python-yeelight Documentation

Release 0.3.2

Stavros Korokithakis

Jun 21, 2017

Contents

The YeeLight Python library is a small library that lets you control your YeeLight RGB LED bulbs over WiFi. The latest version can be found at:

https://gitlab.com/stavros/python-yeelight

To see a real-world usage example, have a look at yeecli, a command-line YeeLight utility that uses this library.

`yeelight` currently does not support some features of the YeeLight API, such as discovery, but is mostly otherwise complete.

CHAPTER 1

Installation

To install `yeelight`, you can use `pip`:

```
pip install yeelight
```

That's all that's required to install the library.

CHAPTER 2

Usage

First of all, you need to discover your bulb's IP. If you already know it, you can skip to the next section.

Discovering all the devices on your network and their capabilities is easy with *discover_bulbs*:

```
>>> from yeelight import discover_bulbs
>>> discover_bulbs()
[{'capabilities': {'bright': '50',
                   'color_mode': '1',
                   'ct': '2700',
                   'fw_ver': '45',
                   'hue': '359',
                   'id': '0x0000000002dfb19a',
                   'model': 'color',
                   'name': 'bedroom',
                   'power': 'off',
                   'rgb': '16711935',
                   'sat': '100',
                   'support': 'get_prop set_default set_power toggle '
                              'set_bright start_cf stop_cf set_scene cron_add '
                              'cron_get cron_del set_ct_abx set_rgb set_hsv '
                              'set_adjust set_music set_name'},
  'ip': '192.168.0.19',
  'port': 55443},
  {'capabilities': {'bright': '50',
                    'color_mode': '1',
                    'ct': '2700',
                    'fw_ver': '45',
                    'hue': '359',
                    'id': '0x0000000002dfb2f1',
                    'model': 'color',
                    'name': 'livingroom',
                    'power': 'off',
                    'rgb': '16711935',
                    'sat': '100',
                    'support': 'get_prop set_default set_power toggle '
```

```
                              'set_bright start_cf stop_cf set_scene cron_add '
                              'cron_get cron_del set_ct_abx set_rgb set_hsv '
                              'set_adjust set_music set_name'},
  'ip': '192.168.0.23',
  'port': 55443}]
```

That's it, now you know the addresses of all the bulbs on your local network.

Now that you've discovered your bulb's IP, it's time to instantiate a new `Bulb`:

```
>>> from yeelight import Bulb
>>> bulb = Bulb("192.168.0.19")

# Turn the bulb on.
>>> bulb.turn_on()

# Turn the bulb off.
>>> bulb.turn_off()

# Toggle power.
>>> bulb.toggle()

# Set brightness to 50%.
>>> bulb.set_brightness(50)

# Set RGB value.
>>> bulb.set_rgb(255, 0, 0)

# Set HSV value.
>>> bulb.set_hsv(320, 100, 50)

# Set hue and saturation, but keep value (brightness) the same.
>>> bulb.set_hsv(320, 100)

# Set color temperature.
>>> bulb.set_color_temp(4700)

# Save this setting as default.
>>> bulb.set_default()
```

For efficiency, `yeelight` will use a single TCP connection for all the above commands. However, this means that, if there's an error, a command could raise a `socket.error` exception and need to be retried. Note that YeeLight connections are rate-limited to 60 per minute. If you need your connection to not have a limit, you need to use *Music mode*.

For a complete list of the commands you can issue, see the *API reference*.

By default, `yeelight` will refuse to make any changes to the bulb if it's off:

```
>>> bulb.set_brightness(10)
AssertionError: Commands have no effect when the bulb is off.
```

You can check the bulb's state by reading its properties:

```
>>> bulb.get_properties
{'bright': u'10',
 'color_mode': u'2',
 'ct': u'2700',
 'delayoff': u'0',
```

```
 'flow_params': u'',
 'flowing': u'0',
 'hue': u'300',
 'music_on': u'0',
 'name': u'My light',
 'power': u'off',
 'rgb': u'16737280',
 'sat': u'100'}
```

If you want to always turn the bulb on before running a command, set `auto_on` to `True`. This will refresh the bulb's properties before most calls, and will cost you an extra message per command, so watch out for rate-limiting:

```
>>> bulb.auto_on = True

# Or, when instantiating:
>>> bulb = Bulb("192.168.0.19", auto_on=True)

# This will work even if the bulb is off.
>>> bulb.set_brightness(10)
```

For documentation of the Flow feature, see *Working with Flow*.

