio
from kasa import SmartPlug

async def main():
    dev = SmartPlug("127.0.0.1")  # We create the instance inside the main loop
    while True:
        await dev.update()  # Request an update
        print(dev.emeter_realtime)
        await asyncio.sleep(0.5)  # Sleep some time between updates

if __name__ == "__main__":
    asyncio.run(main())
```

Refer to device type specific classes for more examples: *SmartPlug*, *SmartBulb*, *SmartStrip*, *SmartDimmer*, *SmartLightStrip*.

8.2 Energy Consumption and Usage Statistics

Note: In order to use the helper methods to calculate the statistics correctly, your devices need to have correct time set. The devices use NTP and public servers from NTP Pool Project to synchronize their time.

8.2.1 Energy Consumption

The availability of energy consumption sensors depend on the device. While most of the bulbs support it, only specific switches (e.g., HS110) or strips (e.g., HS300) support it. You can use *has_emeter* to check for the availability.

8.2.2 Usage statistics

You can use *on_since* to query for the time the device has been turned on. Some devices also support reporting the usage statistics on daily or monthly basis. You can access this information using through the usage module (`kasa.modules.Usage`):

```
dev = SmartPlug("127.0.0.1")
usage = dev.modules["usage"]
print(f"Minutes on this month: {usage.usage_this_month}")
print(f"Minutes on today: {usage.usage_today}")
```

8.3 API documentation

class kasa.**SmartDevice**(*host: str*, ***, *port: Optional[int] = None*, *credentials: Optional[kasa.credentials.Credentials] = None*, *timeout: Optional[int] = None*)

Base class for all supported device types.

You don't usually want to initialize this class manually, but either use *Discover* class, or use one of the subclasses:

- *SmartPlug*
- *SmartBulb*
- *SmartStrip*
- *SmartDimmer*
- *SmartLightStrip*

To initialize, you have to await *update()* at least once. This will allow accessing the properties using the exposed properties.

All changes to the device are done using awaitable methods, which will not change the cached values, but you must await update() separately.

Errors reported by the device are raised as SmartDeviceExceptions, and should be handled by the user of the library.

Examples:

```
>>> import asyncio
>>> dev = SmartDevice("127.0.0.1")
>>> asyncio.run(dev.update())
```

All devices provide several informational properties:

```
>>> dev.alias
Kitchen
>>> dev.model
HS110(EU)
>>> dev.rssi
-71
>>> dev.mac
50:C7:BF:01:F8:CD
```

Some information can also be changed programmatically:

```
>>> asyncio.run(dev.set_alias("new alias"))
>>> asyncio.run(dev.set_mac("01:23:45:67:89:ab"))
>>> asyncio.run(dev.update())
>>> dev.alias
new alias
>>> dev.mac
01:23:45:67:89:ab
```

When initialized using discovery or using a subclass, you can check the type of the device:

```
>>> dev.is_bulb
False
>>> dev.is_strip
False
>>> dev.is_plug
True
```

You can also get the hardware and software as a dict, or access the full device response:

```
>>> dev.hw_info
{'sw_ver': '1.2.5 Build 171213 Rel.101523',
 'hw_ver': '1.0',
 'mac': '01:23:45:67:89:ab',
 'type': 'IOT.SMARTPLUGSWITCH',
 'hwId': '45E29DA8382494D2E82688B52A0B2EB5',
 'fwId': '00000000000000000000000000000000',
 'oemId': '3D341ECE302C0642C99E31CE2430544B',
 'dev_name': 'Wi-Fi Smart Plug With Energy Monitoring'}
>>> dev.sys_info
```

All devices can be turned on and off:

```
>>> asyncio.run(dev.turn_off())
>>> asyncio.run(dev.turn_on())
>>> asyncio.run(dev.update())
>>> dev.is_on
True
```

Some devices provide energy consumption meter, and regular update will already fetch some information:

```
>>> dev.has_emeter
True
>>> dev.emeter_realtime
<EmeterStatus power=0.983971 voltage=235.595234 current=0.015342 total=32.448>
>>> dev.emeter_today
>>> dev.emeter_this_month
```

You can also query the historical data (note that these needs to be awaited), keyed with month/day:

