Smart Bulb Interaction Documentation

Author: Tyler Terry

TinyTuya API developer: Jason Cox

Github Respiratory: https://github.com/jasonacox/tinytuya

Table of Contents

Introduction

This document provides instructions on how to use the TinyTuya library to control and monitor Tuya-based smart devices, such as plugs, switches, lights, and window covers, using Python. The TinyTuya library allows you to communicate with these devices over the local area network (LAN) or the TuyaCloud API.

Hardware Used

- Merkury Innovations A21 Smart Color Light Bulb | 4 Bulb Pack ($19.88)

Technologies Used

- TinyTuya library
- Python interpreter (version 2.7 or 3.x recommended)
- Tuya-based smart devices (plugs, switches, lights, window covers)
- Smart Life or Tuya Smart app for iPhone or Android
- Tuya developer account at iot.tuya.com

Requirements

- A Python interpreter (version 2.7 or 3.x recommended)
- The TinyTuya library, which can be installed using pip install tinytuya
- The required dependencies pycryptodome, requests, and colorama, which will be installed automatically when you install tinytuya
- A Tuya-based smart device that is activated and connected to your home network
- The device ID, address (IPv4), and local key of the Tuya device you want to control
- A Tuya developer account at iot.tuya.com and the authorization key (API ID and secret) and data center associated with your Tuya cloud project

Installation & Initialization

1. Install the tinytuya library by running pip install tinytuya. This will also install the required dependencies pycryptodome, requests, and colorama if they are not already installed.

```
pip install tinytuya
```

2. Activate your Tuya-based smart devices by using the Smart Life or Tuya Smart app on your iPhone or Android device. This is necessary to make the devices accessible through the Tuya Cloud.

3. Use the built-in network scanner in tinytuya to find the Tuya devices on your local network. Run the command python -m tinytuya scan to see a list of devices with their addresses, device IDs, and versions.

```
python -m tinytuya scan
```

4. Set up a Tuya developer account at iot.tuya.com and log in. Create a new cloud project, and skip the configuration wizard. Remember the authorization key (API ID and secret) and the data center that you select.

5. In the Tuya cloud project, go to the Devices tab and link your Tuya app account by clicking on the "Add App Account" button and scanning the QR code with the Smart Life app on your phone. This will link all of the devices registered in your Smart Life app to your Tuya IoT project.

6. Run the built-in setup wizard in tinytuya to generate a JSON list of all your registered devices, including their secret local keys and names. Run the command python -m tinytuya wizard and follow the prompts to enter your Tuya developer account details and the data center you selected when creating your cloud project. This will create a devices.json file with the information about your devices.

```
python -m tinytuya wizard
```

7. In your Python code, import the tinytuya library and create a Device object, passing in the device ID, address, and local key of the Tuya device you want to control. You can use the set_version method to specify the protocol version of the device (3.1, 3.2, 3.3, or 3.4).

TinyTuya Module Classes and Functions

Global Functions

- devices = deviceScan(): Returns a dictionary of devices found on the local network
- scan(): Interactive scan of the local network
- wizard(): Interactive setup wizard
- set_debug(toggle, color): Activate verbose debugging output

Classes

- OutletDevice(dev_id, address, local_key=None, dev_type='default')
- CoverDevice(dev_id, address, local_key=None, dev_type='default')
- BulbDevice(dev_id, address, local_key=None, dev_type='default')
    - dev_id (str): Device ID e.g. 01234567891234567890
    - address (str): Device Network IP Address e.g. 10.0.1.99 or 0.0.0.0 to auto-find
    - local_key (str, optional): The encryption key. Defaults to None.
    - dev_type (str): Device type for payload options (see below)
- Cloud(apiRegion, apiKey, apiSecret, apiDeviceID, new_sign_algorithm)

Functions

Configuration Settings

- set_version(version): Set the device version to 3.1 [default] or 3.3 (all new devices)
- set_socketPersistent(False/True): Keep the connection open with the device: False [default] or True
- set_socketNODELAY(False/True): Add a cooldown period for slow Tuya devices: False or True [default]
- set_socketRetryLimit(integer): Set the retry count limit [default 5]
- set_socketTimeout(s): Set the connection timeout in seconds [default 5]
- set_dpsUsed(dpsUsed): Set the data points (DPs) to expect (rarely needed)
- set_retry(retry=True): Force a retry if the response payload is truncated
- set_sendWait(num_secs): Set the number of seconds to wait after sending for a response
- set_bulb_type(type): For BulbDevice, set the type to A, B, or C

Device Commands

- status(): Fetch the status of the device (returns a JSON payload)
- detect_available_dps(): Return a list of DPS available from the device
- set_status(on, switch=1, nowait): Control the status of the device to 'on' or 'off' (bool)
    - nowait (default False): Set to True to send the command without waiting for a response
- set_value(index, value, nowait): Send and set the value of any DPS/index on the device
- heartbeat(nowait): Send a heartbeat to the device
- updatedps(index=[1], nowait): Send the updatedps command to the device to refresh DPS values
- turn_on(switch=1, nowait): Turn on the device/switch #
- turn_off(switch=1, nowait): Turn off the device
- set_timer(num_secs, nowait): Set a timer for num_secs seconds on devices (if supported)
- generate_payload(command, data): Generate a TuyaMessage payload for a command with data
- send(payload): Send a payload to the device (does not wait for a response)
- receive(): Receive a payload from the device

