python-kasa

python-kasa developers

Feb 02, 2024

CONTENTS

1 Discovering devices **3**

2 Basic functionalities **5**
- 2.1 Bulbs . . . 5
- 2.2 Power strips . . . 6

3 Energy meter **7**

4 Library usage **9**
- 4.1 Contributing . . . 9
- 4.2 Supported devices . . . 10
- 4.3 Resources . . . 13

5 Command-line usage **15**
- 5.1 Discovery . . . 15
- 5.2 Provisioning . . . 16
- 5.3 `kasa --help` . . . 16

6 Discovering devices **19**
- 6.1 Discovery . . . 19
- 6.2 API documentation . . . 20

7 Common API **23**
- 7.1 SmartDevice class . . . 23
- 7.2 DeviceConfig class . . . 25
- 7.3 Energy Consumption and Usage Statistics . . . 25
- 7.4 API documentation . . . 25

8 Library Design & Modules **33**
- 8.1 Initialization . . . 33
- 8.2 Update Cycle . . . 33
- 8.3 Modules . . . 34
- 8.4 Protocols and Transports . . . 34
- 8.5 API documentation for modules . . . 35
- 8.6 API documentation for protocols and transports . . . 44

9 Bulbs **49**
- 9.1 Supported features . . . 49
- 9.2 Currently unsupported . . . 49
- 9.3 Transitions . . . 50
- 9.4 Command-line usage . . . 50

9.5 API documentation 50

10 Plugs **61**
10.1 API documentation 61

11 Dimmers **67**
11.1 API documentation 67

12 Smart strips **73**
12.1 Command-line usage 73
12.2 API documentation 73

13 Light strips **79**
13.1 API documentation 79

Python Module Index **87**

Index **89**

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

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

Contributions in any form (adding missing features, reporting issues, fixing or triaging existing ones, improving the documentation, or device donations) are more than welcome!

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
```

If you have not yet provisioned your device, you can do so using the cli tool.

CHAPTER

ONE

DISCOVERING DEVICES

Running `kasa discover` will send discovery packets to the default broadcast address (`255.255.255.255`) to discover supported devices. If your system has multiple network interfaces, you can specify the broadcast address using the `--target` option.

The `discover` command will automatically execute the `state` command on all the discovered devices:

```
$ kasa discover
Discovering devices on 255.255.255.255 for 3 seconds

== Bulb McBulby - KL130(EU) ==
        Host: 192.168.xx.xx
        Port: 9999
        Device state: True
        == Generic information ==
        Time:         2023-12-05 14:33:23 (tz: {'index': 6, 'err_code': 0}
        Hardware:     1.0
        Software:     1.8.8 Build 190613 Rel.123436
        MAC (rssi):   1c:3b:f3:xx:xx:xx (-56)
        Location:     {'latitude': None, 'longitude': None}

        == Device specific information ==
        Brightness: 16
        Is dimmable: True
        Color temperature: 2500
        Valid temperature range: ColorTempRange(min=2500, max=9000)
        HSV: HSV(hue=0, saturation=0, value=16)
        Presets:
                index=0 brightness=50 hue=0 saturation=0 color_temp=2500 custom=None␣
↪id=None mode=None
                index=1 brightness=100 hue=299 saturation=95 color_temp=0 custom=None␣
↪id=None mode=None
                index=2 brightness=100 hue=120 saturation=75 color_temp=0 custom=None␣
↪id=None mode=None
                index=3 brightness=100 hue=240 saturation=75 color_temp=0 custom=None␣
↪id=None mode=None

        == Current State ==
        <EmeterStatus power=2.4 voltage=None current=None total=None>

        == Modules ==
        + <Module Schedule (smartlife.iot.common.schedule) for 192.168.xx.xx>
```

(continues on next page)

(continued from previous page)

```
        + <Module Usage (smartlife.iot.common.schedule) for 192.168.xx.xx>
        + <Module Antitheft (smartlife.iot.common.anti_theft) for 192.168.xx.xx>
        + <Module Time (smartlife.iot.common.timesetting) for 192.168.xx.xx>
        + <Module Emeter (smartlife.iot.common.emeter) for 192.168.xx.xx>
        - <Module Countdown (countdown) for 192.168.xx.xx>
        + <Module Cloud (smartlife.iot.common.cloud) for 192.168.xx.xx>
```

If your device requires authentication to control it, you need to pass the credentials using `--username` and `--password` options.

CHAPTER

TWO

BASIC FUNCTIONALITIES

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

The syntax to control device is `kasa --host <ip address> <command>`. Use `kasa --help` (or consult the documentation) to get a list of all available commands and options. Some examples of available options include JSON output (`--json`), defining timeouts (`--timeout` and `--discovery-timeout`).

Each individual command may also have additional options, which are shown when called with the `--help` option. For example, `--transition` on bulbs requests a smooth state change, while `--name` and `--index` are used on power strips to select the socket to act on:

```
$ kasa on --help

Usage: kasa on [OPTIONS]

  Turn the device on.

Options:
  --index INTEGER
  --name TEXT
  --transition INTEGER
  --help                Show this message and exit.
```

2.1 Bulbs

Common commands for bulbs and light strips include:

- `brightness` to control the brightness
- `hsv` to control the colors
- `temperature` to control the color temperatures

When executed without parameters, these commands will report the current state.

Some devices support `--transition` option to perform a smooth state change. For example, the following turns the light to 30% brightness over a period of five seconds:

```
$ kasa --host <addr> brightness --transition 5000 30
```

See `--help` for additional options and the documentation for more details about supported features and limitations.

2.2 Power strips

Each individual socket can be controlled separately by passing `--index` or `--name` to the command. If neither option is defined, the commands act on the whole power strip.

For example:

```
$ kasa --host <addr> off  # turns off all sockets
$ kasa --host <addr> off --name 'Socket1'  # turns off socket named 'Socket1'
```

See `--help` for additional options and the documentation for more details about supported features and limitations.

CHAPTER

THREE

ENERGY METER

Running `kasa emeter` command will return the current consumption. Possible options include `--year` and `--month` for retrieving historical state, and reseting the counters can be done with `--erase`.

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

CHAPTER

FOUR

LIBRARY USAGE

If you want to use this library in your own project, a good starting point is to check the documentation on discovering devices. You can find several code examples in the API documentation of each of the implementation base classes, check out the documentation for the base class shared by all supported devices.

The library design and module structure is described in a separate page.

The device type specific documentation can be found in their separate pages:

- Plugs
- Bulbs
- Dimmers
- Power strips
- Light strips

4.1 Contributing

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

4.1.1 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.

4.1.2 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.

4.1.3 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.

4.1.4 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. Note, that this works currently only on kasa-branded devices which use port 9999 for communications.

4.2 Supported devices

In principle, most kasa-branded devices that are locally controllable using the official Kasa mobile app work with this library.

The following lists 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 (use `python -m devtools.dump_devinfo` to generate one).**

4.2.1 Plugs

- HS100
- HS103
- HS105
- HS107
- HS110
- KP100
- KP105
- KP115
- KP125
- KP125M *See note below*
- KP401
- EP10
- EP25 *See note below*

4.2.2 Power Strips

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

4.2.3 Wall switches

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

4.2.4 Bulbs

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

4.2.5 Light strips

- KL400L5
- KL420L5
- KL430

4.2.6 Tapo branded devices

The library has recently added a limited supported for devices that carry Tapo branding.

At the moment, the following devices have been confirmed to work:

Plugs

- Tapo P110
- Tapo P125M
- Tapo P135 (dimming not yet supported)
- Tapo TP15

Bulbs

- Tapo L510B
- Tapo L510E
- Tapo L530E

Light strips

- Tapo L900-5
- Tapo L900-10
- Tapo L920-5
- Tapo L930-5

Wall switches

- Tapo S500D
- Tapo S505

Power strips

- Tapo P300
- Tapo TP25

4.2.7 Newer Kasa branded devices

Some newer hardware versions of Kasa branded devices are now using the same protocol as Tapo branded devices. Support for these devices is currently limited as per TAPO branded devices:

- Kasa EP25 (plug) hw_version 2.6
- Kasa KP125M (plug)
- Kasa KS205 (Wifi/Matter Wall Switch)
- Kasa KS225 (Wifi/Matter Wall Dimmer Switch)

If your device is unlisted but working, please open a pull request to update the list and add a fixture file (use `python -m devtools.dump_devinfo` to generate one).

4.3 Resources

4.3.1 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.

4.3.2 Library Users

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

4.3.3 TP-Link Tapo support

This library has recently added a limited supported for devices that carry Tapo branding. That support is currently limited to the cli. The package `kasa.tapo` is in flux and if you use it directly you should expect it could break in future releases until this statement is removed.

Other TAPO libraries are:

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

- Home Assistant integration

CHAPTER

FIVE

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. Note that the `--type` parameter only works for legacy devices using port 9999.

To avoid discovering the devices for newer KASA or TAPO devices using port 20002 for discovery the `--device-family`, `-encrypt-type` and optional `-login-version` options can be passed and the devices will probably require authentication via `--username` and `--password`. Refer to `kasa --help` for detailed usage.

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.

5.1 Discovery

The tool can automatically discover supported devices using a broadcast-based discovery protocol. This works by sending an UDP datagram on ports 9999 and 20002 to the broadcast address (defaulting to `255.255.255.255`).

Newer devices that respond on port 20002 will require TP-Link cloud credentials to be passed (unless they have never been connected to the TP-Link cloud) or they will report as having failed authentication when trying to query the device. Use `--username` and `--password` options to specify credentials. These values can also be set as environment variables via `KASA_USERNAME` and `KASA_PASSWORD`.

On multihomed systems, you can use `--target` option to specify the broadcast 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.

5.2 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.

Note: For devices requiring authentication, the device-stored credentials can be changed using the `update-credentials` commands, for example, to match with other cloud-connected devices. However, note that communications with devices provisioned using this method will stop working when connected to the cloud.

5.3 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]
  -v, --verbose                   Be more verbose on output
  -d, --debug                     Print debug output
  --type [plug|bulb|dimmer|strip|lightstrip]
  --json / --no-json              Output raw device response as JSON.
  --encrypt-type [KLAP|AES|XOR]
  --device-family [IOT.SMARTPLUGSWITCH|IOT.SMARTBULB|SMART.KASAPLUG|SMART.
↪KASASWITCH|SMART.TAPOPLUG|SMART.TAPOBULB|SMART.TAPOSWITCH]
  --login-version INTEGER
  --timeout INTEGER               Timeout for device communications.
                                  [default: 5]
  --discovery-timeout INTEGER     Timeout for discovery.  [default: 3]
  --username TEXT                 Username/email address to authenticate to
                                  device.
  --password TEXT                 Password to use to authenticate to device.
  --credentials-hash TEXT         Hashed credentials used to authenticate to
                                  the device.
  --version                       Show the version and exit.
  --help                          Show this message and exit.
```