```
>>> asyncio.run(dev.get_emeter_monthly(year=2016))
{11: 1.089, 12: 1.582}
>>> asyncio.run(dev.get_emeter_daily(year=2016, month=11))
{24: 0.026, 25: 0.109}
```

add_module(*name: str*, *module: kasa.modules.module.Module*)

Register a module.

property alias: str

Return device name (alias).

async static connect(*host: str*, ***, *port: Optional[int] = None*, *timeout=5*, *credentials: Optional[kasa.credentials.Credentials] = None*, *device_type: Optional[kasa.device_type.DeviceType] = None*) → *kasa.smartdevice.SmartDevice*

Connect to a single device by the given IP address.

This method avoids the UDP based discovery process and will connect directly to the device to query its type.

It is generally preferred to avoid `discover_single()` and use this function instead as it should perform better when the WiFi network is congested or the device is not responding to discovery requests.

The device type is discovered by querying the device.

Parameters

- **host** – Hostname of device to query
- **device_type** – Device type to use for the device. If not given, the device type is discovered by querying the device. If the device type is already known, it is preferred to pass it to avoid the extra query to the device to discover its type.

Return type *SmartDevice*

Returns Object for querying/controlling found device.

async current_consumption() → float

Get the current power consumption in Watt.

property device_id: str

Return unique ID for the device.

If not overridden, this is the MAC address of the device. Individual sockets on strips will override this.

property device_type: kasa.device_type.DeviceType

Return the device type.

property emeter_realtime: kasa.emeterstatus.EmeterStatus

Return current energy readings.

property emeter_this_month: Optional[float]

Return this month's energy consumption in kWh.

property emeter_today: Optional[float]

Return today's energy consumption in kWh.

emeter_type = 'emeter'

async erase_emeter_stats() → Dict

Erase energy meter statistics.

property features: Set[str]

Return a set of features that the device supports.

async get_emeter_daily(*year: Optional[int] = None*, *month: Optional[int] = None*, *kwh: bool = True*) → Dict

Retrieve daily statistics for a given month.

Parameters

- **year** – year for which to retrieve statistics (default: this year)
- **month** – month for which to retrieve statistics (default: this month)
- **kwh** – return usage in kWh (default: True)

Returns mapping of day of month to value

async get_emeter_monthly(*year: Optional[int] = None*, *kwh: bool = True*) → Dict

Retrieve monthly statistics for a given year.

Parameters

- **year** – year for which to retrieve statistics (default: this year)
- **kwh** – return usage in kWh (default: True)

Returns dict: mapping of month to value

async get_emeter_realtime() → kasa.emeterstatus.EmeterStatus

Retrieve current energy readings.

get_plug_by_index(*index: int*) → *kasa.smartdevice.SmartDevice*

Return child device for the given index.

get_plug_by_name(*name: str*) → *kasa.smartdevice.SmartDevice*

Return child device for the given name.

async get_sys_info() → Dict[str, Any]

Retrieve system information.

async get_time() → Optional[datetime.datetime]

Return current time from the device, if available.

async get_timezone() → Dict

Return timezone information.

property has_children: bool

Return true if the device has children devices.

property has_emeter: bool

Return True if device has an energy meter.

property hw_info: Dict

Return hardware information.

This returns just a selection of sysinfo keys that are related to hardware.

property internal_state: Any

Return the internal state of the instance.

The returned object contains the raw results from the last update call. This should only be used for debugging purposes.

property is_bulb: bool

Return True if the device is a bulb.

property is_color: bool

Return True if the device supports color changes.

property is_dimmable: bool

Return True if the device is dimmable.

property is_dimmer: bool

Return True if the device is a dimmer.

property is_light_strip: bool

Return True if the device is a led strip.

property is_off: bool

Return True if device is off.

property is_on: bool

Return True if the device is on.

property is_plug: bool

Return True if the device is a plug.

property is_strip: bool

Return True if the device is a strip.

property is_strip_socket: bool

Return True if the device is a strip socket.

property is_variable_color_temp: bool

Return True if the device supports color temperature.

property location: Dict

Return geographical location.

property mac: str

Return mac address.

Returns mac address in hexadecimal with colons, e.g. 01:23:45:67:89:ab

property max_device_response_size: int

Returns the maximum response size the device can safely construct.

property model: str

Return device model.

property on_since: Optional[datetime.datetime]

Return pretty-printed on-time, or None if not available.

async reboot(*delay: int = 1*) → None

Reboot the device.

Note that giving a delay of zero causes this to block, as the device reboots immediately without responding to the call.

property rssi: Optional[int]

Return WiFi signal strength (rssi).

async set_alias(*alias: str*) → None

Set the device name (alias).

async set_mac(*mac*)

Set the mac address.

Parameters **mac** (*str*) – mac in hexadecimal with colons, e.g. 01:23:45:67:89:ab

property state_information: Dict[str, Any]

Return device-type specific, end-user friendly state information.

property supported_modules: List[str]

Return a set of modules supported by the device.

property sys_info: Dict[str, Any]

Return system information.

Do not call this function from within the SmartDevice class itself as @requires_update will be affected for other properties.

property time: datetime.datetime

Return current time from the device.

property timezone: Dict

Return the current timezone.

async turn_off(***kwargs*) → Dict

Turn off the device.

async turn_on(***kwargs*) → Dict

Turn device on.

async update(*update_children: bool = True*)

Query the device to update the data.

Needed for properties that are decorated with *requires_update*.

update_from_discover_info(*info: Dict[str, Any]*) → None

Update state from info from the discover call.

async wifi_join(*ssid*, *password*, *keytype=3*)

Join the given wifi network.

If joining the network fails, the device will return to AP mode after a while.

async wifi_scan() → List[kasa.smartdevice.WifiNetwork]

Scan for available wifi networks.

CHAPTER

NINE

LIBRARY DESIGN & MODULES

This page aims to provide some details on the design and internals of this library. You might be interested in this if you want to improve this library, or if you are just looking to access some information that is not currently exposed.

Contents

- *Initialization*
- *Update Cycle*
- *Modules*
- *API documentation for modules*

9.1 Initialization

Use `discover()` to perform udp-based broadcast discovery on the network. This will return you a list of device instances based on the discovery replies.

If the device's host is already known, you can use to construct a device instance with `connect()`.

When connecting a device with the `connect()` method, it is recommended to pass the device type as well as this allows the library to use the correct device class for the device without having to query the device.

9.2 Update Cycle

When `update()` is called, the library constructs a query to send to the device based on *supported modules*. Internally, each module defines `query()` to describe what they want query during the update.

The returned data is cached internally to avoid I/O on property accesses. All properties defined both in the device class and in the module classes follow this principle.

While the properties are designed to provide a nice API to use for common use cases, you may sometimes want to access the raw, cached data as returned by the device. This can be done using the `internal_state` property.

9.3 Modules

The functionality provided by all *SmartDevice* instances is (mostly) done inside separate modules. While the individual device-type specific classes provide an easy access for the most import features, you can also access individual modules through `kasa.SmartDevice.modules`. You can get the list of supported modules for a given device instance using *supported_modules*.

Note: If you only need some module-specific information, you can call the wanted method on the module to avoid using *update()*.

9.4 API documentation for modules

Module for individual feature modules.

class kasa.modules.**AmbientLight**(*device:* SmartDevice, *module: str*)

Implements ambient light controls for the motion sensor.

call(*method*, *params=None*)

Call the given method with the given parameters.

async current_brightness() → int

Return current brightness.

Return value units.

property data

Return the module specific raw data from the last update.

property enabled: bool

Return True if the module is enabled.

property estimated_query_response_size

Estimated maximum size of query response.

The inheriting modules implement this to estimate how large a query response will be so that queries can be split should an estimated response be too large

property is_supported: bool

Return whether the module is supported by the device.

property presets: dict

Return device-defined presets for brightness setting.

query()

Request configuration.

query_for_command(*query*, *params=None*)

Create a request object for the given parameters.

async set_brightness_limit(*value: int*)

Set the limit when the motion sensor is inactive.

See *presets* for preset values. Custom values are also likely allowed.

async set_enabled(*state: bool*)

Enable/disable LAS.

class kasa.modules.**Antitheft**(*device:* SmartDevice, *module: str*)

Implementation of the antitheft module.

This shares the functionality among other rule-based modules.

call(*method*, *params=None*)

Call the given method with the given parameters.

property data

Return the module specific raw data from the last update.

async delete_all_rules()

Delete all rules.

async delete_rule(*rule: kasa.modules.rulemodule.Rule*)

Delete the given rule.

property estimated_query_response_size

Estimated maximum size of query response.

The inheriting modules implement this to estimate how large a query response will be so that queries can be split should an estimated response be too large

property is_supported: bool

Return whether the module is supported by the device.

query()

Prepare the query for rules.

query_for_command(*query*, *params=None*)

Create a request object for the given parameters.

property rules: List[kasa.modules.rulemodule.Rule]

Return the list of rules for the service.

async set_enabled(*state: bool*)

Enable or disable the service.

class kasa.modules.**Cloud**(*device:* SmartDevice, *module: str*)

Module implementing support for cloud services.

call(*method*, *params=None*)

Call the given method with the given parameters.

connect(*username: str*, *password: str*)

Login to the cloud using given information.

property data

Return the module specific raw data from the last update.

disconnect()

Disconnect from the cloud.

property estimated_query_response_size

Estimated maximum size of query response.

The inheriting modules implement this to estimate how large a query response will be so that queries can be split should an estimated response be too large

get_available_firmwares()

Return list of available firmwares.

property info: kasa.modules.cloud.CloudInfo

Return information about the cloud connectivity.

property is_supported: bool

Return whether the module is supported by the device.

query()

Request cloud connectivity info.

query_for_command(*query*, *params=None*)

Create a request object for the given parameters.

set_server(*url: str*)

Set the update server URL.

class kasa.modules.**Countdown**(*device:* SmartDevice, *module: str*)

Implementation of countdown module.

call(*method*, *params=None*)

Call the given method with the given parameters.

property data

Return the module specific raw data from the last update.

async delete_all_rules()

Delete all rules.

async delete_rule(*rule: kasa.modules.rulemodule.Rule*)

Delete the given rule.

property estimated_query_response_size

Estimated maximum size of query response.

The inheriting modules implement this to estimate how large a query response will be so that queries can be split should an estimated response be too large

property is_supported: bool

Return whether the module is supported by the device.

query()

Prepare the query for rules.

query_for_command(*query*, *params=None*)

Create a request object for the given parameters.

property rules: List[kasa.modules.rulemodule.Rule]

Return the list of rules for the service.

async set_enabled(*state: bool*)

Enable or disable the service.

class kasa.modules.**Emeter**(*device:* SmartDevice, *module: str*)

Emeter module.

call(*method*, *params=None*)

Call the given method with the given parameters.

property daily_data

Return statistics on daily basis.

property data

Return the module specific raw data from the last update.

property emeter_this_month: Optional[float]

Return this month's energy consumption in kWh.

property emeter_today: Optional[float]

Return today's energy consumption in kWh.

async erase_stats()

Erase all stats.

Uses different query than usage meter.

property estimated_query_response_size

Estimated maximum query response size.

async get_daystat(**, year=None, month=None, kwh=True*) → Dict

Return daily stats for the given year & month.

The return value is a dictionary of {day: energy, ... }.

async get_monthstat(**, year=None, kwh=True*) → Dict

Return monthly stats for the given year.

The return value is a dictionary of {month: energy, ... }.

async get_raw_daystat(**, year=None, month=None*) → Dict

Return raw daily stats for the given year & month.

async get_raw_monthstat(**, year=None*) → Dict

Return raw monthly stats for the given year.

async get_realtime()

Return real-time statistics.

property is_supported: bool

Return whether the module is supported by the device.

property monthly_data

Return statistics on monthly basis.

query()

Return the base query.

query_for_command(*query*, *params=None*)

Create a request object for the given parameters.

property realtime: kasa.emeterstatus.EmeterStatus

Return current energy readings.

property usage_this_month

Return usage in this month in minutes.

property usage_today

Return today's usage in minutes.

class kasa.modules.**Module**(*device:* SmartDevice, *module: str*)

Base class implemention for all modules.

The base classes should implement *query* to return the query they want to be executed during the regular update cycle.

call(*method*, *params=None*)

Call the given method with the given parameters.

property data

Return the module specific raw data from the last update.

property estimated_query_response_size

Estimated maximum size of query response.

The inheriting modules implement this to estimate how large a query response will be so that queries can be split should an estimated response be too large

property is_supported: bool

Return whether the module is supported by the device.

abstract query()

Query to execute during the update cycle.

The inheriting modules implement this to include their wanted queries to the query that gets executed when Device.update() gets called.

query_for_command(*query*, *params=None*)

Create a request object for the given parameters.

class kasa.modules.**Motion**(*device:* SmartDevice, *module: str*)

Implements the motion detection (PIR) module.

call(*method*, *params=None*)

Call the given method with the given parameters.

property data

Return the module specific raw data from the last update.

property enabled: bool

Return True if module is enabled.

property estimated_query_response_size

Estimated maximum size of query response.

The inheriting modules implement this to estimate how large a query response will be so that queries can be split should an estimated response be too large

property inactivity_timeout: int

Return inactivity timeout in milliseconds.

property is_supported: bool

Return whether the module is supported by the device.

query()

Request PIR configuration.

query_for_command(*query*, *params=None*)

Create a request object for the given parameters.

property range: kasa.modules.motion.Range

Return motion detection range.

async set_enabled(*state: bool*)

Enable/disable PIR.

async set_inactivity_timeout(*timeout: int*)

Set inactivity timeout in milliseconds.

Note, that you need to delete the default "Smart Control" rule in the app to avoid reverting this back to 60 seconds after a period of time.

async set_range(**, range: Optional[kasa.modules.motion.Range] = None, custom_range: Optional[int] = None*)

Set the range for the sensor.

Parameters

- **range** – for using standard ranges
- **custom_range** – range in decimeters, overrides the range parameter

class kasa.modules.**Rule**(**, id: str, name: str, enable: bool, wday: List[int], repeat: bool, sact: Optional[kasa.modules.rulemodule.Action] = None, stime_opt: kasa.modules.rulemodule.TimeOption, smin: int, eact: Optional[kasa.modules.rulemodule.Action] = None, etime_opt: kasa.modules.rulemodule.TimeOption, emin: int, s_light: Optional[Dict] = None*)

Representation of a rule.

Config

alias of pydantic.v1.config.BaseConfig

classmethod construct(*_fields_set: Optional[SetStr] = None*, ***values: Any*) → Model

Creates a new model setting __dict__ and __fields_set__ from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

copy(**, include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None, exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None, update: Optional[DictStrAny] = None, deep: bool = False*) → Model

Duplicate a model, optionally choose which fields to include, exclude and change.

Parameters

- **include** – fields to include in new model
- **exclude** – fields to exclude from new model, as with values this takes precedence over include
- **update** – values to change/add in the new model. Note: the data is not validated before creating the new model: you should trust this data
- **deep** – set to *True* to make a deep copy of the model

Returns new model instance

dict(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → DictStrAny

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

eact: Optional[kasa.modules.rulemodule.Action]

emin: int

enable: bool

etime_opt: kasa.modules.rulemodule.TimeOption

classmethod from_orm(*obj: Any*) → Model

id: str

json(*, *include: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *exclude: Optional[Union[AbstractSetIntStr, MappingIntStrAny]] = None*, *by_alias: bool = False*, *skip_defaults: Optional[bool] = None*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Optional[Callable[[Any], Any]] = None*, *models_as_dict: bool = True*, ***dumps_kwargs: Any*) → str

Generate a JSON representation of the model, *include* and *exclude* arguments as per *dict()*.

encoder is an optional function to supply as *default* to json.dumps(), other arguments as per *json.dumps()*.

name: str

classmethod parse_file(*path: Union[str, pathlib.Path]*, *, *content_type: str = None*, *encoding: str = 'utf8'*, *proto: pydantic.v1.parse.Protocol = None*, *allow_pickle: bool = False*) → Model

classmethod parse_obj(*obj: Any*) → Model

classmethod parse_raw(*b: Union[str, bytes]*, *, *content_type: str = None*, *encoding: str = 'utf8'*, *proto: pydantic.v1.parse.Protocol = None*, *allow_pickle: bool = False*) → Model

repeat: bool

s_light: Optional[Dict]

sact: Optional[kasa.modules.rulemodule.Action]

classmethod schema(*by_alias: bool = True*, *ref_template: str = '#/definitions/{model}'*) → DictStrAny

classmethod schema_json(*, *by_alias: bool = True*, *ref_template: str = '#/definitions/{model}'*, ***dumps_kwargs: Any*) → str

smin: int

stime_opt: kasa.modules.rulemodule.TimeOption

classmethod update_forward_refs(***localns: Any*) → None

Try to update ForwardRefs on fields based on this Model, globalns and localns.

classmethod validate(*value: Any*) → Model

wday: List[int]

class kasa.modules.**RuleModule**(*device:* SmartDevice, *module: str*)

Base class for rule-based modules, such as countdown and antitheft.

call(*method*, *params=None*)

Call the given method with the given parameters.

property data

Return the module specific raw data from the last update.

async delete_all_rules()

Delete all rules.

async delete_rule(*rule: kasa.modules.rulemodule.Rule*)

Delete the given rule.

property estimated_query_response_size

Estimated maximum size of query response.

The inheriting modules implement this to estimate how large a query response will be so that queries can be split should an estimated response be too large

property is_supported: bool

Return whether the module is supported by the device.

query()

Prepare the query for rules.

query_for_command(*query*, *params=None*)

Create a request object for the given parameters.

property rules: List[kasa.modules.rulemodule.Rule]

Return the list of rules for the service.

async set_enabled(*state: bool*)

Enable or disable the service.

class kasa.modules.**Schedule**(*device:* SmartDevice, *module: str*)

Implements the scheduling interface.

call(*method*, *params=None*)

Call the given method with the given parameters.

property data

Return the module specific raw data from the last update.

async delete_all_rules()

Delete all rules.

async delete_rule(*rule: kasa.modules.rulemodule.Rule*)

Delete the given rule.

property estimated_query_response_size

Estimated maximum size of query response.

The inheriting modules implement this to estimate how large a query response will be so that queries can be split should an estimated response be too large

property is_supported: bool

Return whether the module is supported by the device.

query()

Prepare the query for rules.

query_for_command(*query*, *params=None*)

Create a request object for the given parameters.

property rules: List[kasa.modules.rulemodule.Rule]

Return the list of rules for the service.

async set_enabled(*state: bool*)

Enable or disable the service.

class kasa.modules.**Time**(*device:* SmartDevice, *module: str*)

Implements the timezone settings.

call(*method*, *params=None*)

Call the given method with the given parameters.

property data

Return the module specific raw data from the last update.

property estimated_query_response_size

Estimated maximum size of query response.

The inheriting modules implement this to estimate how large a query response will be so that queries can be split should an estimated response be too large

async get_time()

Return current device time.

async get_timezone()

Request timezone information from the device.

property is_supported: bool

Return whether the module is supported by the device.

query()

Request time and timezone.

query_for_command(*query*, *params=None*)

Create a request object for the given parameters.

property time: datetime.datetime

Return current device time.

property timezone

Return current timezone.

class kasa.modules.**Usage**(*device:* SmartDevice, *module: str*)

Baseclass for emeter/usage interfaces.

call(*method*, *params=None*)

Call the given method with the given parameters.

property daily_data

Return statistics on daily basis.

property data

Return the module specific raw data from the last update.

async erase_stats()

Erase all stats.

property estimated_query_response_size

Estimated maximum query response size.

async get_daystat(*, *year=None*, *month=None*) → Dict

Return daily stats for the given year & month.

The return value is a dictionary of {day: time, ...}.

async get_monthstat(*, *year=None*) → Dict

Return monthly stats for the given year.

The return value is a dictionary of {month: time, ...}.

async get_raw_daystat(*, *year=None*, *month=None*) → Dict

Return raw daily stats for the given year & month.

async get_raw_monthstat(*, *year=None*) → Dict

Return raw monthly stats for the given year.

property is_supported: bool

Return whether the module is supported by the device.

property monthly_data

Return statistics on monthly basis.

query()

Return the base query.

query_for_command(*query*, *params=None*)

Create a request object for the given parameters.

property usage_this_month

Return usage in this month in minutes.

property usage_today

Return today's usage in minutes.

CHAPTER

TEN

BULBS

Contents

- *Supported features*
- *Currently unsupported*
- *Transitions*
- *Command-line usage*
- *API documentation*

10.1 Supported features

- Turning on and off
- Setting brightness, color temperature, and color (in HSV)
- Querying emeter information
- Transitions
- Presets

10.2 Currently unsupported

- Setting the default transitions
- Timers

Note: Feel free to open a pull request to add support for more features!

10.3 Transitions

All commands changing the bulb state can be accompanied with a transition, e.g., to slowly fade the light off. The transition time is in milliseconds, 0 means immediate change. If no transition value is given, the default setting as configured for the bulb will be used.

Note: Accepted values are command (and potentially bulb) specific, feel free to improve the documentation on accepted values.

Example: While KL130 allows at least up to 15 second transitions for smooth turning off transitions, turning it on will not be so smooth.

10.4 Command-line usage

All command-line commands can be used with transition period for smooth changes.

Example: Turn the bulb off over a 15 second time period.