CHAPTER 3

Effects

`yeelight` provides full support for effects. Effects control whether the bulb changes from one state to the other immediately or gradually, and how long the gradual change takes.

You can either specify effects to run by default when instantiating, or with each call:

```
>>> bulb = Bulb("192.168.0.19", effect="smooth", duration=1000)

# This will turn the bulb on gradually within one second:
>>> bulb.turn_on()

# This will turn the bulb on immediately:
>>> bulb.turn_on(effect="sudden")

# You can easily change the default effect, too:
>>> bulb.effect = "sudden"

# This will turn the bulb off immediately:
>>> bulb.turn_on()
```

There are two effect types, `"sudden"` and `"smooth"`. The `"sudden"` type ignores the `duration` parameter.

Keep in mind that the `effect` and `duration` parameters *must* be passed by keyword.

Working with Flow

"Flow" is a special mode the bulb can be set to, which is basically a list of transitions to perform in succession. For example, a flow can be a constant cycling of colors from one to the next, until it is stopped, or it can be a quick blink of a certain color.

Usage

To create a flow, we need to instantiate a *Flow* object and add some transitions to it. The *Flow* object accepts three parameters, a `count`, which is the number of *loops* to perform (*NOT* the number of transitions!), an `action`, which is what to do at the end of the flow, and `transitions`, a list of the transitions themselves.

In the original protocol spec, the `count` argument specifies the number of transitions to perform. For example, if you have 10 transitions, and `count` is 5, the bulb will stop transitioning in the middle of your flow and exit. This is rather counter-intuitive, so the `yeelight` library changes this to mean "number of loops". If you want to loop once, no matter how long your transition list, just specify 1. 0 means "loop forever".

`action` can be `Flow.actions.recover` to return to the original state after this flow is stopped (either through the *stop_flow* method or just because the loop has stopped), `Flow.actions.stay` to just stay at the last state of the flow, or `Flow.actions.off` to turn off afterwards.

`transitions` is a list of transition instances. There are various transition classes available, which are detailed in the *API reference*. The bulbs seem to be limited to around nine transitions (any more will produce an "invalid command" error).

Let's see a few examples.

Simple example

A simple example is to cycle the color temperature from 1700 to 6500 twice, with a one-second delay in-between (the examples assume we have a *Bulb* instance in `bulb`), and then return to the previous state:

```
from yeelight import *
transitions = [
    TemperatureTransition(1700, duration=1000),
    SleepTransition(duration=1000),
    TemperatureTransition(6500, duration=1000)
]

flow = Flow(
    count=2,
    action=Flow.actions.recover,
    transitions=transitions
)

bulb.start_flow(flow)
```

Infinite color cycle

We can cycle between colors forever like so:

```
from yeelight import *
transitions = [
    RGBTransition(255, 0, 255, duration=1000)
]

flow = Flow(
    count=0,  # Cycle forever.
    transitions=transitions
)

bulb.start_flow(flow)
```

To stop the flow (and return to the previous state, because of the default `Flow.actions.recover` action, you can use *`stop_flow`*:

```
bulb.stop_flow()
```

Quick pulse

If you want to connect the bulb to a notification system, you can fire off a quick pulse. For example, to pulse the bulb green twice when there is a WhatsApp message and return, we can do:

```
from yeelight import *
transitions = [HSVTransition(hue, 100, duration=500)
               for hue in range(0, 359, 40)]

flow = Flow(
    count=2,
    transitions=transitions
)

bulb.start_flow(flow)
```

Pretty easy!

Transition presets

The library includes some preset transitions in the *`yeelight.transitions`* module, to make it easy for you to start.