(continues on next page)

(continued from previous page)

```
Commands:
  alias                Get or set the device (or plug) alias.
  brightness           Get or set brightness.
  command              Run a raw command on the device.
  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.
  update-credentials   Update device credentials for authenticated devices.
  usage                Query usage for historical consumption.
  wifi                 Commands to control wifi settings.
```

CHAPTER

SIX

DISCOVERING DEVICES

Contents

- *Discovery*
- *API documentation*

6.1 Discovery

Discovery works by sending broadcast UDP packets to two known TP-link discovery ports, 9999 and 20002. Port 9999 is used for legacy devices that do not use strong encryption and 20002 is for newer devices that use different levels of encryption. If a device uses port 20002 for discovery you will obtain some basic information from the device via discovery, but you will need to await *SmartDevice.update()* to get full device information. Credentials will most likely be required for port 20002 devices although if the device has never been connected to the tplink cloud it may work without credentials.

To query or update the device requires authentication via *Credentials* and if this is invalid or not provided it will raise an *AuthenticationException*.

If discovery encounters an unsupported device when calling via *Discover.discover_single()* it will raise a *UnsupportedDeviceException*. If discovery encounters a device when calling *Discover.discover()*, you can provide a callback to the `on_unsupported` parameter to handle these.

Example:

```
import asyncio
from kasa import Discover, Credentials

async def main():
    device = await Discover.discover_single(
        "127.0.0.1",
        credentials=Credentials("myusername", "mypassword"),
        discovery_timeout=10
    )

    await device.update()  # Request the update
    print(device.alias)  # Print out the alias

    devices = await Discover.discover(
        credentials=Credentials("myusername", "mypassword"),
```

(continues on next page)

(continued from previous page)

```
        discovery_timeout=10
    )
    for ip, device in devices.items():
        await device.update()
        print(device.alias)

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

6.2 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 *DeviceConfig* of the device is already known, you can initialize the corresponding device class directly without discovery.

The protocol uses UDP broadcast datagrams on port 9999 and 20002 for discovery. Legacy devices support discovery on port 9999 and newer devices on 20002.

Newer devices that respond on port 20002 will most likely require TP-Link cloud credentials to be passed if queries or updates are to be performed on the returned devices.

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, discovery_timeout=5, discovery_packets=3, interface=None, on_unsupported=None, credentials=None, port=None, timeout=None) → Dict[str, SmartDevice]

Discover supported devices.

Sends discovery message to 255.255.255.255:9999 and 255.255.255.255:20002 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
- **discovery_timeout** – Seconds to wait for responses, defaults to 5
- **discovery_packets** – Number of discovery packets to broadcast
- **interface** – Bind to specific interface
- **on_unsupported** – Optional callback when unsupported devices are discovered
- **credentials** – Credentials for devices requiring authentication
- **port** – Override the discovery port for devices listening on 9999
- **timeout** – Query timeout in seconds for devices returned by discovery

Returns

dictionary with discovered devices

async static discover_single(*host: str*, *, *discovery_timeout: int = 5*, *port: Optional[int] = None*, *timeout: Optional[int] = None*, *credentials: Optional[*Credentials*] = None*) → *SmartDevice*

Discover a single device by the given IP address.

It is generally preferred to avoid *discover_single()* and use *SmartDevice.connect()* 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
- **discovery_timeout** – Timeout in seconds for discovery
- **port** – Optionally set a different port for legacy devices using port 9999
- **timeout** – Timeout in seconds device for devices queries
- **credentials** – Credentials for devices that require authentication

Return type

SmartDevice

Returns

Object for querying/controlling found device.

CHAPTER

SEVEN

COMMON API

Contents

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

7.1 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 for legacy devices:

```
import asyncio
from kasa import SmartPlug

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

(continues on next page)

(continued from previous page)

```
    await p.update()  # Request the update
    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 are connecting to a newer KASA or TAPO device you can get the device via discovery or connect directly with *DeviceConfig*:

```
import asyncio
from kasa import Discover, Credentials

async def main():
    device = await Discover.discover_single(
        "127.0.0.1",
        credentials=Credentials("myusername", "mypassword"),
        discovery_timeout=10
    )

    config = device.config # DeviceConfig.to_dict() can be used to store for later

    # To connect directly later without discovery

    later_device = await SmartDevice.connect(config=config)

    await later_device.update()

    print(later_device.alias)  # Print out the alias
```

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*.

7.2 DeviceConfig class

The *DeviceConfig* class can be used to initialise devices with parameters to allow them to be connected to without using discovery. This is required for newer KASA and TAPO devices that use different protocols for communication and will not respond on port 9999 but instead use different encryption protocols over http port 80. Currently there are three known types of encryption for TP-Link devices and two different protocols. Devices with automatic firmware updates enabled may update to newer versions of the encryption without separate notice, so discovery can be helpful to determine the correct config.

To connect directly pass a *DeviceConfig* object to *SmartDevice.connect()*.

A *DeviceConfig* can be constucted manually if you know the *DeviceConfig.connection_type* values for the device or alternatively the config can be retrieved from *SmartDevice.config* post discovery and then re-used.

7.3 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.

7.3.1 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.

7.3.2 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}")
```

7.4 API documentation

class kasa.**SmartDevice**(*host: str*, ***, *config: Optional[*DeviceConfig*] = None*, *protocol: Optional[*BaseProtocol*] = 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:00:00:00
```

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': '00000000000000000000000000000000',
```

(continues on next page)