```
$ kasa --type bulb --host <host> off --transition 15000
```

Example: Change the bulb to red with 20% brightness over 15 seconds:

```
$ kasa --type bulb --host <host> hsv 0 100 20 --transition 15000
```

10.5 API documentation

class kasa.**SmartBulb**(*host: str*, ***, *port: Optional[int] = None*, *credentials: Optional[kasa.credentials.Credentials] = None*, *timeout: Optional[int] = None*)

Representation of a TP-Link Smart Bulb.

To initialize, you have to await *update()* at least once. This will allow accessing the properties using the exposed properties.

All changes to the device are done using awaitable methods, which will not change the cached values, so you must await *update()* to fetch updates values from the device.

Errors reported by the device are raised as `SmartDeviceExceptions`, and should be handled by the user of the library.

Examples:

```
>>> import asyncio
>>> bulb = SmartBulb("127.0.0.1")
>>> asyncio.run(bulb.update())
>>> print(bulb.alias)
Bulb2
```

Bulbs, like any other supported devices, can be turned on and off:

```
>>> asyncio.run(bulb.turn_off())
>>> asyncio.run(bulb.turn_on())
>>> asyncio.run(bulb.update())
>>> print(bulb.is_on)
True
```

You can use the `is_`-prefixed properties to check for supported features:

```
>>> bulb.is_dimmable
True
>>> bulb.is_color
True
>>> bulb.is_variable_color_temp
True
```

All known bulbs support changing the brightness:

```
>>> bulb.brightness
30
>>> asyncio.run(bulb.set_brightness(50))
>>> asyncio.run(bulb.update())
>>> bulb.brightness
50
```

Bulbs supporting color temperature can be queried for the supported range:

```
>>> bulb.valid_temperature_range
ColorTempRange(min=2500, max=9000)
>>> asyncio.run(bulb.set_color_temp(3000))
>>> asyncio.run(bulb.update())
>>> bulb.color_temp
3000
```

Color bulbs can be adjusted by passing hue, saturation and value:

```
>>> asyncio.run(bulb.set_hsv(180, 100, 80))
>>> asyncio.run(bulb.update())
>>> bulb.hsv
HSV(hue=180, saturation=100, value=80)
```

If you don't want to use the default transitions, you can pass *transition* in milliseconds. All methods changing the state of the device support this parameter:

- *turn_on()*
- *turn_off()*
- *set_hsv()*
- *set_color_temp()*
- *set_brightness()*

Light strips (e.g., KL420L5) do not support this feature, but silently ignore the parameter. The following changes the brightness over a period of 10 seconds:

```
>>> asyncio.run(bulb.set_brightness(100, transition=10_000))
```

Bulb configuration presets can be accessed using the *presets()* property:

```
>>> bulb.presets
[SmartBulbPreset(index=0, brightness=50, hue=0, saturation=0, color_temp=2700,␣
↪custom=None, id=None, mode=None), SmartBulbPreset(index=1, brightness=100,␣
↪hue=0, saturation=75, color_temp=0, custom=None, id=None, mode=None),␣
↪SmartBulbPreset(index=2, brightness=100, hue=120, saturation=75, color_temp=0,
↪ custom=None, id=None, mode=None), SmartBulbPreset(index=3, brightness=100,␣
↪hue=240, saturation=75, color_temp=0, custom=None, id=None, mode=None)]
```

To modify an existing preset, pass *SmartBulbPreset* instance to *save_preset()* method:

```
>>> preset = bulb.presets[0]
>>> preset.brightness
50
>>> preset.brightness = 100
>>> asyncio.run(bulb.save_preset(preset))
>>> bulb.presets[0].brightness
100
```

LIGHT_SERVICE = 'smartlife.iot.smartbulb.lightingservice'

SET_LIGHT_METHOD = 'transition_light_state'

add_module(*name: str*, *module: kasa.modules.module.Module*)

Register a module.

property alias: str

Return device name (alias).

property brightness: int

Return the current brightness in percentage.

property color_temp: int

Return color temperature of the device in kelvin.

async static connect(*host: str*, ***, *port: Optional[int] = None*, *timeout=5*, *credentials: Optional[kasa.credentials.Credentials] = None*, *device_type: Optional[kasa.device_type.DeviceType] = None*) → *kasa.smartdevice.SmartDevice*

Connect to a single device by the given IP address.

This method avoids the UDP based discovery process and will connect directly to the device to query its type.

It is generally preferred to avoid `discover_single()` and use this function instead as it should perform better when the WiFi network is congested or the device is not responding to discovery requests.

The device type is discovered by querying the device.

Parameters

- **host** – Hostname of device to query
- **device_type** – Device type to use for the device. If not given, the device type is discovered by querying the device. If the device type is already known, it is preferred to pass it to avoid the extra query to the device to discover its type.

Return type *SmartDevice*

Returns Object for querying/controlling found device.

async current_consumption() → float

Get the current power consumption in Watt.

property device_id: str

Return unique ID for the device.

If not overridden, this is the MAC address of the device. Individual sockets on strips will override this.

property device_type: kasa.device_type.DeviceType

Return the device type.

property emeter_realtime: kasa.emeterstatus.EmeterStatus

Return current energy readings.

property emeter_this_month: Optional[float]

Return this month's energy consumption in kWh.

property emeter_today: Optional[float]

Return today's energy consumption in kWh.

emeter_type = 'smartlife.iot.common.emeter'

async erase_emeter_stats() → Dict

Erase energy meter statistics.

property features: Set[str]

Return a set of features that the device supports.

async get_emeter_daily(*year: Optional[int] = None*, *month: Optional[int] = None*, *kwh: bool = True*) → Dict

Retrieve daily statistics for a given month.

Parameters

- **year** – year for which to retrieve statistics (default: this year)
- **month** – month for which to retrieve statistics (default: this month)
- **kwh** – return usage in kWh (default: True)

Returns mapping of day of month to value

async get_emeter_monthly(*year: Optional[int] = None*, *kwh: bool = True*) → Dict

Retrieve monthly statistics for a given year.

Parameters

- **year** – year for which to retrieve statistics (default: this year)
- **kwh** – return usage in kWh (default: True)

Returns dict: mapping of month to value

async get_emeter_realtime() → kasa.emeterstatus.EmeterStatus

Retrieve current energy readings.

async get_light_details() → Dict[str, int]

Return light details.

Example:

```
{'lamp_beam_angle': 290, 'min_voltage': 220, 'max_voltage': 240,
 'wattage': 5, 'incandescent_equivalent': 40, 'max_lumens': 450,
  'color_rendering_index': 80}
```

async get_light_state() → Dict[str, Dict]

Query the light state.

get_plug_by_index(*index: int*) → *kasa.smartdevice.SmartDevice*

Return child device for the given index.

get_plug_by_name(*name: str*) → *kasa.smartdevice.SmartDevice*

Return child device for the given name.

async get_sys_info() → Dict[str, Any]

Retrieve system information.

async get_time() → Optional[datetime.datetime]

Return current time from the device, if available.

async get_timezone() → Dict

Return timezone information.

async get_turn_on_behavior() → *kasa.smartbulb.TurnOnBehaviors*

Return the behavior for turning the bulb on.

property has_children: bool

Return true if the device has children devices.

property has_effects: bool

Return True if the device supports effects.

property has_emeter: bool

Return that the bulb has an emeter.

property hsv: kasa.smartbulb.HSV

Return the current HSV state of the bulb.

Returns hue, saturation and value (degrees, %, %)

property hw_info: Dict

Return hardware information.

This returns just a selection of sysinfo keys that are related to hardware.

property internal_state: Any

Return the internal state of the instance.

The returned object contains the raw results from the last update call. This should only be used for debugging purposes.

property is_bulb: bool

Return True if the device is a bulb.

property is_color: bool

Whether the bulb supports color changes.

property is_dimmable: bool

Whether the bulb supports brightness changes.

property is_dimmer: bool

Return True if the device is a dimmer.

property is_light_strip: bool

Return True if the device is a led strip.

property is_off: bool

Return True if device is off.

property is_on: bool

Return whether the device is on.

property is_plug: bool

Return True if the device is a plug.

property is_strip: bool

Return True if the device is a strip.

property is_strip_socket: bool

Return True if the device is a strip socket.

property is_variable_color_temp: bool

Whether the bulb supports color temperature changes.

property light_state: Dict[str, str]

Query the light state.

property location: Dict

Return geographical location.

property mac: str

Return mac address.

Returns mac address in hexadecimal with colons, e.g. 01:23:45:67:89:ab

property max_device_response_size: int

Returns the maximum response size the device can safely construct.

property model: str

Return device model.

property on_since: Optional[datetime.datetime]

Return pretty-printed on-time, or None if not available.

property presets: List[*kasa.smartbulb.SmartBulbPreset*]

Return a list of available bulb setting presets.

async reboot(*delay: int = 1*) → None

Reboot the device.

Note that giving a delay of zero causes this to block, as the device reboots immediately without responding to the call.

property rssi: Optional[int]

Return WiFi signal strength (rssi).

async save_preset(*preset:* kasa.smartbulb.SmartBulbPreset)

Save a setting preset.

You can either construct a preset object manually, or pass an existing one obtained using *presets()*.

async set_alias(*alias: str*) → None

Set the device name (alias).

Overridden to use a different module name.

async set_brightness(*brightness: int*, *, *transition: Optional[int] = None*) → Dict

Set the brightness in percentage.

Parameters

- **brightness** (*int*) – brightness in percent
- **transition** (*int*) – transition in milliseconds.

async set_color_temp(*temp: int*, *, *brightness=None*, *transition: Optional[int] = None*) → Dict

Set the color temperature of the device in kelvin.

Parameters

- **temp** (*int*) – The new color temperature, in Kelvin
- **transition** (*int*) – transition in milliseconds.

async set_hsv(*hue: int*, *saturation: int*, *value: Optional[int] = None*, *, *transition: Optional[int] = None*) → Dict

Set new HSV.

Parameters

- **hue** (*int*) – hue in degrees
- **saturation** (*int*) – saturation in percentage [0,100]
- **value** (*int*) – value in percentage [0, 100]
- **transition** (*int*) – transition in milliseconds.

async set_light_state(*state: Dict*, *, *transition: Optional[int] = None*) → Dict

Set the light state.

async set_mac(*mac*)

Set the mac address.

Parameters **mac** (*str*) – mac in hexadecimal with colons, e.g. 01:23:45:67:89:ab

async set_turn_on_behavior(*behavior:* kasa.smartbulb.TurnOnBehaviors)

Set the behavior for turning the bulb on.

If you do not want to manually construct the behavior object, you should use *get_turn_on_behavior()* to get the current settings.

property state_information: Dict[str, Any]

Return bulb-specific state information.

property supported_modules: List[str]

Return a set of modules supported by the device.

property sys_info: Dict[str, Any]

Return system information.

Do not call this function from within the SmartDevice class itself as @requires_update will be affected for other properties.

property time: datetime.datetime

Return current time from the device.

property timezone: Dict

Return the current timezone.

async turn_off(*, *transition: Optional[int] = None*, ***kwargs*) → Dict

Turn the bulb off.

Parameters **transition** (*int*) – transition in milliseconds.

async turn_on(*, *transition: Optional[int] = None*, ***kwargs*) → Dict

Turn the bulb on.

Parameters **transition** (*int*) – transition in milliseconds.

async update(*update_children: bool = True*)

Query the device to update the data.

Needed for properties that are decorated with *requires_update*.

update_from_discover_info(*info: Dict[str, Any]*) → None

Update state from info from the discover call.

property valid_temperature_range: kasa.smartbulb.ColorTempRange

Return the device-specific white temperature range (in Kelvin).

Returns White temperature range in Kelvin (minimum, maximum)

async wifi_join(*ssid*, *password*, *keytype=3*)

Join the given wifi network.

If joining the network fails, the device will return to AP mode after a while.

async wifi_scan() → List[kasa.smartdevice.WifiNetwork]

Scan for available wifi networks.

class kasa.SmartBulbPreset(*, *index: int*, *brightness: int*, *hue: Optional[int] = None*, *saturation: Optional[int] = None*, *color_temp: Optional[int] = None*, *custom: Optional[int] = None*, *id: Optional[str] = None*, *mode: Optional[int] = None*)

Bulb configuration preset.

brightness: int

color_temp: Optional[int]

custom: Optional[int]

hue: Optional[int]

id: Optional[str]

index: `int`

mode: `Optional[int]`

saturation: `Optional[int]`

class kasa.**TurnOnBehaviors**(*, *soft_on:* kasa.smartbulb.TurnOnBehavior, *hard_on:* kasa.smartbulb.TurnOnBehavior)

Model to contain turn on behaviors.

hard: *kasa.smartbulb.TurnOnBehavior*

The behavior when the bulb has been off from mains power.

soft: *kasa.smartbulb.TurnOnBehavior*

The behavior when the bulb is turned on programmatically.

class kasa.**TurnOnBehavior**(*, *index: Optional[int] = None*, *mode: kasa.smartbulb.BehaviorMode*)

Model to present a single turn on behavior.

Parameters

- **preset** (*int*) – the index number of wanted preset.
- **mode** (*BehaviorMode*) – last status or preset mode. If you are changing existing settings, you should not set this manually.

To change the behavior, it is only necessary to change the *preset* field to contain either the preset index, or `None` for the last known state.

class Config

Configuration to make the validator run when changing the values.

validate_assignment = True

mode: `kasa.smartbulb.BehaviorMode`

Wanted behavior

preset: `Optional[int]`

Index of preset to use, or `None` for the last known state.

CHAPTER

ELEVEN

PLUGS

Contents

- *API documentation*

Note: Feel free to open a pull request to improve the documentation!

11.1 API documentation

class kasa.**SmartPlug**(*host: str*, ***, *port: Optional[int] = None*, *credentials: Optional[kasa.credentials.Credentials] = None*, *timeout: Optional[int] = None*)

Representation of a TP-Link Smart Switch.

To initialize, you have to await *update()* at least once. This will allow accessing the properties using the exposed properties.

All changes to the device are done using awaitable methods, which will not change the cached values, but you must await *update()* separately.

Errors reported by the device are raised as SmartDeviceExceptions, and should be handled by the user of the library.

Examples:

```
>>> import asyncio
>>> plug = SmartPlug("127.0.0.1")
>>> asyncio.run(plug.update())
>>> plug.alias
Kitchen
```

Setting the LED state:

```
>>> asyncio.run(plug.set_led(True))
>>> asyncio.run(plug.update())
>>> plug.led
True
```

For more examples, see the *SmartDevice* class.

add_module(*name: str*, *module: kasa.modules.module.Module*)

Register a module.

property alias: str

Return device name (alias).

children: List['SmartDevice']

async static connect(*host: str*, ***, *port: Optional[int] = None*, *timeout=5*, *credentials: Optional[kasa.credentials.Credentials] = None*, *device_type: Optional[kasa.device_type.DeviceType] = None*) → *kasa.smartdevice.SmartDevice*

Connect to a single device by the given IP address.

This method avoids the UDP based discovery process and will connect directly to the device to query its type.

It is generally preferred to avoid `discover_single()` and use this function instead as it should perform better when the WiFi network is congested or the device is not responding to discovery requests.

The device type is discovered by querying the device.

Parameters

- **host** – Hostname of device to query
- **device_type** – Device type to use for the device. If not given, the device type is discovered by querying the device. If the device type is already known, it is preferred to pass it to avoid the extra query to the device to discover its type.

Return type *SmartDevice*

Returns Object for querying/controlling found device.

async current_consumption() → float

Get the current power consumption in Watt.

property device_id: str

Return unique ID for the device.

If not overridden, this is the MAC address of the device. Individual sockets on strips will override this.

property device_type: kasa.device_type.DeviceType

Return the device type.

property emeter_realtime: kasa.emeterstatus.EmeterStatus

Return current energy readings.

property emeter_this_month: Optional[float]

Return this month's energy consumption in kWh.

property emeter_today: Optional[float]

Return today's energy consumption in kWh.

emeter_type = 'emeter'

async erase_emeter_stats() → Dict

Erase energy meter statistics.

property features: Set[str]

Return a set of features that the device supports.

async get_emeter_daily(*year: Optional[int] = None*, *month: Optional[int] = None*, *kwh: bool = True*) → Dict

Retrieve daily statistics for a given month.

Parameters

- **year** – year for which to retrieve statistics (default: this year)
- **month** – month for which to retrieve statistics (default: this month)
- **kwh** – return usage in kWh (default: True)

Returns mapping of day of month to value

async get_emeter_monthly(*year: Optional[int] = None*, *kwh: bool = True*) → Dict

Retrieve monthly statistics for a given year.

Parameters

- **year** – year for which to retrieve statistics (default: this year)
- **kwh** – return usage in kWh (default: True)

Returns dict: mapping of month to value

async get_emeter_realtime() → kasa.emeterstatus.EmeterStatus

Retrieve current energy readings.

get_plug_by_index(*index: int*) → *kasa.smartdevice.SmartDevice*

Return child device for the given index.

get_plug_by_name(*name: str*) → *kasa.smartdevice.SmartDevice*

Return child device for the given name.

async get_sys_info() → Dict[str, Any]

Retrieve system information.

async get_time() → Optional[datetime.datetime]

Return current time from the device, if available.

async get_timezone() → Dict

Return timezone information.

property has_children: bool

Return true if the device has children devices.

property has_emeter: bool

Return True if device has an energy meter.

property hw_info: Dict

Return hardware information.

This returns just a selection of sysinfo keys that are related to hardware.

property internal_state: Any

Return the internal state of the instance.

The returned object contains the raw results from the last update call. This should only be used for debugging purposes.

property is_bulb: bool

Return True if the device is a bulb.

property is_color: bool

Return True if the device supports color changes.

property is_dimmable: bool

Return True if the device is dimmable.

property is_dimmer: bool

Return True if the device is a dimmer.

property is_light_strip: bool

Return True if the device is a led strip.

property is_off: bool

Return True if device is off.

property is_on: bool

Return whether device is on.

property is_plug: bool

Return True if the device is a plug.

property is_strip: bool

Return True if the device is a strip.

property is_strip_socket: bool

Return True if the device is a strip socket.

property is_variable_color_temp: bool

Return True if the device supports color temperature.

property led: bool

Return the state of the led.

property location: Dict

Return geographical location.

property mac: str

Return mac address.

Returns mac address in hexadecimal with colons, e.g. 01:23:45:67:89:ab

property max_device_response_size: int

Returns the maximum response size the device can safely construct.

property model: str

Return device model.

modules: Dict[str, Any]

property on_since: Optional[datetime.datetime]

Return pretty-printed on-time, or None if not available.

protocol: TPLinkProtocol

async reboot(*delay: int = 1*) → None

Reboot the device.

Note that giving a delay of zero causes this to block, as the device reboots immediately without responding to the call.

property rssi: Optional[int]

Return WiFi signal strength (rssi).

async set_alias(*alias: str*) → None

Set the device name (alias).

async set_led(*state: bool*)

Set the state of the led (night mode).

async set_mac(*mac*)

Set the mac address.

Parameters **mac** (*str*) – mac in hexadecimal with colons, e.g. 01:23:45:67:89:ab

property state_information: Dict[str, Any]

Return switch-specific state information.

property supported_modules: List[str]

Return a set of modules supported by the device.

property sys_info: Dict[str, Any]

Return system information.

Do not call this function from within the SmartDevice class itself as @requires_update will be affected for other properties.

property time: datetime.datetime

Return current time from the device.

property timezone: Dict

Return the current timezone.

async turn_off(***kwargs*)

Turn the switch off.

async turn_on(***kwargs*)

Turn the switch on.

async update(*update_children: bool = True*)

Query the device to update the data.

Needed for properties that are decorated with *requires_update*.

update_from_discover_info(*info: Dict[str, Any]*) → None

Update state from info from the discover call.

async wifi_join(*ssid*, *password*, *keytype=3*)

Join the given wifi network.

If joining the network fails, the device will return to AP mode after a while.

async wifi_scan() → List[kasa.smartdevice.WifiNetwork]

Scan for available wifi networks.

CHAPTER

TWELVE

DIMMERS

Contents

- *API documentation*

Note: Feel free to open a pull request to improve the documentation!

12.1 API documentation

class kasa.**SmartDimmer**(*host: str*, ***, *port: Optional[int] = None*, *credentials: Optional[kasa.credentials.Credentials] = None*, *timeout: Optional[int] = None*)

Representation of a TP-Link Smart Dimmer.

Dimmers work similarly to plugs, but provide also support for adjusting the brightness. This class extends *SmartPlug* interface.

To initialize, you have to await *update()* at least once. This will allow accessing the properties using the exposed properties.

All changes to the device are done using awaitable methods, which will not change the cached values, but you must await *update()* separately.

Errors reported by the device are raised as SmartDeviceExceptions, and should be handled by the user of the library.

Examples: >>> import asyncio >>> dimmer = SmartDimmer("192.168.1.105") >>> asyncio.run(dimmer.turn_on()) >>> dimmer.brightness 25