You can use the transitions simply by calling the preset:

```
from yeelight.transitions import *
from yeelight import Flow

flow = Flow(
    count=10,
    transitions=disco(),  # Call the transition preset to get the
                          # transitions you like.
)

bulb.start_flow(flow)
```

Remember that the transition presets are functions, so you need to call them. That's because some of them take parameters.

yeelight package

This is the autogenerated API documentation. Use it as a reference to the public API of the project.

yeelight module

`yeelight.`**`discover_bulbs`**(*timeout=2*)
Discover all the bulbs in the local network.

Parameters **timeout** (*int*) – How many seconds to wait for replies. Discovery will always take exactly this long to run, as it can't know when all the bulbs have finished responding.

Returns A list of dictionaries, containing the ip, port and capabilities of each of the bulbs in the network.

class yeelight.**Bulb**(*ip*, *port=55443*, *effect='smooth'*, *duration=300*, *auto_on=False*)

The main controller class of a physical YeeLight bulb.

Parameters

- **ip** (*str*) – The IP of the bulb.
- **port** (*int*) – The port to connect to on the bulb.
- **effect** (*str*) – The type of effect. Can be "smooth" or "sudden".
- **duration** (*int*) – The duration of the effect, in milliseconds. The minimum is 30. This is ignored for sudden effects.
- **auto_on** (*bool*) – Whether to call *ensure_on()* to turn the bulb on automatically before each operation, if it is off. This renews the properties of the bulb before each message, costing you one extra message per command. Turn this off and do your own checking with *get_properties()* or run *ensure_on()* yourself if you're worried about rate-limiting.

bulb_type

The type of bulb we're communicating with.

Returns a *BulbType* describing the bulb type. This can either be *Color* or *White*.

When trying to access before properties are known, the bulb type is unknown.

Return type *yeelight.BulbType*

Returns The bulb's type.

cron_add(*event_type*, *value*)

Add an event to cron.

Example:

```
>>> bulb.cron_add(CronType.off, 10)
```

Parameters **event_type** (yeelight.enums.CronType) – The type of event. Currently, only CronType.off.

cron_del(*event_type*)

Remove an event from cron.

Parameters **event_type** (yeelight.enums.CronType) – The type of event. Currently, only CronType.off.

cron_get(*event_type*)

Retrieve an event from cron.

Parameters **event_type** (yeelight.enums.CronType) – The type of event. Currently, only CronType.off.

ensure_on()

Turn the bulb on if it is off.

get_properties()
Retrieve and return the properties of the bulb.

This method also updates `last_properties` when it is called.

Returns A dictionary of param: value items.

Return type dict

last_properties
The last properties we've seen the bulb have.

This might potentially be out of date, as there's no background listener for the bulb's notifications. To update it, call *get_properties*.

music_mode
Return whether the music mode is active.

Return type bool

Returns True if music mode is on, False otherwise.

send_command(*method*, *params=None*)
Send a command to the bulb.

Parameters

- **method** (*str*) – The name of the method to send.
- **params** (*list*) – The list of parameters for the method.

Raises ***BulbException*** – When the bulb indicates an error condition.

Returns The response from the bulb.

set_adjust(*action*, *prop*, ***kwargs*)
Adjust a parameter.

I don't know what this is good for. I don't know how to use it, or why. I'm just including it here for completeness, and because it was easy, but it won't get any particular love.

Parameters

- **action** (*str*) – The direction of adjustment. Can be "increase", "decrease" or "circle".
- **prop** (*str*) – The property to adjust. Can be "bright" for brightness, "ct" for color temperature and "color" for color. The only action for "color" can be "circle". Why? Who knows.

set_brightness(*brightness*, ***kwargs*)
Set the bulb's brightness.

Parameters **brightness** (*int*) – The brightness value to set (1-100).

set_color_temp(*degrees*, ***kwargs*)
Set the bulb's color temperature.

Parameters **degrees** (*int*) – The degrees to set the color temperature to (1700-6500).

set_default()
Set the bulb's current state as default.

set_hsv(*hue*, *saturation*, *value=None*, ***kwargs*)
Set the bulb's HSV value.