(continued from previous page)

```
 'fwId': '00000000000000000000000000000000',
 'oemId': '00000000000000000000000000000000',
 '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=0.928511 voltage=231.067823 current=0.014937 total=55.139>
>>> 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: Module*)

Register a module.

property alias: Optional[str]

Return device name (alias).

property config: *DeviceConfig*

Return the device configuration.

async static connect(**, host: Optional[str] = None*, *config: Optional[*DeviceConfig*] = None*) → *SmartDevice*

Connect to a single device by the given hostname or device configuration.

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

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.

Parameters

- **host** – Hostname of device to query
- **config** – Connection parameters to ensure the correct protocol and connection options are used.

Return type

SmartDevice

Returns
Object for querying/controlling found device.

property credentials: Optional[*Credentials*]
The device credentials.

property credentials_hash: Optional[str]
The protocol specific hash of the credentials the device is using.

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: DeviceType
Return the device type.

async disconnect()
Disconnect and close any underlying connection resources.

property emeter_realtime: 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() → EmeterStatus
Retrieve current energy readings.

get_plug_by_index(*index: int*) → *SmartDevice*
Return child device for the given index.

get_plug_by_name(*name: str*) → *SmartDevice*
Return child device for the given name.

async get_sys_info() → Dict[str, Any]
Retrieve system information.

async get_time() → Optional[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 host: str
The device host.

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]

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

property port: int

The device port.

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

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: str*, *password: str*, *keytype: str = '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[WifiNetwork]

Scan for available wifi networks.

class kasa.**DeviceConfig**(*host: str*, *timeout: ~typing.Optional[int] = 5*, *port_override: ~typing.Optional[int] = None*, *credentials: ~typing.Optional[~kasa.credentials.Credentials] = None*, *credentials_hash: ~typing.Optional[str] = None*, *batch_size: ~typing.Optional[int] = None*, *connection_type: ~kasa.deviceconfig.ConnectionType = <factory>*, *uses_http: bool = False*, *http_client: ~typing.Optional[ClientSession] = None*)

Class to represent paramaters that determine how to connect to devices.

DEFAULT_TIMEOUT = 5

host: str

IP address or hostname

timeout: Optional[int] = 5

Timeout for querying the device

port_override: Optional[int] = None

Override the default 9999 port to support port forwarding

credentials: Optional[*Credentials*] = None

Credentials for devices requiring authentication

credentials_hash: Optional[str] = None

Credentials hash for devices requiring authentication. If credentials are also supplied they take precendence over credentials_hash. Credentials hash can be retrieved from *SmartDevice.credentials_hash*

batch_size: Optional[int] = None

The protocol specific type of connection. Defaults to the legacy type.

connection_type: ConnectionType

The batch size for protoools supporting multiple request batches.

uses_http: bool = False

True if the device uses http. Consumers should retrieve rather than set this in order to determine whether they should pass a custom http client if desired.

http_client: Optional[ClientSession] = None

Set a custom http_client for the device to use.

to_dict(*, *credentials_hash: Optional[str] = None*, *exclude_credentials: bool = False*) → Dict[str, Dict[str, str]]

Convert device config to dict.

static from_dict(*config_dict: Dict[str, Dict[str, str]]*) → *DeviceConfig*

Return device config from dict.

class kasa.**Credentials**(*username: str = ''*, *password: str = ''*)

Credentials for authentication.

password: str = ''

Password of the cloud account

username: str = ''

Username (email address) of the cloud account

class kasa.**SmartDeviceException**(**args: Any*, ***kwargs: Any*)

Base exception for device errors.

class kasa.**AuthenticationException**(**args: Any*, ***kwargs: Any*)

Base exception for device authentication errors.

class kasa.**UnsupportedDeviceException**(**args: Any*, ***kwargs: Any*)

Exception for trying to connect to unsupported devices.

CHAPTER

EIGHT

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*
- *Protocols and Transports*
- *API documentation for modules*
- *API documentation for protocols and transports*

8.1 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()*.

The *connect()* also enables support for connecting to new KASA SMART protocol and TAPO devices directly using the parameter *DeviceConfig*. Simply serialize the *config* property via *to_dict()* and then deserialize it later with *from_dict()* and then pass it into *connect()*.

8.2 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.

8.3 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()*.

8.4 Protocols and Transports

The library supports two different TP-Link protocols, `IOT` and `SMART`. `IOT` is the original Kasa protocol and `SMART` is the newer protocol supported by TAPO devices and newer KASA devices. The original protocol has a `target`, `command`, `args` interface whereas the new protocol uses a different set of commands and has a `method`, `parameters` interface. Confusingly TP-Link originally called the Kasa line "Kasa Smart" and hence this library used "Smart" in a lot of the module and class names but actually they were built to work with the `IOT` protocol.

In 2021 TP-Link started updating the underlying communication transport used by Kasa devices to make them more secure. It switched from a TCP connection with static XOR type of encryption to a transport called `KLAP` which communicates over http and uses handshakes to negotiate a dynamic encryption cipher. This automatic update was put on hold and only seemed to affect UK HS100 models.

In 2023 TP-Link started updating the underlying communication transport used by Tapo devices to make them more secure. It switched from AES encryption via public key exchange to use `KLAP` encryption and negotiation due to concerns around impersonation with AES. The encryption cipher is the same as for Kasa KLAP but the handshake seeds are slightly different. Also in 2023 TP-Link started releasing newer Kasa branded devices using the `SMART` protocol. This appears to be driven by hardware version rather than firmware.

In order to support these different configurations the library migrated from a single protocol class `TPLinkSmartHomeProtocol` to support pluggable transports and protocols. The classes providing this functionality are:

- *BaseProtocol*
- *IotProtocol*
- *SmartProtocol*
- *BaseTransport*
- *XorTransport*
- *AesTransport*
- *KlapTransport*
- *KlapTransportV2*

8.5 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: 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[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: 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: 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[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: 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: 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[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[Action] = None*, *stime_opt: TimeOption*, *smin: int*, *eact: Optional[Action] = None*, *etime_opt: TimeOption*, *emin: int*, *s_light: Optional[Dict] = None*)

Representation of a rule.

Config

alias of 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[Action]

emin: int

enable: bool

etime_opt: 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, Path]*, ***, *content_type: str = None*, *encoding: str = 'utf8'*, *proto: 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: Protocol = None*, *allow_pickle: bool = False*) → Model

repeat: bool

s_light: Optional[Dict]

sact: Optional[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: 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: 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[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: 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[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
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.

8.6 API documentation for protocols and transports

class kasa.protocol.**BaseProtocol**(*, *transport:* BaseTransport)

Base class for all TP-Link Smart Home communication.

abstract async close() → None

Close the protocol. Abstract method to be overriden.

property config: *DeviceConfig*

Return the connection parameters the device is using.

abstract async query(*request: Union[str, Dict]*, *retry_count: int = 3*) → Dict

Query the device for the protocol. Abstract method to be overriden.

class kasa.iotprotocol.**IotProtocol**(*, *transport:* BaseTransport)

Class for the legacy TPLink IOT KASA Protocol.

BACKOFF_SECONDS_AFTER_TIMEOUT = 1

async close() → None

Close the underlying transport.

property config: *DeviceConfig*

Return the connection parameters the device is using.

async query(*request: Union[str, Dict]*, *retry_count: int = 3*) → Dict

Query the device retrying for retry_count on failure.

class kasa.smartprotocol.**SmartProtocol**(*, *transport:* BaseTransport)

Class for the new TPLink SMART protocol.

BACKOFF_SECONDS_AFTER_TIMEOUT = 1

DEFAULT_MULTI_REQUEST_BATCH_SIZE = 5

async close() → None
Close the underlying transport.

property config: *DeviceConfig*
Return the connection parameters the device is using.

get_smart_request(*method*, *params=None*) → str
Get a request message as a string.

async query(*request: Union[str, Dict]*, *retry_count: int = 3*) → Dict
Query the device retrying for retry_count on failure.

class kasa.protocol.**BaseTransport**(*, *config:* DeviceConfig)
Base class for all TP-Link protocol transports.

DEFAULT_TIMEOUT = 5

abstract async close() → None
Close the transport. Abstract method to be overriden.

abstract property credentials_hash: str
The hashed credentials used by the transport.

abstract property default_port: int
The default port for the transport.

abstract async reset() → None
Reset internal state.

abstract async send(*request: str*) → Dict
Send a message to the device and return a response.

class kasa.xortransport.**XorTransport**(*, *config:* DeviceConfig)
XorTransport class.

BLOCK_SIZE = 4

DEFAULT_PORT: int = 9999

DEFAULT_TIMEOUT = 5

async close() → None
Close the connection.

close_without_wait() → None
Close the connection without waiting for the connection to close.

property credentials_hash: str
The hashed credentials used by the transport.

property default_port
Default port for the transport.

async reset() → None

Reset the transport.

The transport cannot be reset so we must close instead.

async send(*request: str*) → Dict

Send a message to the device and return a response.

class kasa.klaptransport.**KlapTransport**(*, *config:* DeviceConfig)

Implementation of the KLAP encryption protocol.

KLAP is the name used in device discovery for TP-Link's new encryption protocol, used by newer firmware versions.

DEFAULT_PORT: int = 80

DEFAULT_TIMEOUT = 5

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

SESSION_COOKIE_NAME = 'TP_SESSIONID'

TIMEOUT_COOKIE_NAME = 'TIMEOUT'

async close() → None

Close the http client and reset internal state.

property credentials_hash: str

The hashed credentials used by the transport.

property default_port

Default port for the transport.

static generate_auth_hash(*creds:* Credentials)

Generate an md5 auth hash for the protocol on the supplied credentials.

static generate_owner_hash(*creds:* Credentials)

Return the MD5 hash of the username in this object.

static handshake1_seed_auth_hash(*local_seed: bytes*, *remote_seed: bytes*, *auth_hash: bytes*)

Generate an md5 auth hash for the protocol on the supplied credentials.

static handshake2_seed_auth_hash(*local_seed: bytes*, *remote_seed: bytes*, *auth_hash: bytes*)

Generate an md5 auth hash for the protocol on the supplied credentials.

async perform_handshake() → Any

Perform handshake1 and handshake2.

Sets the encryption_session if successful.

async perform_handshake1() → Tuple[bytes, bytes, bytes]

Perform handshake1.

async perform_handshake2(*local_seed*, *remote_seed*, *auth_hash*) → KlapEncryptionSession

Perform handshake2.

async reset() → None

Reset internal handshake state.

async send(*request: str*)

Send the request.

class kasa.klaptransport.KlapTransportV2(*, *config:* DeviceConfig)

Implementation of the KLAP encryption protocol with v2 hanshake hashes.

DEFAULT_PORT: int = 80

DEFAULT_TIMEOUT = 5

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

SESSION_COOKIE_NAME = 'TP_SESSIONID'

TIMEOUT_COOKIE_NAME = 'TIMEOUT'

async close() → None

Close the http client and reset internal state.

property credentials_hash: str

The hashed credentials used by the transport.

property default_port

Default port for the transport.

static generate_auth_hash(*creds:* Credentials)

Generate an md5 auth hash for the protocol on the supplied credentials.

static generate_owner_hash(*creds:* Credentials)

Return the MD5 hash of the username in this object.

static handshake1_seed_auth_hash(*local_seed: bytes*, *remote_seed: bytes*, *auth_hash: bytes*)

Generate an md5 auth hash for the protocol on the supplied credentials.

static handshake2_seed_auth_hash(*local_seed: bytes*, *remote_seed: bytes*, *auth_hash: bytes*)

Generate an md5 auth hash for the protocol on the supplied credentials.

async perform_handshake() → Any

Perform handshake1 and handshake2.

Sets the encryption_session if successful.

async perform_handshake1() → Tuple[bytes, bytes, bytes]

Perform handshake1.

async perform_handshake2(*local_seed*, *remote_seed*, *auth_hash*) → KlapEncryptionSession

Perform handshake2.

async reset() → None

Reset internal handshake state.

async send(*request: str*)

Send the request.

class kasa.aestransport.AesTransport(*, *config:* DeviceConfig)

Implementation of the AES encryption protocol.

AES is the name used in device discovery for TP-Link's TAPO encryption protocol, sometimes used by newer firmware versions on kasa devices.

COMMON_HEADERS = {'Accept': 'application/json', 'Content-Type': 'application/json', 'requestByApp': 'true'}

CONTENT_LENGTH = 'Content-Length'

DEFAULT_PORT: int = 80

DEFAULT_TIMEOUT = 5

KEY_PAIR_CONTENT_LENGTH = 314

SESSION_COOKIE_NAME = 'TP_SESSIONID'

TIMEOUT_COOKIE_NAME = 'TIMEOUT'

async close() → None
Close the http client and reset internal state.

property credentials_hash: str
The hashed credentials used by the transport.

property default_port: int
Default port for the transport.

static hash_credentials(*login_v2: bool*, *credentials:* Credentials) → Tuple[str, str]
Hash the credentials.

async perform_handshake() → None
Perform the handshake.

async perform_login()
Login to the device.

async reset() → None
Reset internal handshake and login state.

async send(*request: str*) → Dict[str, Any]
Send the request.

async send_secure_passthrough(*request: str*) → Dict[str, Any]
Send encrypted message as passthrough.

async try_login(*login_params: Dict[str, Any]*) → None
Try to login with supplied login_params.

CHAPTER

NINE

BULBS

Contents

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

9.1 Supported features

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

9.2 Currently unsupported

- Setting the default transitions
- Timers

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

9.3 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.

9.4 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
```