OutletDevice Functions

- set_dimmer(percentage): Set the dimmer value for the device (0-100%)

CoverDevice Functions

- open_cover(switch=1): Open the cover
- close_cover(switch=1): Close the cover
- stop_cover(switch=1): Stop the cover

BulbDevice Functions

- set_colour(r, g, b, nowait): Set the colour of the bulb in RGB values
- set_hsv(h, s, v, nowait): Set the colour of the bulb in HSV values
- set_white(brightness, colourtemp, nowait): Set the brightness and colour temperature of the white mode
- set_white_percentage(brightness=100, colourtemp=0, nowait): Set the brightness and colour temperature of the white mode as a percentage
- set_brightness(brightness, nowait): Set the brightness of the bulb
- set_brightness_percentage(brightness=100, nowait): Set the brightness of the bulb as a percentage

- set_colourtemp(colourtemp, nowait): Set the colour temperature of the bulb
- set_colourtemp_percentage(colourtemp=100, nowait): Set the colour temperature of the bulb as a percentage
- set_scene(scene, nowait): Set the scene mode of the bulb
    - scene: 1=nature, 3=rave, 4=rainbow
- set_mode(mode='white', nowait): Set the mode of the bulb
    - mode: white, colour, scene, music, etc.
- result = brightness(): Get the brightness of the bulb
- result = colourtemp(): Get the colour temperature of the bulb
- (r, g, b) = colour_rgb(): Get the colour of the bulb in RGB values
- (h,s,v) = colour_hsv(): Get the colour of the bulb in HSV values
- result = state(): Get the state of the bulb (on or off)

Cloud

- setregion(apiRegion): Set the API region for the Cloud connection
- cloudrequest(url, action=[POST if post else GET], post={}, query={}): Make a request to the Tuya Cloud API
- getdevices(verbose=False): Get a list of devices registered to the Tuya Cloud API
- getstatus(deviceid): Get the status of a device
- getfunctions(deviceid): Get the functions of a device
- getproperties(deviceid): Get the properties of a device
- getdps(deviceid): Get the data points (DPs) of a device
- sendcommand(deviceid, commands): Send a command to a device
- getconnectstatus(deviceid): Get the connection status of a device
- getdevicelog(deviceid, start=[now - 1 day], end=[now], evtype="1,2,3,4,5,6,7,8,9,10", size=0, max_fetches=50, start_row_key=None, params={}): Get the device log for a device

Example Usage

```
import tinytuya

"""
OUTLET Device
"""
d = tinytuya.OutletDevice('DEVICE_ID_HERE', 'IP_ADDRESS_HERE', 'LOCAL_KEY_HERE')
d.set_version(3.3)
data = d.status()

# Show status and state of first controlled switch on device
print('Dictionary %r' % data)
print('State (bool, true is ON) %r' % data['dps']['1'])

# Toggle switch state
switch_state = data['dps']['1']
data = d.set_status(not switch_state)  # This requires a valid key
if data:
        print('set_status() result %r' % data)

# On a switch that has 4 controllable ports, turn the fourth OFF (1 is the first)
data = d.set_status(False, 4)
if data:
        print('set_status() result %r' % data)
        print('set_status() extra %r' % data[20:-8])

"""
RGB Bulb Device
"""
import time

d = tinytuya.BulbDevice('DEVICE_ID_HERE', 'IP_ADDRESS_HERE', 'LOCAL_KEY_HERE')
d.set_version(3.3)  # IMPORTANT to set this regardless of version
d.set_socketPersistent(True)  # Optional: Keep socket open for multiple commands
data = d.status()

# Show status of first controlled switch on device
print('Dictionary %r' % data)

# Set to RED Color - set_colour(r, g, b):
d.set_colour(255,0,0)
```

```
# Cycle through the Rainbow
rainbow = {"red": [255, 0, 0], "orange": [255, 127, 0], "yellow": [255, 200, 0],
        "green": [0, 255, 0], "blue": [0, 0, 255], "indigo": [46, 43, 95],
        "violet": [139, 0, 255]}
for color in rainbow:
        [r, g, b] = rainbow[color]
        d.set_colour(r, g, b, nowait=True)  # nowait = Go fast don't wait for response
        time.sleep(0.25)

# Brightness: Type A devices range = 25-255 and Type B = 10-1000
d.set_brightness(1000)

# Set to White - set_white(brightness, colourtemp):
#        colourtemp: Type A devices range = 0-255 and Type B = 0-1000
d.set_white(1000,10)

# Set Bulb to Scene Mode
d.set_mode('scene')

# Scene Example: Set Color Rotation Scene
d.set_value(25,
'07464602000003e803e8000000004646020078 03e803e8000000004646020 0f003e803e800000000464
602003d03e803e8000000004646020 0ae03e803e8000000004646020 11303e803e800000000')
```