Parameters

- **hue** (*int*) – The hue to set (0-359).
- **saturation** (*int*) – The saturation to set (0-100).
- **value** (*int*) – The value to set (0-100). If omitted, the bulb's brightness will remain the same as before the change.

set_name(*name*)
Set the bulb's name.

Parameters **name** (*str*) – The string you want to set as the bulb's name.

set_rgb(*red*, *green*, *blue*, ***kwargs*)
Set the bulb's RGB value.

Parameters

- **red** (*int*) – The red value to set (0-255).
- **green** (*int*) – The green value to set (0-255).
- **blue** (*int*) – The blue value to set (0-255).

start_flow(*flow*)
Start a flow.

Parameters **flow** (yeelight.Flow) – The Flow instance to start.

start_music(*port=0*)
Start music mode.

Music mode essentially upgrades the existing connection to a reverse one (the bulb connects to the library), removing all limits and allowing you to send commands without being rate-limited.

Starting music mode will start a new listening socket, tell the bulb to connect to that, and then close the old connection. If the bulb cannot connect to the host machine for any reason, bad things will happen (such as library freezes).

Parameters **port** (*int*) – The port to listen on. If none is specified, a random port will be chosen.

stop_flow()
Stop a flow.

stop_music()
Stop music mode.

Stopping music mode will close the previous connection. Calling `stop_music` more than once, or while not in music mode, is safe.

toggle(***kwargs*)
Toggle the bulb on or off.

turn_off(***kwargs*)
Turn the bulb off.

turn_on(***kwargs*)
Turn the bulb on.

class yeelight.**BulbException**
A generic yeelight exception.

This exception is raised when bulb informs about errors, e.g., when trying to issue unsupported commands to the bulb.

class yeelight.**BulbType**
The bulb's type.

This is either `White` or `Color`, or `Unknown` if the properties have not been fetched yet.

Color = 1

Unknown = -1

White = 0

Flow objects

class yeelight.**Flow**(*count=0*, *action=<Action.recover: 0>*, *transitions=None*)
A complete flow, consisting of one or multiple transitions.

Example:

```
>>> transitions = [RGBTransition(255, 0, 0), SleepTransition(400)]
>>> Flow(3, Flow.actions.recover, transitions)
```

Parameters

- **count** (*int*) – The number of times to run this flow (0 to run forever).
- **action** (*action*) – The action to take after the flow stops. Can be `Flow.actions.recover` to go back to the state before the flow, `Flow.actions.stay` to stay at the last state, and `Flow.actions.off` to turn off.
- **transitions** (*list*) – A list of `FlowTransition` instances that describe the flow transitions to perform.

actions
alias of `Action`

expression
Return a YeeLight-compatible expression that implements this flow.

Return type list

class yeelight.**HSVTransition**(*hue*, *saturation*, *duration=300*, *brightness=100*)
An HSV transition.

Parameters

- **hue** (*int*) – The color hue to transition to (0-359).
- **saturation** (*int*) – The color saturation to transition to (0-100).
- **duration** (*int*) – The duration of the effect, in milliseconds. The minimum is 50.
- **brightness** (*int*) – The brightness value to transition to (1-100).

class yeelight.**RGBTransition**(*red*, *green*, *blue*, *duration=300*, *brightness=100*)
An RGB transition.

Parameters

- **red** (*int*) – The value of red (0-255).
- **green** (*int*) – The value of green (0-255).
- **blue** (*int*) – The value of blue (0-255).

- **duration** (*int*) – The duration of the effect, in milliseconds. The minimum is 50.
- **brightness** (*int*) – The brightness value to transition to (1-100).

class yeelight.**TemperatureTransition**(*degrees*, *duration=300*, *brightness=100*)
A Color Temperature transition.

Parameters

- **degrees** (*int*) – The degrees to set the color temperature to (1700-6500).
- **duration** (*int*) – The duration of the effect, in milliseconds. The minimum is 50.
- **brightness** (*int*) – The brightness value to transition to (1-100).

class yeelight.**SleepTransition**(*duration=300*)
A Sleep transition.