9.5 API documentation

class kasa.**SmartBulb**(*host: str*, ***, *config: Optional[DeviceConfig] = None*, *protocol: Optional[BaseProtocol] = 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: Module*)

Register a module.

property alias: Optional[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.

property config: *DeviceConfig*

Return the device configuration.

async static connect(*, *host: Optional[str] = None*, *config: Optional[*DeviceConfig*] = None*) → *SmartDevice*

Connect to a single device by the given hostname or device configuration.

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

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.

Parameters

- **host** – Hostname of device to query
- **config** – Connection parameters to ensure the correct protocol and connection options are used.

Return type

SmartDevice

Returns
Object for querying/controlling found device.

property credentials: Optional[*Credentials*]
The device credentials.

property credentials_hash: Optional[str]
The protocol specific hash of the credentials the device is using.

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: DeviceType
Return the device type.

async disconnect()
Disconnect and close any underlying connection resources.

property emeter_realtime: 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() → 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*) → *SmartDevice*
Return child device for the given index.

get_plug_by_name(*name: str*) → *SmartDevice*
Return child device for the given name.

async get_sys_info() → Dict[str, Any]
Retrieve system information.

async get_time() → Optional[datetime]
Return current time from the device, if available.

async get_timezone() → Dict
Return timezone information.

async get_turn_on_behavior() → *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 host: str
The device host.

property hsv: 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]

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

property port: int

The device port.

property presets: List[*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:* 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:* 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

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: ColorTempRange

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

Returns
White temperature range in Kelvin (minimum, maximum)

async wifi_join(*ssid: str*, *password: str*, *keytype: str = '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[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.smartbulb.BehaviorMode(value, names=None, *values, module=None, qualname=None, type=None, start=1, boundary=None)

Enum to present type of turn on behavior.

Last = 'last_status'

Return to the last state known state.

Preset = 'customize_preset'

Use chosen preset.

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

Model to contain turn on behaviors.

hard: *TurnOnBehavior*

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

soft: *TurnOnBehavior*

The behavior when the bulb is turned on programmatically.

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

Wanted behavior

preset: Optional[int]

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

CHAPTER

TEN

PLUGS

Contents

- *API documentation*

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

10.1 API documentation

class kasa.**SmartPlug**(*host: str*, ***, *config: Optional[*DeviceConfig*] = None*, *protocol: Optional[*BaseProtocol*] = 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 *SmartDeviceException*s, 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: Module*)

Register a module.

property alias: Optional[str]

Return device name (alias).

children: List['SmartDevice']

property config: *DeviceConfig*

Return the device configuration.

async static connect(*, *host: Optional[str] = None*, *config: Optional[*DeviceConfig*] = None*) → *SmartDevice*

Connect to a single device by the given hostname or device configuration.

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

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.

Parameters

- **host** – Hostname of device to query
- **config** – Connection parameters to ensure the correct protocol and connection options are used.

Return type
SmartDevice

Returns
Object for querying/controlling found device.

property credentials: Optional[*Credentials*]

The device credentials.

property credentials_hash: Optional[str]

The protocol specific hash of the credentials the device is using.

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: DeviceType

Return the device type.

async disconnect()

Disconnect and close any underlying connection resources.

property emeter_realtime: 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() → EmeterStatus

Retrieve current energy readings.

get_plug_by_index(*index: int*) → *SmartDevice*

Return child device for the given index.

get_plug_by_name(*name: str*) → *SmartDevice*

Return child device for the given name.

async get_sys_info() → Dict[str, Any]

Retrieve system information.

async get_time() → Optional[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 host: str

The device host.

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]

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

property port: int

The device port.

protocol: *BaseProtocol*

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

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: str*, *password: str*, *keytype: str = '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[WifiNetwork]

Scan for available wifi networks.

CHAPTER

ELEVEN

DIMMERS

Contents

- *API documentation*

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

11.1 API documentation

class kasa.**SmartDimmer**(*host: str*, ***, *config: Optional[*DeviceConfig*] = None*, *protocol: Optional[*BaseProtocol*] = 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 *SmartDeviceException*s, 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: Module*)

Register a module.

property alias: Optional[str]

Return device name (alias).

property brightness: int

Return current brightness on dimmers.

Will return a range between 0 - 100.

children: List['SmartDevice']

property config: *DeviceConfig*

Return the device configuration.

async static connect(*, *host: Optional[str] = None*, *config: Optional[*DeviceConfig*] = None*) → *SmartDevice*

Connect to a single device by the given hostname or device configuration.

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

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.

Parameters

- **host** – Hostname of device to query
- **config** – Connection parameters to ensure the correct protocol and connection options are used.

Return type
SmartDevice

Returns
Object for querying/controlling found device.

property credentials: Optional[*Credentials*]

The device credentials.

property credentials_hash: Optional[str]

The protocol specific hash of the credentials the device is using.

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: DeviceType

Return the device type.

async disconnect()

Disconnect and close any underlying connection resources.

property emeter_realtime: 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() → EmeterStatus

Retrieve current energy readings.

get_plug_by_index(*index: int*) → *SmartDevice*

Return child device for the given index.

get_plug_by_name(*name: str*) → *SmartDevice*

Return child device for the given name.

async get_sys_info() → Dict[str, Any]

Retrieve system information.

async get_time() → Optional[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 host: str

The device host.

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]

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

property port: int

The device port.

protocol: *BaseProtocol*

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: ActionType*, *action: 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: 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

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: str*, *password: str*, *keytype: str = '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[WifiNetwork]

Scan for available wifi networks.