```
>>> asyncio.run(dimmer.set_brightness(50))
>>> asyncio.run(dimmer.update())
>>> dimmer.brightness
50
```

Refer to *SmartPlug* for the full API.

DIMMER_SERVICE = 'smartlife.iot.dimmer'

add_module(*name: str*, *module: kasa.modules.module.Module*)

Register a module.

property alias: str

Return device name (alias).

property brightness: int

Return current brightness on dimmers.

Will return a range between 0 - 100.

children: List['SmartDevice']

async static connect(*host: str*, ***, *port: Optional[int] = None*, *timeout=5*, *credentials: Optional[kasa.credentials.Credentials] = None*, *device_type: Optional[kasa.device_type.DeviceType] = None*) → *kasa.smartdevice.SmartDevice*

Connect to a single device by the given IP address.

This method avoids the UDP based discovery process and will connect directly to the device to query its type.

It is generally preferred to avoid `discover_single()` and use this function instead as it should perform better when the WiFi network is congested or the device is not responding to discovery requests.

The device type is discovered by querying the device.

Parameters

- **host** – Hostname of device to query
- **device_type** – Device type to use for the device. If not given, the device type is discovered by querying the device. If the device type is already known, it is preferred to pass it to avoid the extra query to the device to discover its type.

Return type *SmartDevice*

Returns Object for querying/controlling found device.

async current_consumption() → float

Get the current power consumption in Watt.

property device_id: str

Return unique ID for the device.

If not overridden, this is the MAC address of the device. Individual sockets on strips will override this.

property device_type: kasa.device_type.DeviceType

Return the device type.

property emeter_realtime: kasa.emeterstatus.EmeterStatus

Return current energy readings.

property emeter_this_month: Optional[float]

Return this month's energy consumption in kWh.

property emeter_today: Optional[float]

Return today's energy consumption in kWh.

emeter_type = 'emeter'

async erase_emeter_stats() → Dict

Erase energy meter statistics.

property features: Set[str]

Return a set of features that the device supports.

async get_behaviors()

Return button behavior settings.

async get_emeter_daily(*year: Optional[int] = None*, *month: Optional[int] = None*, *kwh: bool = True*) → Dict

Retrieve daily statistics for a given month.

Parameters

- **year** – year for which to retrieve statistics (default: this year)
- **month** – month for which to retrieve statistics (default: this month)
- **kwh** – return usage in kWh (default: True)

Returns mapping of day of month to value

async get_emeter_monthly(*year: Optional[int] = None*, *kwh: bool = True*) → Dict

Retrieve monthly statistics for a given year.

Parameters

- **year** – year for which to retrieve statistics (default: this year)
- **kwh** – return usage in kWh (default: True)

Returns dict: mapping of month to value

async get_emeter_realtime() → kasa.emeterstatus.EmeterStatus

Retrieve current energy readings.

get_plug_by_index(*index: int*) → *kasa.smartdevice.SmartDevice*

Return child device for the given index.

get_plug_by_name(*name: str*) → *kasa.smartdevice.SmartDevice*

Return child device for the given name.

async get_sys_info() → Dict[str, Any]

Retrieve system information.

async get_time() → Optional[datetime.datetime]

Return current time from the device, if available.

async get_timezone() → Dict

Return timezone information.

property has_children: bool

Return true if the device has children devices.

property has_emeter: bool

Return True if device has an energy meter.

property hw_info: Dict

Return hardware information.

This returns just a selection of sysinfo keys that are related to hardware.

property internal_state: Any

Return the internal state of the instance.

The returned object contains the raw results from the last update call. This should only be used for debugging purposes.

property is_bulb: bool

Return True if the device is a bulb.

property is_color: bool

Return True if the device supports color changes.

property is_dimmable: bool

Whether the switch supports brightness changes.

property is_dimmer: bool

Return True if the device is a dimmer.

property is_light_strip: bool

Return True if the device is a led strip.

property is_off: bool

Return True if device is off.

property is_on: bool

Return whether device is on.

property is_plug: bool

Return True if the device is a plug.

property is_strip: bool

Return True if the device is a strip.

property is_strip_socket: bool

Return True if the device is a strip socket.

property is_variable_color_temp: bool

Return True if the device supports color temperature.

property led: bool

Return the state of the led.

property location: Dict

Return geographical location.

property mac: str

Return mac address.

Returns mac address in hexadecimal with colons, e.g. 01:23:45:67:89:ab

property max_device_response_size: int

Returns the maximum response size the device can safely construct.

property model: str

Return device model.

modules: Dict[str, Any]

property on_since: Optional[datetime.datetime]

Return pretty-printed on-time, or None if not available.

protocol: TPLinkProtocol

async reboot(*delay: int = 1*) → None

Reboot the device.

Note that giving a delay of zero causes this to block, as the device reboots immediately without responding to the call.

property rssi: Optional[int]

Return WiFi signal strength (rssi).

async set_alias(*alias: str*) → None

Set the device name (alias).

async set_brightness(*brightness: int*, *, *transition: Optional[int] = None*)

Set the new dimmer brightness level in percentage.

Parameters **transition** (*int*) – transition duration in milliseconds. Using a transition will cause the dimmer to turn on.

async set_button_action(*action_type: kasa.smartdimmer.ActionType*, *action: kasa.smartdimmer.ButtonAction*, *index: Optional[int] = None*)

Set action to perform on button click/hold.

Parameters

- **ActionType** (*action_type*) – whether to control double click or hold action.
- **ButtonAction** (*action*) – what should the button do (nothing, instant, gentle, change preset)
- **int** (*index*) – in case of preset change, the preset to select

async set_dimmer_transition(*brightness: int*, *transition: int*)

Turn the bulb on to brightness percentage over transition milliseconds.

A brightness value of 0 will turn off the dimmer.

async set_fade_time(*fade_type: kasa.smartdimmer.FadeType*, *time: int*)

Set time for fade in / fade out.

async set_led(*state: bool*)

Set the state of the led (night mode).

async set_mac(*mac*)

Set the mac address.

Parameters **mac** (*str*) – mac in hexadecimal with colons, e.g. 01:23:45:67:89:ab

property state_information: Dict[str, Any]

Return switch-specific state information.

property supported_modules: List[str]

Return a set of modules supported by the device.

property sys_info: Dict[str, Any]

Return system information.

Do not call this function from within the SmartDevice class itself as @requires_update will be affected for other properties.

property time: datetime.datetime

Return current time from the device.

property timezone: Dict

Return the current timezone.

async turn_off(*, *transition: Optional[int] = None*, ***kwargs*)

Turn the bulb off.

Parameters **transition** (*int*) – transition duration in milliseconds.

async turn_on(*, *transition: Optional[int] = None*, ***kwargs*)

Turn the bulb on.

Parameters **transition** (*int*) – transition duration in milliseconds.

async update(*update_children: bool = True*)

Query the device to update the data.

Needed for properties that are decorated with *requires_update*.

update_from_discover_info(*info: Dict[str, Any]*) → None

Update state from info from the discover call.

async wifi_join(*ssid*, *password*, *keytype=3*)

Join the given wifi network.

If joining the network fails, the device will return to AP mode after a while.

async wifi_scan() → List[kasa.smartdevice.WifiNetwork]

Scan for available wifi networks.

CHAPTER

THIRTEEN

SMART STRIPS

Contents

- *Command-line usage*
- *API documentation*

Note: Feel free to open a pull request to improve the documentation!

13.1 Command-line usage

To command a single socket of a strip, you will need to specify it either by using `--index` or by using `--name`. If not specified, the commands will act on the parent device: turning the strip off will turn off all sockets.

Example: Turn on the first socket (the indexing starts from zero):

```
$ kasa --type strip --host <host> on --index 0
```

Example: Turn off the socket by name:

```
$ kasa --type strip --host <host> off --name "Maybe Kitchen"
```

13.2 API documentation

class kasa.**SmartStrip**(*host: str*, ***, *port: Optional[int] = None*, *credentials: Optional[kasa.credentials.Credentials] = None*, *timeout: Optional[int] = None*)

Representation of a TP-Link Smart Power Strip.

A strip consists of the parent device and its children. All methods of the parent act on all children, while the child devices share the common API with the *SmartPlug* class.

To initialize, you have to await *update()* at least once. This will allow accessing the properties using the exposed properties.

All changes to the device are done using awaitable methods, which will not change the cached values, but you must await *update()* separately.

Errors reported by the device are raised as `SmartDeviceExceptions`, and should be handled by the user of the library.

Examples:

```
>>> import asyncio
>>> strip = SmartStrip("127.0.0.1")
>>> asyncio.run(strip.update())
>>> strip.alias
TP-LINK_Power Strip_CF69
```

All methods act on the whole strip:

```
>>> for plug in strip.children:
>>>     print(f"{plug.alias}: {plug.is_on}")
Plug 1: True
Plug 2: False
Plug 3: False
>>> strip.is_on
True
>>> asyncio.run(strip.turn_off())
```

Accessing individual plugs can be done using the *children* property:

```
>>> len(strip.children)
3
>>> for plug in strip.children:
>>>     print(f"{plug.alias}: {plug.is_on}")
Plug 1: False
Plug 2: False
Plug 3: False
>>> asyncio.run(strip.children[1].turn_on())
>>> asyncio.run(strip.update())
>>> strip.is_on
True
```

For more examples, see the *SmartDevice* class.

add_module(*name: str*, *module: kasa.modules.module.Module*)

Register a module.

property alias: str

Return device name (alias).

children: List['SmartDevice']

async static connect(*host: str*, ***, *port: Optional[int] = None*, *timeout=5*, *credentials: Optional[kasa.credentials.Credentials] = None*, *device_type: Optional[kasa.device_type.DeviceType] = None*) → *kasa.smartdevice.SmartDevice*

Connect to a single device by the given IP address.

This method avoids the UDP based discovery process and will connect directly to the device to query its type.

It is generally preferred to avoid `discover_single()` and use this function instead as it should perform better when the WiFi network is congested or the device is not responding to discovery requests.

The device type is discovered by querying the device.

Parameters

- **host** – Hostname of device to query
- **device_type** – Device type to use for the device. If not given, the device type is discovered by querying the device. If the device type is already known, it is preferred to pass it to avoid the extra query to the device to discover its type.

Return type *SmartDevice*

Returns Object for querying/controlling found device.

async current_consumption() → float

Get the current power consumption in watts.

property device_id: str

Return unique ID for the device.

If not overridden, this is the MAC address of the device. Individual sockets on strips will override this.

property device_type: kasa.device_type.DeviceType

Return the device type.

property emeter_realtime: kasa.emeterstatus.EmeterStatus

Return current energy readings.

property emeter_this_month: Optional[float]

Return this month's energy consumption in kWh.

property emeter_today: Optional[float]

Return this month's energy consumption in kWh.

emeter_type = 'emeter'

async erase_emeter_stats()

Erase energy meter statistics for all plugs.

property features: Set[str]

Return a set of features that the device supports.

async get_emeter_daily(*year: Optional[int] = None*, *month: Optional[int] = None*, *kwh: bool = True*) → Dict

Retrieve daily statistics for a given month.

Parameters

- **year** – year for which to retrieve statistics (default: this year)
- **month** – month for which to retrieve statistics (default: this month)
- **kwh** – return usage in kWh (default: True)

Returns mapping of day of month to value

async get_emeter_monthly(*year: Optional[int] = None*, *kwh: bool = True*) → Dict

Retrieve monthly statistics for a given year.

Parameters

- **year** – year for which to retrieve statistics (default: this year)
- **kwh** – return usage in kWh (default: True)

async get_emeter_realtime() → kasa.emeterstatus.EmeterStatus

Retrieve current energy readings.

get_plug_by_index(*index: int*) → *kasa.smartdevice.SmartDevice*

Return child device for the given index.

get_plug_by_name(*name: str*) → *kasa.smartdevice.SmartDevice*

Return child device for the given name.

async get_sys_info() → Dict[str, Any]

Retrieve system information.

async get_time() → Optional[datetime.datetime]

Return current time from the device, if available.

async get_timezone() → Dict

Return timezone information.

property has_children: bool

Return true if the device has children devices.

property has_emeter: bool

Return True if device has an energy meter.

property hw_info: Dict

Return hardware information.

This returns just a selection of sysinfo keys that are related to hardware.

property internal_state: Any

Return the internal state of the instance.

The returned object contains the raw results from the last update call. This should only be used for debugging purposes.

property is_bulb: bool

Return True if the device is a bulb.

property is_color: bool

Return True if the device supports color changes.

property is_dimmable: bool

Return True if the device is dimmable.

property is_dimmer: bool

Return True if the device is a dimmer.

property is_light_strip: bool

Return True if the device is a led strip.

property is_off: bool

Return True if device is off.

property is_on: bool

Return if any of the outlets are on.

property is_plug: bool

Return True if the device is a plug.

property is_strip: bool

Return True if the device is a strip.

property is_strip_socket: bool

Return True if the device is a strip socket.

property is_variable_color_temp: bool

Return True if the device supports color temperature.

property led: bool

Return the state of the led.

property location: Dict

Return geographical location.

property mac: str

Return mac address.

Returns mac address in hexadecimal with colons, e.g. 01:23:45:67:89:ab

property max_device_response_size: int

Returns the maximum response size the device can safely construct.

property model: str

Return device model.

modules: Dict[str, Any]

property on_since: Optional[datetime.datetime]

Return the maximum on-time of all outlets.

protocol: TPLinkProtocol

async reboot(*delay: int = 1*) → None

Reboot the device.

Note that giving a delay of zero causes this to block, as the device reboots immediately without responding to the call.

property rssi: Optional[int]

Return WiFi signal strength (rssi).

async set_alias(*alias: str*) → None

Set the device name (alias).

async set_led(*state: bool*)

Set the state of the led (night mode).

async set_mac(*mac*)

Set the mac address.

Parameters **mac** (*str*) – mac in hexadecimal with colons, e.g. 01:23:45:67:89:ab

property state_information: Dict[str, Any]

Return strip-specific state information.

Returns Strip information dict, keys in user-presentable form.

property supported_modules: List[str]

Return a set of modules supported by the device.

property sys_info: Dict[str, Any]

Return system information.

Do not call this function from within the SmartDevice class itself as @requires_update will be affected for other properties.

property time: datetime.datetime

Return current time from the device.

property timezone: Dict

Return the current timezone.

async turn_off(***kwargs*)

Turn the strip off.

async turn_on(***kwargs*)

Turn the strip on.

async update(*update_children: bool = True*)

Update some of the attributes.

Needed for methods that are decorated with *requires_update*.

update_from_discover_info(*info: Dict[str, Any]*) → None

Update state from info from the discover call.

async wifi_join(*ssid*, *password*, *keytype=3*)

Join the given wifi network.

If joining the network fails, the device will return to AP mode after a while.

async wifi_scan() → List[kasa.smartdevice.WifiNetwork]

Scan for available wifi networks.

CHAPTER

FOURTEEN

LIGHT STRIPS

Contents

- *API documentation*

Note: Feel free to open a pull request to improve the documentation!

14.1 API documentation

class kasa.**SmartLightStrip**(*host: str*, ***, *port: Optional[int] = None*, *credentials: Optional[kasa.credentials.Credentials] = None*, *timeout: Optional[int] = None*)

Representation of a TP-Link Smart light strip.

Light strips work similarly to bulbs, but use a different service for controlling, and expose some extra information (such as length and active effect). This class extends *SmartBulb* interface.

Examples:

```
>>> import asyncio
>>> strip = SmartLightStrip("127.0.0.1")
>>> asyncio.run(strip.update())
>>> print(strip.alias)
KL430 pantry lightstrip
```

Getting the length of the strip:

```
>>> strip.length
16
```

Currently active effect:

```
>>> strip.effect
{'brightness': 50, 'custom': 0, 'enable': 0, 'id': '', 'name': ''}
```

Note: The device supports some features that are not currently implemented, feel free to find out how to control them and create a PR!

See *SmartBulb* for more examples.

LIGHT_SERVICE = 'smartlife.iot.lightStrip'

SET_LIGHT_METHOD = 'set_light_state'

add_module(*name: str*, *module: kasa.modules.module.Module*)

Register a module.

property alias: str

Return device name (alias).

property brightness: int

Return the current brightness in percentage.

children: List['SmartDevice']

property color_temp: int

Return color temperature of the device in kelvin.

async static connect(*host: str*, ***, *port: Optional[int] = None*, *timeout=5*, *credentials: Optional[kasa.credentials.Credentials] = None*, *device_type: Optional[kasa.device_type.DeviceType] = None*) → *kasa.smartdevice.SmartDevice*

Connect to a single device by the given IP address.

This method avoids the UDP based discovery process and will connect directly to the device to query its type.

It is generally preferred to avoid `discover_single()` and use this function instead as it should perform better when the WiFi network is congested or the device is not responding to discovery requests.

The device type is discovered by querying the device.

Parameters

- **host** – Hostname of device to query
- **device_type** – Device type to use for the device. If not given, the device type is discovered by querying the device. If the device type is already known, it is preferred to pass it to avoid the extra query to the device to discover its type.

Return type *SmartDevice*

Returns Object for querying/controlling found device.

async current_consumption() → float

Get the current power consumption in Watt.

property device_id: str

Return unique ID for the device.

If not overridden, this is the MAC address of the device. Individual sockets on strips will override this.

property device_type: kasa.device_type.DeviceType

Return the device type.

property effect: Dict

Return effect state.

Example:

{'brightness': 50, 'custom': 0, 'enable': 0, 'id': '', 'name': ''}

property effect_list: Optional[List[str]]

Return built-in effects list.

Example: ['Aurora', 'Bubbling Cauldron', ...]

property emeter_realtime: kasa.emeterstatus.EmeterStatus

Return current energy readings.

property emeter_this_month: Optional[float]

Return this month's energy consumption in kWh.

property emeter_today: Optional[float]

Return today's energy consumption in kWh.

emeter_type = 'smartlife.iot.common.emeter'

async erase_emeter_stats() → Dict

Erase energy meter statistics.

property features: Set[str]

Return a set of features that the device supports.

async get_emeter_daily(*year: Optional[int] = None*, *month: Optional[int] = None*, *kwh: bool = True*) → Dict

Retrieve daily statistics for a given month.

Parameters

- **year** – year for which to retrieve statistics (default: this year)
- **month** – month for which to retrieve statistics (default: this month)
- **kwh** – return usage in kWh (default: True)

Returns mapping of day of month to value

async get_emeter_monthly(*year: Optional[int] = None*, *kwh: bool = True*) → Dict

Retrieve monthly statistics for a given year.

Parameters

- **year** – year for which to retrieve statistics (default: this year)
- **kwh** – return usage in kWh (default: True)

Returns dict: mapping of month to value

async get_emeter_realtime() → kasa.emeterstatus.EmeterStatus

Retrieve current energy readings.

async get_light_details() → Dict[str, int]

Return light details.

Example:

```
{'lamp_beam_angle': 290, 'min_voltage': 220, 'max_voltage': 240,
 'wattage': 5, 'incandescent_equivalent': 40, 'max_lumens': 450,
  'color_rendering_index': 80}
```

async get_light_state() → Dict[str, Dict]

Query the light state.

get_plug_by_index(*index: int*) → *kasa.smartdevice.SmartDevice*

Return child device for the given index.

get_plug_by_name(*name: str*) → *kasa.smartdevice.SmartDevice*

Return child device for the given name.

async get_sys_info() → Dict[str, Any]

Retrieve system information.

async get_time() → Optional[datetime.datetime]

Return current time from the device, if available.

async get_timezone() → Dict

Return timezone information.

async get_turn_on_behavior() → *kasa.smartbulb.TurnOnBehaviors*

Return the behavior for turning the bulb on.

property has_children: bool

Return true if the device has children devices.

property has_effects: bool

Return True if the device supports effects.

property has_emeter: bool

Return that the bulb has an emeter.

property hsv: kasa.smartbulb.HSV

Return the current HSV state of the bulb.

Returns hue, saturation and value (degrees, %, %)

property hw_info: Dict

Return hardware information.

This returns just a selection of sysinfo keys that are related to hardware.

property internal_state: Any

Return the internal state of the instance.

The returned object contains the raw results from the last update call. This should only be used for debugging purposes.

property is_bulb: bool

Return True if the device is a bulb.

property is_color: bool

Whether the bulb supports color changes.

property is_dimmable: bool

Whether the bulb supports brightness changes.

property is_dimmer: bool

Return True if the device is a dimmer.

property is_light_strip: bool

Return True if the device is a led strip.

property is_off: bool

Return True if device is off.

property is_on: bool

Return whether the device is on.

property is_plug: bool

Return True if the device is a plug.

property is_strip: bool

Return True if the device is a strip.

property is_strip_socket: bool

Return True if the device is a strip socket.

property is_variable_color_temp: bool

Whether the bulb supports color temperature changes.

property length: int

Return length of the strip.

property light_state: Dict[str, str]

Query the light state.

property location: Dict

Return geographical location.

property mac: str

Return mac address.

Returns mac address in hexadecimal with colons, e.g. 01:23:45:67:89:ab

property max_device_response_size: int

Returns the maximum response size the device can safely construct.

property model: str

Return device model.

modules: Dict[str, Any]

property on_since: Optional[datetime.datetime]

Return pretty-printed on-time, or None if not available.

property presets: List[*kasa.smartbulb.SmartBulbPreset*]

Return a list of available bulb setting presets.

protocol: TPLinkProtocol

async reboot(*delay: int = 1*) → None

Reboot the device.

Note that giving a delay of zero causes this to block, as the device reboots immediately without responding to the call.

property rssi: Optional[int]

Return WiFi signal strength (rssi).

async save_preset(*preset:* kasa.smartbulb.SmartBulbPreset)

Save a setting preset.

You can either construct a preset object manually, or pass an existing one obtained using *presets()*.

async set_alias(*alias: str*) → None

Set the device name (alias).

Overridden to use a different module name.

async set_brightness(*brightness: int*, *, *transition: Optional[int] = None*) → Dict

Set the brightness in percentage.

Parameters

- **brightness** (*int*) – brightness in percent
- **transition** (*int*) – transition in milliseconds.

async set_color_temp(*temp: int*, *, *brightness=None*, *transition: Optional[int] = None*) → Dict

Set the color temperature of the device in kelvin.

Parameters

- **temp** (*int*) – The new color temperature, in Kelvin
- **transition** (*int*) – transition in milliseconds.

async set_custom_effect(*effect_dict: Dict*) → None

Set a custom effect on the device.

Parameters **effect_dict** (*str*) – The custom effect dict to set

async set_effect(*effect: str*, *, *brightness: Optional[int] = None*, *transition: Optional[int] = None*) → None

Set an effect on the device.

If brightness or transition is defined, its value will be used instead of the effect-specific default.

See *effect_list()* for available effects, or use *set_custom_effect()* for custom effects.

Parameters

- **effect** (*str*) – The effect to set
- **brightness** (*int*) – The wanted brightness
- **transition** (*int*) – The wanted transition time

async set_hsv(*hue: int*, *saturation: int*, *value: Optional[int] = None*, *, *transition: Optional[int] = None*) → Dict

Set new HSV.

Parameters

- **hue** (*int*) – hue in degrees
- **saturation** (*int*) – saturation in percentage [0,100]
- **value** (*int*) – value in percentage [0, 100]
- **transition** (*int*) – transition in milliseconds.

async set_light_state(*state: Dict*, *, *transition: Optional[int] = None*) → Dict

Set the light state.

async set_mac(*mac*)

Set the mac address.

Parameters **mac** (*str*) – mac in hexadecimal with colons, e.g. 01:23:45:67:89:ab

async set_turn_on_behavior(*behavior:* kasa.smartbulb.TurnOnBehaviors)

Set the behavior for turning the bulb on.

If you do not want to manually construct the behavior object, you should use *get_turn_on_behavior()* to get the current settings.

property state_information: Dict[str, Any]

Return strip specific state information.

property supported_modules: List[str]

Return a set of modules supported by the device.

property sys_info: Dict[str, Any]

Return system information.

Do not call this function from within the SmartDevice class itself as @requires_update will be affected for other properties.

property time: datetime.datetime

Return current time from the device.

property timezone: Dict

Return the current timezone.

async turn_off(*, *transition: Optional[int] = None*, ***kwargs*) → Dict

Turn the bulb off.

Parameters **transition** (*int*) – transition in milliseconds.

async turn_on(*, *transition: Optional[int] = None*, ***kwargs*) → Dict

Turn the bulb on.

Parameters **transition** (*int*) – transition in milliseconds.

async update(*update_children: bool = True*)

Query the device to update the data.

Needed for properties that are decorated with *requires_update*.

update_from_discover_info(*info: Dict[str, Any]*) → None

Update state from info from the discover call.

property valid_temperature_range: kasa.smartbulb.ColorTempRange

Return the device-specific white temperature range (in Kelvin).

Returns White temperature range in Kelvin (minimum, maximum)

async wifi_join(*ssid*, *password*, *keytype=3*)

Join the given wifi network.

If joining the network fails, the device will return to AP mode after a while.

async wifi_scan() → List[kasa.smartdevice.WifiNetwork]

Scan for available wifi networks.

PYTHON MODULE INDEX

k

kasa, 18
kasa.modules, 26

INDEX

A

add_module() (*kasa.SmartBulb method*), 42
add_module() (*kasa.SmartDevice method*), 22
add_module() (*kasa.SmartDimmer method*), 55
add_module() (*kasa.SmartLightStrip method*), 68
add_module() (*kasa.SmartPlug method*), 49
add_module() (*kasa.SmartStrip method*), 62
alias (*kasa.SmartBulb property*), 42
alias (*kasa.SmartDevice property*), 22
alias (*kasa.SmartDimmer property*), 55
alias (*kasa.SmartLightStrip property*), 68
alias (*kasa.SmartPlug property*), 50
alias (*kasa.SmartStrip property*), 62

B

brightness (*kasa.SmartBulb property*), 42
brightness (*kasa.SmartBulbPreset attribute*), 47
brightness (*kasa.SmartDimmer property*), 56
brightness (*kasa.SmartLightStrip property*), 68

C

children (*kasa.SmartDimmer attribute*), 56
children (*kasa.SmartLightStrip attribute*), 68
children (*kasa.SmartPlug attribute*), 50
children (*kasa.SmartStrip attribute*), 62
color_temp (*kasa.SmartBulb property*), 42
color_temp (*kasa.SmartBulbPreset attribute*), 47
color_temp (*kasa.SmartLightStrip property*), 68
connect() (*kasa.SmartBulb static method*), 42
connect() (*kasa.SmartDevice static method*), 22
connect() (*kasa.SmartDimmer static method*), 56
connect() (*kasa.SmartLightStrip static method*), 68
connect() (*kasa.SmartPlug static method*), 50
connect() (*kasa.SmartStrip static method*), 62
current_consumption() (*kasa.SmartBulb method*), 43
current_consumption() (*kasa.SmartDevice method*), 23
current_consumption() (*kasa.SmartDimmer method*), 56
current_consumption() (*kasa.SmartLightStrip method*), 68
current_consumption() (*kasa.SmartPlug method*), 50
current_consumption() (*kasa.SmartStrip method*), 63
custom (*kasa.SmartBulbPreset attribute*), 47

D

device_id (*kasa.SmartBulb property*), 43
device_id (*kasa.SmartDevice property*), 23
device_id (*kasa.SmartDimmer property*), 56
device_id (*kasa.SmartLightStrip property*), 68
device_id (*kasa.SmartPlug property*), 50
device_id (*kasa.SmartStrip property*), 63
device_type (*kasa.SmartBulb property*), 43
device_type (*kasa.SmartDevice property*), 23
device_type (*kasa.SmartDimmer property*), 56
device_type (*kasa.SmartLightStrip property*), 68
device_type (*kasa.SmartPlug property*), 50
device_type (*kasa.SmartStrip property*), 63
DIMMER_SERVICE (*kasa.SmartDimmer attribute*), 55
Discover (*class in kasa*), 17
discover() (*kasa.Discover static method*), 18
discover_single() (*kasa.Discover static method*), 18
DISCOVERY_PORT (*kasa.Discover attribute*), 17
DISCOVERY_PORT_2 (*kasa.Discover attribute*), 17
DISCOVERY_QUERY (*kasa.Discover attribute*), 17
DISCOVERY_QUERY_2 (*kasa.Discover attribute*), 17

E

effect (*kasa.SmartLightStrip property*), 68
effect_list (*kasa.SmartLightStrip property*), 69
emeter_realtime (*kasa.SmartBulb property*), 43
emeter_realtime (*kasa.SmartDevice property*), 23
emeter_realtime (*kasa.SmartDimmer property*), 56
emeter_realtime (*kasa.SmartLightStrip property*), 69
emeter_realtime (*kasa.SmartPlug property*), 50
emeter_realtime (*kasa.SmartStrip property*), 63
emeter_this_month (*kasa.SmartBulb property*), 43
emeter_this_month (*kasa.SmartDevice property*), 23
emeter_this_month (*kasa.SmartDimmer property*), 56
emeter_this_month (*kasa.SmartLightStrip property*), 69
emeter_this_month (*kasa.SmartPlug property*), 50
emeter_this_month (*kasa.SmartStrip property*), 63
emeter_today (*kasa.SmartBulb property*), 43

emeter_today (*kasa.SmartDevice property*), 23
emeter_today (*kasa.SmartDimmer property*), 56
emeter_today (*kasa.SmartLightStrip property*), 69
emeter_today (*kasa.SmartPlug property*), 50
emeter_today (*kasa.SmartStrip property*), 63
emeter_type (*kasa.SmartBulb attribute*), 43
emeter_type (*kasa.SmartDevice attribute*), 23
emeter_type (*kasa.SmartDimmer attribute*), 56
emeter_type (*kasa.SmartLightStrip attribute*), 69
emeter_type (*kasa.SmartPlug attribute*), 50
emeter_type (*kasa.SmartStrip attribute*), 63
erase_emeter_stats() (*kasa.SmartBulb method*), 43
erase_emeter_stats() (*kasa.SmartDevice method*), 23
erase_emeter_stats() (*kasa.SmartDimmer method*), 56
erase_emeter_stats() (*kasa.SmartLightStrip method*), 69
erase_emeter_stats() (*kasa.SmartPlug method*), 50
erase_emeter_stats() (*kasa.SmartStrip method*), 63

F

features (*kasa.SmartBulb property*), 43
features (*kasa.SmartDevice property*), 23
features (*kasa.SmartDimmer property*), 56
features (*kasa.SmartLightStrip property*), 69
features (*kasa.SmartPlug property*), 50
features (*kasa.SmartStrip property*), 63

G

get_behaviors() (*kasa.SmartDimmer method*), 57
get_emeter_daily() (*kasa.SmartBulb method*), 43
get_emeter_daily() (*kasa.SmartDevice method*), 23
get_emeter_daily() (*kasa.SmartDimmer method*), 57
get_emeter_daily() (*kasa.SmartLightStrip method*), 69
get_emeter_daily() (*kasa.SmartPlug method*), 50
get_emeter_daily() (*kasa.SmartStrip method*), 63
get_emeter_monthly() (*kasa.SmartBulb method*), 43
get_emeter_monthly() (*kasa.SmartDevice method*), 23
get_emeter_monthly() (*kasa.SmartDimmer method*), 57
get_emeter_monthly() (*kasa.SmartLightStrip method*), 69
get_emeter_monthly() (*kasa.SmartPlug method*), 51
get_emeter_monthly() (*kasa.SmartStrip method*), 63
get_emeter_realtime() (*kasa.SmartBulb method*), 43
get_emeter_realtime() (*kasa.SmartDevice method*), 24
get_emeter_realtime() (*kasa.SmartDimmer method*), 57
get_emeter_realtime() (*kasa.SmartLightStrip method*), 69
get_emeter_realtime() (*kasa.SmartPlug method*), 51
get_emeter_realtime() (*kasa.SmartStrip method*), 63
get_light_details() (*kasa.SmartBulb method*), 43
get_light_details() (*kasa.SmartLightStrip method*), 69
get_light_state() (*kasa.SmartBulb method*), 44
get_light_state() (*kasa.SmartLightStrip method*), 69
get_plug_by_index() (*kasa.SmartBulb method*), 44
get_plug_by_index() (*kasa.SmartDevice method*), 24
get_plug_by_index() (*kasa.SmartDimmer method*), 57
get_plug_by_index() (*kasa.SmartLightStrip method*), 69
get_plug_by_index() (*kasa.SmartPlug method*), 51
get_plug_by_index() (*kasa.SmartStrip method*), 64
get_plug_by_name() (*kasa.SmartBulb method*), 44
get_plug_by_name() (*kasa.SmartDevice method*), 24
get_plug_by_name() (*kasa.SmartDimmer method*), 57
get_plug_by_name() (*kasa.SmartLightStrip method*), 70
get_plug_by_name() (*kasa.SmartPlug method*), 51
get_plug_by_name() (*kasa.SmartStrip method*), 64
get_sys_info() (*kasa.SmartBulb method*), 44
get_sys_info() (*kasa.SmartDevice method*), 24