Parameters **duration** (*int*) – The duration of the effect, in milliseconds. The minimum is 50.

Transition presets

Pre-made transitions, for your strobing pleasure.

yeelight.transitions.**alarm**(*duration=250*)
Red alarm; flashing bright red to dark red.

Parameters **duration** (*int*) – The duration between hi/lo brightness,in milliseconds.

Returns A list of transitions.

Return type list

yeelight.transitions.**christmas**(*duration=250*, *brightness=100*, *sleep=3000*)
Color changes from red to green, like christmas lights.

Parameters

- **duration** (*int*) – The duration between red and green, in milliseconds.
- **brightness** (*int*) – The brightness of the transition.
- **sleep** (*int*) – The time to sleep between colors, in milliseconds.

Returns A list of transitions.

Return type list

yeelight.transitions.**disco**(*bpm=120*)
Color changes to the beat.

Parameters **bpm** (*int*) – The beats per minute to pulse to.

Returns A list of transitions.

Return type list

yeelight.transitions.**lsd**(*duration=3000*, *brightness=100*)
Gradual changes to a pleasing, trippy palette.

Parameters **brightness** (*int*) – The brightness of the transition.

Returns A list of transitions.

Return type list

yeelight.transitions.**police**(*duration=300*, *brightness=100*)
Color changes from red to blue, like police lights.

Parameters

- **duration** (*int*) – The duration between red and blue, in milliseconds.
- **brightness** (*int*) – The brightness of the transition.

Returns A list of transitions.

Return type list

yeelight.transitions.**police2**(*duration=250*, *brightness=100*)
Color flashes red and then blue, like urgent police lights.

Parameters

- **duration** (*int*) – The duration to fade to next color, in milliseconds.
- **brightness** (*int*) – The brightness of the transition.

Returns A list of transitions.

Return type list

yeelight.transitions.**pulse**(*red*, *green*, *blue*, *duration=250*, *brightness=100*)
Pulse a single color once (mainly to be used for notifications).

Parameters

- **red** (*int*) – The red color component to pulse (0-255).
- **green** (*int*) – The green color component to pulse (0-255).
- **blue** (*int*) – The blue color component to pulse (0-255).
- **duration** (*int*) – The duration to pulse for, in milliseconds.
- **brightness** (*int*) – The brightness to pulse at (1-100).

Returns A list of transitions.

Return type list

yeelight.transitions.**randomloop**(*duration=750*, *brightness=100*, *count=9*)
Color changes between `count` randomly chosen colors.

Parameters

- **duration** (*int*) – The duration to fade to next color, in milliseconds.
- **brightness** (*int*) – The brightness of the transition.
- **count** (*int*) – The number of random chosen colors in transition.

Returns A list of transitions.

Return type list

yeelight.transitions.**rgb**(*duration=250*, *brightness=100*, *sleep=3000*)
Color changes from red to green to blue.

Parameters

- **duration** (*int*) – The duration to fade to next color, in milliseconds.
- **brightness** (*int*) – The brightness of the transition.
- **sleep** (*int*) – The time to sleep between colors, in milliseconds

Returns A list of transitions.

Return type list

yeelight.transitions.**slowdown**(*duration=2000*, *brightness=100*, *count=8*)
Changes between `count` random chosen colors with increasing transition time.

Parameters

- **duration** (*int*) – The duration to fade to next color, in milliseconds.
- **brightness** (*int*) – The brightness of the transition.
- **count** (*int*) – The number of random chosen colors in transition.

Returns A list of transitions.

Return type list

yeelight.transitions.**strobe**()
Rapid flashing on and off.

Returns A list of transitions.

Return type list

yeelight.transitions.**strobe_color**(*brightness=100*)
Rapid flashing colors.

Parameters **brightness** (*int*) – The brightness of the transition.

Returns A list of transitions.

Return type list

yeelight.transitions.**temp**()
Slowly-changing color temperature.

Returns A list of transitions.

Return type list

Enums

class yeelight.enums.**CronType**
Bases: `enum.Enum`

The type of event in cron.

off = 0

Python Module Index

y

Index

U

W

Y