CHAPTER

TWELVE

SMART STRIPS

Contents

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

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

12.1 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"
```

12.2 API documentation

class kasa.**SmartStrip**(*host: str*, ***, *config: Optional[*DeviceConfig*] = None*, *protocol: Optional[*BaseProtocol*] = 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 *SmartDeviceException*s, 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: Module*)

Register a module.

property alias: Optional[str]

Return device name (alias).

children: List['SmartDevice']

property config: *DeviceConfig*

Return the device configuration.

async static connect(*, *host: Optional[str] = None*, *config: Optional[*DeviceConfig*] = None*) → *SmartDevice*

Connect to a single device by the given hostname or device configuration.

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

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.

Parameters

- **host** – Hostname of device to query
- **config** – Connection parameters to ensure the correct protocol and connection options are used.

Return type
SmartDevice

Returns
Object for querying/controlling found device.

property credentials: Optional[*Credentials*]

The device credentials.

property credentials_hash: Optional[str]

The protocol specific hash of the credentials the device is using.

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: DeviceType

Return the device type.

async disconnect()

Disconnect and close any underlying connection resources.

property emeter_realtime: 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() → EmeterStatus

Retrieve current energy readings.

get_plug_by_index(*index: int*) → *SmartDevice*

Return child device for the given index.

get_plug_by_name(*name: str*) → *SmartDevice*

Return child device for the given name.

async get_sys_info() → Dict[str, Any]

Retrieve system information.

async get_time() → Optional[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 host: str

The device host.

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]

Return the maximum on-time of all outlets.

property port: int

The device port.

protocol: *BaseProtocol*

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

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: str*, *password: str*, *keytype: str = '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[WifiNetwork]

Scan for available wifi networks.

CHAPTER

THIRTEEN

LIGHT STRIPS

Contents

- *API documentation*

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

13.1 API documentation

class kasa.**SmartLightStrip**(*host: str*, ***, *config: Optional[*DeviceConfig*] = None*, *protocol: Optional[*BaseProtocol*] = 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: Module*)

Register a module.

property alias: Optional[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.

property config: *DeviceConfig*

Return the device configuration.

async static connect(*, *host: Optional[str] = None*, *config: Optional[*DeviceConfig*] = None*) → *SmartDevice*

Connect to a single device by the given hostname or device configuration.

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

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.

Parameters

- **host** – Hostname of device to query
- **config** – Connection parameters to ensure the correct protocol and connection options are used.

Return type
SmartDevice

Returns
Object for querying/controlling found device.

property credentials: Optional[*Credentials*]

The device credentials.

property credentials_hash: Optional[str]

The protocol specific hash of the credentials the device is using.

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: DeviceType

Return the device type.

async disconnect()

Disconnect and close any underlying connection resources.

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: 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() → 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*) → *SmartDevice*

Return child device for the given index.

get_plug_by_name(*name: str*) → *SmartDevice*

Return child device for the given name.

async get_sys_info() → Dict[str, Any]

Retrieve system information.

async get_time() → Optional[datetime]

Return current time from the device, if available.

async get_timezone() → Dict

Return timezone information.

async get_turn_on_behavior() → *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 host: str

The device host.

property hsv: 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]

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

property port: int

The device port.

property presets: List[*SmartBulbPreset*]

Return a list of available bulb setting presets.

protocol: *BaseProtocol*

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:* 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:* 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

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: ColorTempRange

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

Returns
White temperature range in Kelvin (minimum, maximum)

async wifi_join(*ssid: str*, *password: str*, *keytype: str = '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[WifiNetwork]

Scan for available wifi networks.

PYTHON MODULE INDEX

k

kasa, 21
kasa.discover, 17
kasa.modules, 32

INDEX

A

add_module() (*kasa.SmartBulb method*), 52
add_module() (*kasa.SmartDevice method*), 27
add_module() (*kasa.SmartDimmer method*), 67
add_module() (*kasa.SmartLightStrip method*), 80
add_module() (*kasa.SmartPlug method*), 61
add_module() (*kasa.SmartStrip method*), 74
AesTransport (*class in kasa.aestransport*), 47
alias (*kasa.SmartBulb property*), 52
alias (*kasa.SmartDevice property*), 27
alias (*kasa.SmartDimmer property*), 67
alias (*kasa.SmartLightStrip property*), 80
alias (*kasa.SmartPlug property*), 62
alias (*kasa.SmartStrip property*), 74
AuthenticationException (*class in kasa*), 32

B

BACKOFF_SECONDS_AFTER_TIMEOUT (*kasa.iotprotocol.IotProtocol attribute*), 44
BACKOFF_SECONDS_AFTER_TIMEOUT (*kasa.smartprotocol.SmartProtocol attribute*), 44
BaseProtocol (*class in kasa.protocol*), 44
BaseTransport (*class in kasa.protocol*), 45
batch_size (*kasa.DeviceConfig attribute*), 31
BehaviorMode (*class in kasa.smartbulb*), 58
BLOCK_SIZE (*kasa.xortransport.XorTransport attribute*), 45
brightness (*kasa.SmartBulb property*), 52
brightness (*kasa.SmartBulbPreset attribute*), 58
brightness (*kasa.SmartDimmer property*), 68
brightness (*kasa.SmartLightStrip property*), 80

C

children (*kasa.SmartDimmer attribute*), 68
children (*kasa.SmartLightStrip attribute*), 80
children (*kasa.SmartPlug attribute*), 62
children (*kasa.SmartStrip attribute*), 74
close() (*kasa.aestransport.AesTransport method*), 48
close() (*kasa.iotprotocol.IotProtocol method*), 44
close() (*kasa.klaptransport.KlapTransport method*), 46
close() (*kasa.klaptransport.KlapTransportV2 method*), 47
close() (*kasa.protocol.BaseProtocol method*), 44
close() (*kasa.protocol.BaseTransport method*), 45
close() (*kasa.smartprotocol.SmartProtocol method*), 45
close() (*kasa.xortransport.XorTransport method*), 45
close_without_wait() (*kasa.xortransport.XorTransport method*), 45
color_temp (*kasa.SmartBulb property*), 52
color_temp (*kasa.SmartBulbPreset attribute*), 58
color_temp (*kasa.SmartLightStrip property*), 80
COMMON_HEADERS (*kasa.aestransport.AesTransport attribute*), 47
config (*kasa.iotprotocol.IotProtocol property*), 44
config (*kasa.protocol.BaseProtocol property*), 44
config (*kasa.SmartBulb property*), 52
config (*kasa.SmartDevice property*), 27
config (*kasa.SmartDimmer property*), 68
config (*kasa.SmartLightStrip property*), 80
config (*kasa.SmartPlug property*), 62
config (*kasa.smartprotocol.SmartProtocol property*), 45
config (*kasa.SmartStrip property*), 74
connect() (*kasa.SmartBulb static method*), 52
connect() (*kasa.SmartDevice static method*), 27
connect() (*kasa.SmartDimmer static method*), 68
connect() (*kasa.SmartLightStrip static method*), 80
connect() (*kasa.SmartPlug static method*), 62
connect() (*kasa.SmartStrip static method*), 74
connection_type (*kasa.DeviceConfig attribute*), 32
CONTENT_LENGTH (*kasa.aestransport.AesTransport attribute*), 48
Credentials (*class in kasa*), 32