get_sys_info() (*kasa.SmartDimmer method*), 57
get_sys_info() (*kasa.SmartLightStrip method*), 70
get_sys_info() (*kasa.SmartPlug method*), 51
get_sys_info() (*kasa.SmartStrip method*), 64
get_time() (*kasa.SmartBulb method*), 44
get_time() (*kasa.SmartDevice method*), 24
get_time() (*kasa.SmartDimmer method*), 57
get_time() (*kasa.SmartLightStrip method*), 70
get_time() (*kasa.SmartPlug method*), 51
get_time() (*kasa.SmartStrip method*), 64
get_timezone() (*kasa.SmartBulb method*), 44
get_timezone() (*kasa.SmartDevice method*), 24
get_timezone() (*kasa.SmartDimmer method*), 57
get_timezone() (*kasa.SmartLightStrip method*), 70
get_timezone() (*kasa.SmartPlug method*), 51
get_timezone() (*kasa.SmartStrip method*), 64
get_turn_on_behavior() (*kasa.SmartBulb method*), 44
get_turn_on_behavior() (*kasa.SmartLightStrip method*), 70

H

hard (*kasa.TurnOnBehaviors attribute*), 48
has_children (*kasa.SmartBulb property*), 44
has_children (*kasa.SmartDevice property*), 24
has_children (*kasa.SmartDimmer property*), 57
has_children (*kasa.SmartLightStrip property*), 70
has_children (*kasa.SmartPlug property*), 51
has_children (*kasa.SmartStrip property*), 64
has_effects (*kasa.SmartBulb property*), 44

has_effects (*kasa.SmartLightStrip property*), 70
has_emeter (*kasa.SmartBulb property*), 44
has_emeter (*kasa.SmartDevice property*), 24
has_emeter (*kasa.SmartDimmer property*), 57
has_emeter (*kasa.SmartLightStrip property*), 70
has_emeter (*kasa.SmartPlug property*), 51
has_emeter (*kasa.SmartStrip property*), 64
hsv (*kasa.SmartBulb property*), 44
hsv (*kasa.SmartLightStrip property*), 70
hue (*kasa.SmartBulbPreset attribute*), 47
hw_info (*kasa.SmartBulb property*), 44
hw_info (*kasa.SmartDevice property*), 24
hw_info (*kasa.SmartDimmer property*), 57
hw_info (*kasa.SmartLightStrip property*), 70
hw_info (*kasa.SmartPlug property*), 51
hw_info (*kasa.SmartStrip property*), 64

I

id (*kasa.SmartBulbPreset attribute*), 47
index (*kasa.SmartBulbPreset attribute*), 47
internal_state (*kasa.SmartBulb property*), 44
internal_state (*kasa.SmartDevice property*), 24
internal_state (*kasa.SmartDimmer property*), 57
internal_state (*kasa.SmartLightStrip property*), 70
internal_state (*kasa.SmartPlug property*), 51
internal_state (*kasa.SmartStrip property*), 64
is_bulb (*kasa.SmartBulb property*), 44
is_bulb (*kasa.SmartDevice property*), 24
is_bulb (*kasa.SmartDimmer property*), 58
is_bulb (*kasa.SmartLightStrip property*), 70
is_bulb (*kasa.SmartPlug property*), 51
is_bulb (*kasa.SmartStrip property*), 64
is_color (*kasa.SmartBulb property*), 44
is_color (*kasa.SmartDevice property*), 24
is_color (*kasa.SmartDimmer property*), 58
is_color (*kasa.SmartLightStrip property*), 70
is_color (*kasa.SmartPlug property*), 51
is_color (*kasa.SmartStrip property*), 64
is_dimmable (*kasa.SmartBulb property*), 45
is_dimmable (*kasa.SmartDevice property*), 24
is_dimmable (*kasa.SmartDimmer property*), 58
is_dimmable (*kasa.SmartLightStrip property*), 70
is_dimmable (*kasa.SmartPlug property*), 52
is_dimmable (*kasa.SmartStrip property*), 64
is_dimmer (*kasa.SmartBulb property*), 45
is_dimmer (*kasa.SmartDevice property*), 24
is_dimmer (*kasa.SmartDimmer property*), 58
is_dimmer (*kasa.SmartLightStrip property*), 70
is_dimmer (*kasa.SmartPlug property*), 52
is_dimmer (*kasa.SmartStrip property*), 64
is_light_strip (*kasa.SmartBulb property*), 45
is_light_strip (*kasa.SmartDevice property*), 24
is_light_strip (*kasa.SmartDimmer property*), 58
is_light_strip (*kasa.SmartLightStrip property*), 70
is_light_strip (*kasa.SmartPlug property*), 52
is_light_strip (*kasa.SmartStrip property*), 64
is_off (*kasa.SmartBulb property*), 45
is_off (*kasa.SmartDevice property*), 24
is_off (*kasa.SmartDimmer property*), 58
is_off (*kasa.SmartLightStrip property*), 70
is_off (*kasa.SmartPlug property*), 52
is_off (*kasa.SmartStrip property*), 64
is_on (*kasa.SmartBulb property*), 45
is_on (*kasa.SmartDevice property*), 25
is_on (*kasa.SmartDimmer property*), 58
is_on (*kasa.SmartLightStrip property*), 70
is_on (*kasa.SmartPlug property*), 52
is_on (*kasa.SmartStrip property*), 64
is_plug (*kasa.SmartBulb property*), 45
is_plug (*kasa.SmartDevice property*), 25
is_plug (*kasa.SmartDimmer property*), 58
is_plug (*kasa.SmartLightStrip property*), 71
is_plug (*kasa.SmartPlug property*), 52
is_plug (*kasa.SmartStrip property*), 64
is_strip (*kasa.SmartBulb property*), 45
is_strip (*kasa.SmartDevice property*), 25
is_strip (*kasa.SmartDimmer property*), 58
is_strip (*kasa.SmartLightStrip property*), 71
is_strip (*kasa.SmartPlug property*), 52
is_strip (*kasa.SmartStrip property*), 64
is_strip_socket (*kasa.SmartBulb property*), 45
is_strip_socket (*kasa.SmartDevice property*), 25
is_strip_socket (*kasa.SmartDimmer property*), 58
is_strip_socket (*kasa.SmartLightStrip property*), 71
is_strip_socket (*kasa.SmartPlug property*), 52
is_strip_socket (*kasa.SmartStrip property*), 65
is_variable_color_temp (*kasa.SmartBulb property*), 45
is_variable_color_temp (*kasa.SmartDevice property*), 25
is_variable_color_temp (*kasa.SmartDimmer property*), 58
is_variable_color_temp (*kasa.SmartLightStrip property*), 71
is_variable_color_temp (*kasa.SmartPlug property*), 52
is_variable_color_temp (*kasa.SmartStrip property*), 65

K

kasa
    module, 18
kasa.modules
    module, 26

L

led (*kasa.SmartDimmer property*), 58
led (*kasa.SmartPlug property*), 52

led (*kasa.SmartStrip property*), 65
length (*kasa.SmartLightStrip property*), 71
LIGHT_SERVICE (*kasa.SmartBulb attribute*), 42
LIGHT_SERVICE (*kasa.SmartLightStrip attribute*), 68
light_state (*kasa.SmartBulb property*), 45
light_state (*kasa.SmartLightStrip property*), 71
location (*kasa.SmartBulb property*), 45
location (*kasa.SmartDevice property*), 25
location (*kasa.SmartDimmer property*), 58
location (*kasa.SmartLightStrip property*), 71
location (*kasa.SmartPlug property*), 52
location (*kasa.SmartStrip property*), 65

M

mac (*kasa.SmartBulb property*), 45
mac (*kasa.SmartDevice property*), 25
mac (*kasa.SmartDimmer property*), 58
mac (*kasa.SmartLightStrip property*), 71
mac (*kasa.SmartPlug property*), 52
mac (*kasa.SmartStrip property*), 65
max_device_response_size (*kasa.SmartBulb property*), 45
max_device_response_size (*kasa.SmartDevice property*), 25
max_device_response_size (*kasa.SmartDimmer property*), 58
max_device_response_size (*kasa.SmartLightStrip property*), 71
max_device_response_size (*kasa.SmartPlug property*), 52
max_device_response_size (*kasa.SmartStrip property*), 65
mode (*kasa.SmartBulbPreset attribute*), 48
mode (*kasa.TurnOnBehavior attribute*), 48
model (*kasa.SmartBulb property*), 45
model (*kasa.SmartDevice property*), 25
model (*kasa.SmartDimmer property*), 58
model (*kasa.SmartLightStrip property*), 71
model (*kasa.SmartPlug property*), 52
model (*kasa.SmartStrip property*), 65
module
    kasa, 18
    kasa.modules, 26
modules (*kasa.SmartDimmer attribute*), 58
modules (*kasa.SmartLightStrip attribute*), 71
modules (*kasa.SmartPlug attribute*), 52
modules (*kasa.SmartStrip attribute*), 65

O

on_since (*kasa.SmartBulb property*), 45
on_since (*kasa.SmartDevice property*), 25
on_since (*kasa.SmartDimmer property*), 58
on_since (*kasa.SmartLightStrip property*), 71
on_since (*kasa.SmartPlug property*), 52
on_since (*kasa.SmartStrip property*), 65

P

preset (*kasa.TurnOnBehavior attribute*), 48
presets (*kasa.SmartBulb property*), 45
presets (*kasa.SmartLightStrip property*), 71
protocol (*kasa.SmartDimmer attribute*), 59
protocol (*kasa.SmartLightStrip attribute*), 71
protocol (*kasa.SmartPlug attribute*), 52
protocol (*kasa.SmartStrip attribute*), 65

R

reboot() (*kasa.SmartBulb method*), 45
reboot() (*kasa.SmartDevice method*), 25
reboot() (*kasa.SmartDimmer method*), 59
reboot() (*kasa.SmartLightStrip method*), 71
reboot() (*kasa.SmartPlug method*), 52
reboot() (*kasa.SmartStrip method*), 65
rssi (*kasa.SmartBulb property*), 45
rssi (*kasa.SmartDevice property*), 25
rssi (*kasa.SmartDimmer property*), 59
rssi (*kasa.SmartLightStrip property*), 71
rssi (*kasa.SmartPlug property*), 52
rssi (*kasa.SmartStrip property*), 65

S

saturation (*kasa.SmartBulbPreset attribute*), 48
save_preset() (*kasa.SmartBulb method*), 46
save_preset() (*kasa.SmartLightStrip method*), 71
set_alias() (*kasa.SmartBulb method*), 46
set_alias() (*kasa.SmartDevice method*), 25
set_alias() (*kasa.SmartDimmer method*), 59
set_alias() (*kasa.SmartLightStrip method*), 71
set_alias() (*kasa.SmartPlug method*), 53
set_alias() (*kasa.SmartStrip method*), 65
set_brightness() (*kasa.SmartBulb method*), 46
set_brightness() (*kasa.SmartDimmer method*), 59
set_brightness() (*kasa.SmartLightStrip method*), 72
set_button_action() (*kasa.SmartDimmer method*), 59
set_color_temp() (*kasa.SmartBulb method*), 46
set_color_temp() (*kasa.SmartLightStrip method*), 72
set_custom_effect() (*kasa.SmartLightStrip method*), 72
set_dimmer_transition() (*kasa.SmartDimmer method*), 59
set_effect() (*kasa.SmartLightStrip method*), 72
set_fade_time() (*kasa.SmartDimmer method*), 59
set_hsv() (*kasa.SmartBulb method*), 46
set_hsv() (*kasa.SmartLightStrip method*), 72
set_led() (*kasa.SmartDimmer method*), 59
set_led() (*kasa.SmartPlug method*), 53
set_led() (*kasa.SmartStrip method*), 65

SET_LIGHT_METHOD (*kasa.SmartBulb attribute*), 42
SET_LIGHT_METHOD (*kasa.SmartLightStrip attribute*), 68
set_light_state() (*kasa.SmartBulb method*), 46
set_light_state() (*kasa.SmartLightStrip method*), 72
set_mac() (*kasa.SmartBulb method*), 46
set_mac() (*kasa.SmartDevice method*), 25
set_mac() (*kasa.SmartDimmer method*), 59
set_mac() (*kasa.SmartLightStrip method*), 72
set_mac() (*kasa.SmartPlug method*), 53
set_mac() (*kasa.SmartStrip method*), 65
set_turn_on_behavior() (*kasa.SmartBulb method*), 46
set_turn_on_behavior() (*kasa.SmartLightStrip method*), 72
SmartBulb (*class in kasa*), 40
SmartBulbPreset (*class in kasa*), 47
SmartDevice (*class in kasa*), 21
SmartDimmer (*class in kasa*), 55
SmartLightStrip (*class in kasa*), 67
SmartPlug (*class in kasa*), 49
SmartStrip (*class in kasa*), 61
soft (*kasa.TurnOnBehaviors attribute*), 48
state_information (*kasa.SmartBulb property*), 46
state_information (*kasa.SmartDevice property*), 25
state_information (*kasa.SmartDimmer property*), 59
state_information (*kasa.SmartLightStrip property*), 73
state_information (*kasa.SmartPlug property*), 53
state_information (*kasa.SmartStrip property*), 65
supported_modules (*kasa.SmartBulb property*), 46
supported_modules (*kasa.SmartDevice property*), 25
supported_modules (*kasa.SmartDimmer property*), 59
supported_modules (*kasa.SmartLightStrip property*), 73
supported_modules (*kasa.SmartPlug property*), 53
supported_modules (*kasa.SmartStrip property*), 65
sys_info (*kasa.SmartBulb property*), 47
sys_info (*kasa.SmartDevice property*), 25
sys_info (*kasa.SmartDimmer property*), 59
sys_info (*kasa.SmartLightStrip property*), 73
sys_info (*kasa.SmartPlug property*), 53
sys_info (*kasa.SmartStrip property*), 65

T

time (*kasa.SmartBulb property*), 47
time (*kasa.SmartDevice property*), 26
time (*kasa.SmartDimmer property*), 60
time (*kasa.SmartLightStrip property*), 73
time (*kasa.SmartPlug property*), 53
time (*kasa.SmartStrip property*), 66
timezone (*kasa.SmartBulb property*), 47
timezone (*kasa.SmartDevice property*), 26
timezone (*kasa.SmartDimmer property*), 60
timezone (*kasa.SmartLightStrip property*), 73
timezone (*kasa.SmartPlug property*), 53
timezone (*kasa.SmartStrip property*), 66
turn_off() (*kasa.SmartBulb method*), 47
turn_off() (*kasa.SmartDevice method*), 26
turn_off() (*kasa.SmartDimmer method*), 60
turn_off() (*kasa.SmartLightStrip method*), 73
turn_off() (*kasa.SmartPlug method*), 53
turn_off() (*kasa.SmartStrip method*), 66
turn_on() (*kasa.SmartBulb method*), 47
turn_on() (*kasa.SmartDevice method*), 26
turn_on() (*kasa.SmartDimmer method*), 60
turn_on() (*kasa.SmartLightStrip method*), 73
turn_on() (*kasa.SmartPlug method*), 53
turn_on() (*kasa.SmartStrip method*), 66
TurnOnBehavior (*class in kasa*), 48
TurnOnBehavior.Config (*class in kasa*), 48
TurnOnBehaviors (*class in kasa*), 48

U

update() (*kasa.SmartBulb method*), 47
update() (*kasa.SmartDevice method*), 26
update() (*kasa.SmartDimmer method*), 60
update() (*kasa.SmartLightStrip method*), 73
update() (*kasa.SmartPlug method*), 53
update() (*kasa.SmartStrip method*), 66
update_from_discover_info() (*kasa.SmartBulb method*), 47
update_from_discover_info() (*kasa.SmartDevice method*), 26
update_from_discover_info() (*kasa.SmartDimmer method*), 60
update_from_discover_info() (*kasa.SmartLightStrip method*), 73
update_from_discover_info() (*kasa.SmartPlug method*), 53
update_from_discover_info() (*kasa.SmartStrip method*), 66

V

valid_temperature_range (*kasa.SmartBulb property*), 47
valid_temperature_range (*kasa.SmartLightStrip property*), 73
validate_assignment (*kasa.TurnOnBehavior.Config attribute*), 48

W

wifi_join() (*kasa.SmartBulb method*), 47
wifi_join() (*kasa.SmartDevice method*), 26
wifi_join() (*kasa.SmartDimmer method*), 60
wifi_join() (*kasa.SmartLightStrip method*), 73
wifi_join() (*kasa.SmartPlug method*), 53
wifi_join() (*kasa.SmartStrip method*), 66
wifi_scan() (*kasa.SmartBulb method*), 47

wifi_scan() (*kasa.SmartDevice method*), 26
wifi_scan() (*kasa.SmartDimmer method*), 60
wifi_scan() (*kasa.SmartLightStrip method*), 73
wifi_scan() (*kasa.SmartPlug method*), 53
wifi_scan() (*kasa.SmartStrip method*), 66