credentials (*kasa.DeviceConfig attribute*), 31
credentials (*kasa.SmartBulb property*), 53
credentials (*kasa.SmartDevice property*), 28
credentials (*kasa.SmartDimmer property*), 68
credentials (*kasa.SmartLightStrip property*), 80
credentials (*kasa.SmartPlug property*), 62
credentials (*kasa.SmartStrip property*), 75
credentials_hash (*kasa.aestransport.AesTransport property*), 48

credentials_hash (*kasa.DeviceConfig attribute*), 31
credentials_hash (*kasa.klaptransport.KlapTransport property*), 46
credentials_hash (*kasa.klaptransport.KlapTransportV2 property*), 47
credentials_hash (*kasa.protocol.BaseTransport property*), 45
credentials_hash (*kasa.SmartBulb property*), 53
credentials_hash (*kasa.SmartDevice property*), 28
credentials_hash (*kasa.SmartDimmer property*), 68
credentials_hash (*kasa.SmartLightStrip property*), 80
credentials_hash (*kasa.SmartPlug property*), 62
credentials_hash (*kasa.SmartStrip property*), 75
credentials_hash (*kasa.xortransport.XorTransport property*), 45
current_consumption() (*kasa.SmartBulb method*), 53
current_consumption() (*kasa.SmartDevice method*), 28
current_consumption() (*kasa.SmartDimmer method*), 68
current_consumption() (*kasa.SmartLightStrip method*), 80
current_consumption() (*kasa.SmartPlug method*), 62
current_consumption() (*kasa.SmartStrip method*), 75
custom (*kasa.SmartBulbPreset attribute*), 58

D

DEFAULT_MULTI_REQUEST_BATCH_SIZE (*kasa.smartprotocol.SmartProtocol attribute*), 45
DEFAULT_PORT (*kasa.aestransport.AesTransport attribute*), 48
default_port (*kasa.aestransport.AesTransport property*), 48
DEFAULT_PORT (*kasa.klaptransport.KlapTransport attribute*), 46
default_port (*kasa.klaptransport.KlapTransport property*), 46
DEFAULT_PORT (*kasa.klaptransport.KlapTransportV2 attribute*), 47
default_port (*kasa.klaptransport.KlapTransportV2 property*), 47
default_port (*kasa.protocol.BaseTransport property*), 45
DEFAULT_PORT (*kasa.xortransport.XorTransport attribute*), 45
default_port (*kasa.xortransport.XorTransport property*), 45
DEFAULT_TIMEOUT (*kasa.aestransport.AesTransport attribute*), 48
DEFAULT_TIMEOUT (*kasa.DeviceConfig attribute*), 31
DEFAULT_TIMEOUT (*kasa.klaptransport.KlapTransport attribute*), 46
DEFAULT_TIMEOUT (*kasa.klaptransport.KlapTransportV2 attribute*), 47
DEFAULT_TIMEOUT (*kasa.protocol.BaseTransport attribute*), 45
DEFAULT_TIMEOUT (*kasa.xortransport.XorTransport attribute*), 45
device_id (*kasa.SmartBulb property*), 53
device_id (*kasa.SmartDevice property*), 28
device_id (*kasa.SmartDimmer property*), 68
device_id (*kasa.SmartLightStrip property*), 80
device_id (*kasa.SmartPlug property*), 62
device_id (*kasa.SmartStrip property*), 75
device_type (*kasa.SmartBulb property*), 53
device_type (*kasa.SmartDevice property*), 28
device_type (*kasa.SmartDimmer property*), 68
device_type (*kasa.SmartLightStrip property*), 80
device_type (*kasa.SmartPlug property*), 62
device_type (*kasa.SmartStrip property*), 75
DeviceConfig (*class in kasa*), 31
DIMMER_SERVICE (*kasa.SmartDimmer attribute*), 67
disconnect() (*kasa.SmartBulb method*), 53
disconnect() (*kasa.SmartDevice method*), 28
disconnect() (*kasa.SmartDimmer method*), 68
disconnect() (*kasa.SmartLightStrip method*), 80
disconnect() (*kasa.SmartPlug method*), 62
disconnect() (*kasa.SmartStrip method*), 75
Discover (*class in kasa*), 20
discover() (*kasa.Discover static method*), 20
discover_single() (*kasa.Discover static method*), 21
DISCOVERY_PORT (*kasa.Discover attribute*), 20
DISCOVERY_PORT_2 (*kasa.Discover attribute*), 20
DISCOVERY_QUERY (*kasa.Discover attribute*), 20
DISCOVERY_QUERY (*kasa.klaptransport.KlapTransport attribute*), 46
DISCOVERY_QUERY (*kasa.klaptransport.KlapTransportV2 attribute*), 47
DISCOVERY_QUERY_2 (*kasa.Discover attribute*), 20

E

effect (*kasa.SmartLightStrip property*), 81
effect_list (*kasa.SmartLightStrip property*), 81
emeter_realtime (*kasa.SmartBulb property*), 53
emeter_realtime (*kasa.SmartDevice property*), 28
emeter_realtime (*kasa.SmartDimmer property*), 68
emeter_realtime (*kasa.SmartLightStrip property*), 81
emeter_realtime (*kasa.SmartPlug property*), 62
emeter_realtime (*kasa.SmartStrip property*), 75
emeter_this_month (*kasa.SmartBulb property*), 53
emeter_this_month (*kasa.SmartDevice property*), 28
emeter_this_month (*kasa.SmartDimmer property*), 68
emeter_this_month (*kasa.SmartLightStrip property*), 81
emeter_this_month (*kasa.SmartPlug property*), 62
emeter_this_month (*kasa.SmartStrip property*), 75

emeter_today (*kasa.SmartBulb property*), 53
emeter_today (*kasa.SmartDevice property*), 28
emeter_today (*kasa.SmartDimmer property*), 68
emeter_today (*kasa.SmartLightStrip property*), 81
emeter_today (*kasa.SmartPlug property*), 62
emeter_today (*kasa.SmartStrip property*), 75
emeter_type (*kasa.SmartBulb attribute*), 53
emeter_type (*kasa.SmartDevice attribute*), 28
emeter_type (*kasa.SmartDimmer attribute*), 69
emeter_type (*kasa.SmartLightStrip attribute*), 81
emeter_type (*kasa.SmartPlug attribute*), 62
emeter_type (*kasa.SmartStrip attribute*), 75
erase_emeter_stats() (*kasa.SmartBulb method*), 53
erase_emeter_stats() (*kasa.SmartDevice method*), 28
erase_emeter_stats() (*kasa.SmartDimmer method*), 69
erase_emeter_stats() (*kasa.SmartLightStrip method*), 81
erase_emeter_stats() (*kasa.SmartPlug method*), 63
erase_emeter_stats() (*kasa.SmartStrip method*), 75

F

features (*kasa.SmartBulb property*), 53
features (*kasa.SmartDevice property*), 28
features (*kasa.SmartDimmer property*), 69
features (*kasa.SmartLightStrip property*), 81
features (*kasa.SmartPlug property*), 63
features (*kasa.SmartStrip property*), 75
from_dict() (*kasa.DeviceConfig static method*), 32

G

generate_auth_hash() (*kasa.klaptransport.KlapTransport static method*), 46
generate_auth_hash() (*kasa.klaptransport.KlapTransportV2 static method*), 47
generate_owner_hash() (*kasa.klaptransport.KlapTransport static method*), 46
generate_owner_hash() (*kasa.klaptransport.KlapTransportV2 static method*), 47
get_behaviors() (*kasa.SmartDimmer method*), 69
get_emeter_daily() (*kasa.SmartBulb method*), 53
get_emeter_daily() (*kasa.SmartDevice method*), 28
get_emeter_daily() (*kasa.SmartDimmer method*), 69
get_emeter_daily() (*kasa.SmartLightStrip method*), 81
get_emeter_daily() (*kasa.SmartPlug method*), 63
get_emeter_daily() (*kasa.SmartStrip method*), 75
get_emeter_monthly() (*kasa.SmartBulb method*), 53
get_emeter_monthly() (*kasa.SmartDevice method*), 28
get_emeter_monthly() (*kasa.SmartDimmer method*), 69
get_emeter_monthly() (*kasa.SmartLightStrip method*), 81
get_emeter_monthly() (*kasa.SmartPlug method*), 63
get_emeter_monthly() (*kasa.SmartStrip method*), 76
get_emeter_realtime() (*kasa.SmartBulb method*), 54
get_emeter_realtime() (*kasa.SmartDevice method*), 29
get_emeter_realtime() (*kasa.SmartDimmer method*), 69
get_emeter_realtime() (*kasa.SmartLightStrip method*), 81
get_emeter_realtime() (*kasa.SmartPlug method*), 63
get_emeter_realtime() (*kasa.SmartStrip method*), 76
get_light_details() (*kasa.SmartBulb method*), 54
get_light_details() (*kasa.SmartLightStrip method*), 82
get_light_state() (*kasa.SmartBulb method*), 54
get_light_state() (*kasa.SmartLightStrip method*), 82
get_plug_by_index() (*kasa.SmartBulb method*), 54
get_plug_by_index() (*kasa.SmartDevice method*), 29
get_plug_by_index() (*kasa.SmartDimmer method*), 69
get_plug_by_index() (*kasa.SmartLightStrip method*), 82
get_plug_by_index() (*kasa.SmartPlug method*), 63
get_plug_by_index() (*kasa.SmartStrip method*), 76
get_plug_by_name() (*kasa.SmartBulb method*), 54
get_plug_by_name() (*kasa.SmartDevice method*), 29
get_plug_by_name() (*kasa.SmartDimmer method*), 69
get_plug_by_name() (*kasa.SmartLightStrip method*), 82
get_plug_by_name() (*kasa.SmartPlug method*), 63
get_plug_by_name() (*kasa.SmartStrip method*), 76
get_smart_request() (*kasa.smartprotocol.SmartProtocol method*), 45
get_sys_info() (*kasa.SmartBulb method*), 54
get_sys_info() (*kasa.SmartDevice method*), 29
get_sys_info() (*kasa.SmartDimmer method*), 69
get_sys_info() (*kasa.SmartLightStrip method*), 82
get_sys_info() (*kasa.SmartPlug method*), 63
get_sys_info() (*kasa.SmartStrip method*), 76
get_time() (*kasa.SmartBulb method*), 54
get_time() (*kasa.SmartDevice method*), 29
get_time() (*kasa.SmartDimmer method*), 69
get_time() (*kasa.SmartLightStrip method*), 82
get_time() (*kasa.SmartPlug method*), 63
get_time() (*kasa.SmartStrip method*), 76
get_timezone() (*kasa.SmartBulb method*), 54
get_timezone() (*kasa.SmartDevice method*), 29

get_timezone() (*kasa.SmartDimmer method*), 69
get_timezone() (*kasa.SmartLightStrip method*), 82
get_timezone() (*kasa.SmartPlug method*), 63
get_timezone() (*kasa.SmartStrip method*), 76
get_turn_on_behavior() (*kasa.SmartBulb method*), 54
get_turn_on_behavior() (*kasa.SmartLightStrip method*), 82

H

handshake1_seed_auth_hash() (*kasa.klaptransport.KlapTransport static method*), 46
handshake1_seed_auth_hash() (*kasa.klaptransport.KlapTransportV2 static method*), 47
handshake2_seed_auth_hash() (*kasa.klaptransport.KlapTransport static method*), 46
handshake2_seed_auth_hash() (*kasa.klaptransport.KlapTransportV2 static method*), 47
hard (*kasa.TurnOnBehaviors attribute*), 58
has_children (*kasa.SmartBulb property*), 54
has_children (*kasa.SmartDevice property*), 29
has_children (*kasa.SmartDimmer property*), 69
has_children (*kasa.SmartLightStrip property*), 82
has_children (*kasa.SmartPlug property*), 63
has_children (*kasa.SmartStrip property*), 76
has_effects (*kasa.SmartBulb property*), 54
has_effects (*kasa.SmartLightStrip property*), 82
has_emeter (*kasa.SmartBulb property*), 54
has_emeter (*kasa.SmartDevice property*), 29
has_emeter (*kasa.SmartDimmer property*), 69
has_emeter (*kasa.SmartLightStrip property*), 82
has_emeter (*kasa.SmartPlug property*), 63
has_emeter (*kasa.SmartStrip property*), 76
hash_credentials() (*kasa.aestransport.AesTransport static method*), 48
host (*kasa.DeviceConfig attribute*), 31
host (*kasa.SmartBulb property*), 54
host (*kasa.SmartDevice property*), 29
host (*kasa.SmartDimmer property*), 70
host (*kasa.SmartLightStrip property*), 82
host (*kasa.SmartPlug property*), 63
host (*kasa.SmartStrip property*), 76
hsv (*kasa.SmartBulb property*), 54
hsv (*kasa.SmartLightStrip property*), 82
http_client (*kasa.DeviceConfig attribute*), 32
hue (*kasa.SmartBulbPreset attribute*), 58
hw_info (*kasa.SmartBulb property*), 54
hw_info (*kasa.SmartDevice property*), 29
hw_info (*kasa.SmartDimmer property*), 70
hw_info (*kasa.SmartLightStrip property*), 82
hw_info (*kasa.SmartPlug property*), 63
hw_info (*kasa.SmartStrip property*), 76

I

id (*kasa.SmartBulbPreset attribute*), 58
index (*kasa.SmartBulbPreset attribute*), 58
internal_state (*kasa.SmartBulb property*), 54
internal_state (*kasa.SmartDevice property*), 29
internal_state (*kasa.SmartDimmer property*), 70
internal_state (*kasa.SmartLightStrip property*), 82
internal_state (*kasa.SmartPlug property*), 64
internal_state (*kasa.SmartStrip property*), 76
IotProtocol (*class in kasa.iotprotocol*), 44
is_bulb (*kasa.SmartBulb property*), 55
is_bulb (*kasa.SmartDevice property*), 29
is_bulb (*kasa.SmartDimmer property*), 70
is_bulb (*kasa.SmartLightStrip property*), 83
is_bulb (*kasa.SmartPlug property*), 64
is_bulb (*kasa.SmartStrip property*), 76
is_color (*kasa.SmartBulb property*), 55
is_color (*kasa.SmartDevice property*), 29
is_color (*kasa.SmartDimmer property*), 70
is_color (*kasa.SmartLightStrip property*), 83
is_color (*kasa.SmartPlug property*), 64
is_color (*kasa.SmartStrip property*), 76
is_dimmable (*kasa.SmartBulb property*), 55
is_dimmable (*kasa.SmartDevice property*), 29
is_dimmable (*kasa.SmartDimmer property*), 70
is_dimmable (*kasa.SmartLightStrip property*), 83
is_dimmable (*kasa.SmartPlug property*), 64
is_dimmable (*kasa.SmartStrip property*), 76
is_dimmer (*kasa.SmartBulb property*), 55
is_dimmer (*kasa.SmartDevice property*), 29
is_dimmer (*kasa.SmartDimmer property*), 70
is_dimmer (*kasa.SmartLightStrip property*), 83
is_dimmer (*kasa.SmartPlug property*), 64
is_dimmer (*kasa.SmartStrip property*), 76
is_light_strip (*kasa.SmartBulb property*), 55
is_light_strip (*kasa.SmartDevice property*), 29
is_light_strip (*kasa.SmartDimmer property*), 70
is_light_strip (*kasa.SmartLightStrip property*), 83
is_light_strip (*kasa.SmartPlug property*), 64
is_light_strip (*kasa.SmartStrip property*), 77
is_off (*kasa.SmartBulb property*), 55
is_off (*kasa.SmartDevice property*), 29
is_off (*kasa.SmartDimmer property*), 70
is_off (*kasa.SmartLightStrip property*), 83
is_off (*kasa.SmartPlug property*), 64
is_off (*kasa.SmartStrip property*), 77
is_on (*kasa.SmartBulb property*), 55
is_on (*kasa.SmartDevice property*), 29
is_on (*kasa.SmartDimmer property*), 70
is_on (*kasa.SmartLightStrip property*), 83
is_on (*kasa.SmartPlug property*), 64

is_on (*kasa.SmartStrip property*), 77
is_plug (*kasa.SmartBulb property*), 55
is_plug (*kasa.SmartDevice property*), 30
is_plug (*kasa.SmartDimmer property*), 70
is_plug (*kasa.SmartLightStrip property*), 83
is_plug (*kasa.SmartPlug property*), 64
is_plug (*kasa.SmartStrip property*), 77
is_strip (*kasa.SmartBulb property*), 55
is_strip (*kasa.SmartDevice property*), 30
is_strip (*kasa.SmartDimmer property*), 70
is_strip (*kasa.SmartLightStrip property*), 83
is_strip (*kasa.SmartPlug property*), 64
is_strip (*kasa.SmartStrip property*), 77
is_strip_socket (*kasa.SmartBulb property*), 55
is_strip_socket (*kasa.SmartDevice property*), 30
is_strip_socket (*kasa.SmartDimmer property*), 70
is_strip_socket (*kasa.SmartLightStrip property*), 83
is_strip_socket (*kasa.SmartPlug property*), 64
is_strip_socket (*kasa.SmartStrip property*), 77
is_variable_color_temp (*kasa.SmartBulb property*), 55
is_variable_color_temp (*kasa.SmartDevice property*), 30
is_variable_color_temp (*kasa.SmartDimmer property*), 70
is_variable_color_temp (*kasa.SmartLightStrip property*), 83
is_variable_color_temp (*kasa.SmartPlug property*), 64
is_variable_color_temp (*kasa.SmartStrip property*), 77

K

kasa
    module, 21
kasa.discover
    module, 17
kasa.modules
    module, 32
KEY_PAIR_CONTENT_LENGTH (*kasa.aestransport.AesTransport attribute*), 48
KlapTransport (*class in kasa.klaptransport*), 46
KlapTransportV2 (*class in kasa.klaptransport*), 47

L

Last (*kasa.smartbulb.BehaviorMode attribute*), 58
led (*kasa.SmartDimmer property*), 70
led (*kasa.SmartPlug property*), 64
led (*kasa.SmartStrip property*), 77
length (*kasa.SmartLightStrip property*), 83
LIGHT_SERVICE (*kasa.SmartBulb attribute*), 52
LIGHT_SERVICE (*kasa.SmartLightStrip attribute*), 80
light_state (*kasa.SmartBulb property*), 55
light_state (*kasa.SmartLightStrip property*), 83
location (*kasa.SmartBulb property*), 55
location (*kasa.SmartDevice property*), 30
location (*kasa.SmartDimmer property*), 70
location (*kasa.SmartLightStrip property*), 83
location (*kasa.SmartPlug property*), 64
location (*kasa.SmartStrip property*), 77

M

mac (*kasa.SmartBulb property*), 55
mac (*kasa.SmartDevice property*), 30
mac (*kasa.SmartDimmer property*), 70
mac (*kasa.SmartLightStrip property*), 83
mac (*kasa.SmartPlug property*), 64
mac (*kasa.SmartStrip property*), 77
max_device_response_size (*kasa.SmartBulb property*), 55
max_device_response_size (*kasa.SmartDevice property*), 30
max_device_response_size (*kasa.SmartDimmer property*), 71
max_device_response_size (*kasa.SmartLightStrip property*), 83
max_device_response_size (*kasa.SmartPlug property*), 64
max_device_response_size (*kasa.SmartStrip property*), 77
mode (*kasa.SmartBulbPreset attribute*), 58
mode (*kasa.TurnOnBehavior attribute*), 58
model (*kasa.SmartBulb property*), 55
model (*kasa.SmartDevice property*), 30
model (*kasa.SmartDimmer property*), 71
model (*kasa.SmartLightStrip property*), 83
model (*kasa.SmartPlug property*), 64
model (*kasa.SmartStrip property*), 77
module
    kasa, 21
    kasa.discover, 17
    kasa.modules, 32
modules (*kasa.SmartDimmer attribute*), 71
modules (*kasa.SmartLightStrip attribute*), 83
modules (*kasa.SmartPlug attribute*), 65
modules (*kasa.SmartStrip attribute*), 77

O

on_since (*kasa.SmartBulb property*), 55
on_since (*kasa.SmartDevice property*), 30
on_since (*kasa.SmartDimmer property*), 71
on_since (*kasa.SmartLightStrip property*), 84
on_since (*kasa.SmartPlug property*), 65
on_since (*kasa.SmartStrip property*), 77

P

password (*kasa.Credentials attribute*), 32

perform_handshake() (*kasa.aestransport.AesTransport method*), 48
perform_handshake() (*kasa.klaptransport.KlapTransport method*), 46
perform_handshake() (*kasa.klaptransport.KlapTransportV2 method*), 47
perform_handshake1() (*kasa.klaptransport.KlapTransport method*), 46
perform_handshake1() (*kasa.klaptransport.KlapTransportV2 method*), 47
perform_handshake2() (*kasa.klaptransport.KlapTransport method*), 46
perform_handshake2() (*kasa.klaptransport.KlapTransportV2 method*), 47
perform_login() (*kasa.aestransport.AesTransport method*), 48
port (*kasa.SmartBulb property*), 55
port (*kasa.SmartDevice property*), 30
port (*kasa.SmartDimmer property*), 71
port (*kasa.SmartLightStrip property*), 84
port (*kasa.SmartPlug property*), 65
port (*kasa.SmartStrip property*), 77
port_override (*kasa.DeviceConfig attribute*), 31
Preset (*kasa.smartbulb.BehaviorMode attribute*), 58
preset (*kasa.TurnOnBehavior attribute*), 59
presets (*kasa.SmartBulb property*), 56
presets (*kasa.SmartLightStrip property*), 84
protocol (*kasa.SmartDimmer attribute*), 71
protocol (*kasa.SmartLightStrip attribute*), 84
protocol (*kasa.SmartPlug attribute*), 65
protocol (*kasa.SmartStrip attribute*), 77

Q

query() (*kasa.iotprotocol.IotProtocol method*), 44
query() (*kasa.protocol.BaseProtocol method*), 44
query() (*kasa.smartprotocol.SmartProtocol method*), 45

R

reboot() (*kasa.SmartBulb method*), 56
reboot() (*kasa.SmartDevice method*), 30
reboot() (*kasa.SmartDimmer method*), 71
reboot() (*kasa.SmartLightStrip method*), 84
reboot() (*kasa.SmartPlug method*), 65
reboot() (*kasa.SmartStrip method*), 77
reset() (*kasa.aestransport.AesTransport method*), 48
reset() (*kasa.klaptransport.KlapTransport method*), 46
reset() (*kasa.klaptransport.KlapTransportV2 method*), 47
reset() (*kasa.protocol.BaseTransport method*), 45
reset() (*kasa.xortransport.XorTransport method*), 45
rssi (*kasa.SmartBulb property*), 56
rssi (*kasa.SmartDevice property*), 30
rssi (*kasa.SmartDimmer property*), 71
rssi (*kasa.SmartLightStrip property*), 84
rssi (*kasa.SmartPlug property*), 65
rssi (*kasa.SmartStrip property*), 77

S

saturation (*kasa.SmartBulbPreset attribute*), 58
save_preset() (*kasa.SmartBulb method*), 56
save_preset() (*kasa.SmartLightStrip method*), 84
send() (*kasa.aestransport.AesTransport method*), 48
send() (*kasa.klaptransport.KlapTransport method*), 46
send() (*kasa.klaptransport.KlapTransportV2 method*), 47
send() (*kasa.protocol.BaseTransport method*), 45
send() (*kasa.xortransport.XorTransport method*), 46
send_secure_passthrough() (*kasa.aestransport.AesTransport method*), 48
SESSION_COOKIE_NAME (*kasa.aestransport.AesTransport attribute*), 48
SESSION_COOKIE_NAME (*kasa.klaptransport.KlapTransport attribute*), 46
SESSION_COOKIE_NAME (*kasa.klaptransport.KlapTransportV2 attribute*), 47
set_alias() (*kasa.SmartBulb method*), 56
set_alias() (*kasa.SmartDevice method*), 30
set_alias() (*kasa.SmartDimmer method*), 71
set_alias() (*kasa.SmartLightStrip method*), 84
set_alias() (*kasa.SmartPlug method*), 65
set_alias() (*kasa.SmartStrip method*), 78
set_brightness() (*kasa.SmartBulb method*), 56
set_brightness() (*kasa.SmartDimmer method*), 71
set_brightness() (*kasa.SmartLightStrip method*), 84
set_button_action() (*kasa.SmartDimmer method*), 71
set_color_temp() (*kasa.SmartBulb method*), 56
set_color_temp() (*kasa.SmartLightStrip method*), 84
set_custom_effect() (*kasa.SmartLightStrip method*), 84
set_dimmer_transition() (*kasa.SmartDimmer method*), 71
set_effect() (*kasa.SmartLightStrip method*), 84
set_fade_time() (*kasa.SmartDimmer method*), 71
set_hsv() (*kasa.SmartBulb method*), 56
set_hsv() (*kasa.SmartLightStrip method*), 85
set_led() (*kasa.SmartDimmer method*), 71
set_led() (*kasa.SmartPlug method*), 65
set_led() (*kasa.SmartStrip method*), 78
SET_LIGHT_METHOD (*kasa.SmartBulb attribute*), 52

SET_LIGHT_METHOD (*kasa.SmartLightStrip attribute*), 80
set_light_state() (*kasa.SmartBulb method*), 56
set_light_state() (*kasa.SmartLightStrip method*), 85
set_mac() (*kasa.SmartBulb method*), 56
set_mac() (*kasa.SmartDevice method*), 30
set_mac() (*kasa.SmartDimmer method*), 72
set_mac() (*kasa.SmartLightStrip method*), 85
set_mac() (*kasa.SmartPlug method*), 65
set_mac() (*kasa.SmartStrip method*), 78
set_turn_on_behavior() (*kasa.SmartBulb method*), 57
set_turn_on_behavior() (*kasa.SmartLightStrip method*), 85
SmartBulb (*class in kasa*), 50
SmartBulbPreset (*class in kasa*), 58
SmartDevice (*class in kasa*), 25
SmartDeviceException (*class in kasa*), 32
SmartDimmer (*class in kasa*), 67
SmartLightStrip (*class in kasa*), 79
SmartPlug (*class in kasa*), 61
SmartProtocol (*class in kasa.smartprotocol*), 44
SmartStrip (*class in kasa*), 73
soft (*kasa.TurnOnBehaviors attribute*), 58
state_information (*kasa.SmartBulb property*), 57
state_information (*kasa.SmartDevice property*), 30
state_information (*kasa.SmartDimmer property*), 72
state_information (*kasa.SmartLightStrip property*), 85
state_information (*kasa.SmartPlug property*), 65
state_information (*kasa.SmartStrip property*), 78
supported_modules (*kasa.SmartBulb property*), 57
supported_modules (*kasa.SmartDevice property*), 30
supported_modules (*kasa.SmartDimmer property*), 72
supported_modules (*kasa.SmartLightStrip property*), 85
supported_modules (*kasa.SmartPlug property*), 65
supported_modules (*kasa.SmartStrip property*), 78
sys_info (*kasa.SmartBulb property*), 57
sys_info (*kasa.SmartDevice property*), 30
sys_info (*kasa.SmartDimmer property*), 72
sys_info (*kasa.SmartLightStrip property*), 85
sys_info (*kasa.SmartPlug property*), 65
sys_info (*kasa.SmartStrip property*), 78

T

time (*kasa.SmartBulb property*), 57
time (*kasa.SmartDevice property*), 31
time (*kasa.SmartDimmer property*), 72
time (*kasa.SmartLightStrip property*), 85
time (*kasa.SmartPlug property*), 65
time (*kasa.SmartStrip property*), 78
timeout (*kasa.DeviceConfig attribute*), 31
TIMEOUT_COOKIE_NAME (*kasa.aestransport.AesTransport attribute*), 48
TIMEOUT_COOKIE_NAME (*kasa.klaptransport.KlapTransport attribute*), 46
TIMEOUT_COOKIE_NAME (*kasa.klaptransport.KlapTransportV2 attribute*), 47
timezone (*kasa.SmartBulb property*), 57
timezone (*kasa.SmartDevice property*), 31
timezone (*kasa.SmartDimmer property*), 72
timezone (*kasa.SmartLightStrip property*), 85
timezone (*kasa.SmartPlug property*), 65
timezone (*kasa.SmartStrip property*), 78
to_dict() (*kasa.DeviceConfig method*), 32
try_login() (*kasa.aestransport.AesTransport method*), 48
turn_off() (*kasa.SmartBulb method*), 57
turn_off() (*kasa.SmartDevice method*), 31
turn_off() (*kasa.SmartDimmer method*), 72
turn_off() (*kasa.SmartLightStrip method*), 85
turn_off() (*kasa.SmartPlug method*), 65
turn_off() (*kasa.SmartStrip method*), 78
turn_on() (*kasa.SmartBulb method*), 57
turn_on() (*kasa.SmartDevice method*), 31
turn_on() (*kasa.SmartDimmer method*), 72
turn_on() (*kasa.SmartLightStrip method*), 85
turn_on() (*kasa.SmartPlug method*), 65
turn_on() (*kasa.SmartStrip method*), 78
TurnOnBehavior (*class in kasa*), 58
TurnOnBehavior.Config (*class in kasa*), 58
TurnOnBehaviors (*class in kasa*), 58

U

UnsupportedDeviceException (*class in kasa*), 32
update() (*kasa.SmartBulb method*), 57
update() (*kasa.SmartDevice method*), 31
update() (*kasa.SmartDimmer method*), 72
update() (*kasa.SmartLightStrip method*), 86
update() (*kasa.SmartPlug method*), 65
update() (*kasa.SmartStrip method*), 78
update_from_discover_info() (*kasa.SmartBulb method*), 57
update_from_discover_info() (*kasa.SmartDevice method*), 31
update_from_discover_info() (*kasa.SmartDimmer method*), 72
update_from_discover_info() (*kasa.SmartLightStrip method*), 86
update_from_discover_info() (*kasa.SmartPlug method*), 66
update_from_discover_info() (*kasa.SmartStrip method*), 78
username (*kasa.Credentials attribute*), 32
uses_http (*kasa.DeviceConfig attribute*), 32

V

valid_temperature_range (*kasa.SmartBulb property*), 57
valid_temperature_range (*kasa.SmartLightStrip property*), 86
validate_assignment (*kasa.TurnOnBehavior.Config attribute*), 58

W

wifi_join() (*kasa.SmartBulb method*), 57
wifi_join() (*kasa.SmartDevice method*), 31
wifi_join() (*kasa.SmartDimmer method*), 72
wifi_join() (*kasa.SmartLightStrip method*), 86
wifi_join() (*kasa.SmartPlug method*), 66
wifi_join() (*kasa.SmartStrip method*), 78
wifi_scan() (*kasa.SmartBulb method*), 57
wifi_scan() (*kasa.SmartDevice method*), 31
wifi_scan() (*kasa.SmartDimmer method*), 72
wifi_scan() (*kasa.SmartLightStrip method*), 86
wifi_scan() (*kasa.SmartPlug method*), 66
wifi_scan() (*kasa.SmartStrip method*), 78

X

XorTransport (*class in kasa.xortransport*), 45