Table des matières

Tuya / SmartLife Cloud Service

Instruction API

Pour activer l'appel via API avec vos devices SmartLife il vous faudra suivre cette procédure.

tuya.iot.api.setup.1.pdf

TinyTuya

Voici les instructions pour installer les addon python pour manipuler vos device via des script python. Il vous sera possible de le faire via le CLou ou en mode local.

Installation

```
python -m pip install tinytuya
```

Network Scanner

Vous permet de scanner vos devices local.

```
python -m tinytuya scan
```

Command line tools

```
# Listen for Tuya Devices and match to devices.json if available
python -m tinytuya scan

# The above creates a snapshot.json file with IP addresses for devices
# You can use this command to get a rapid poll of status of all devices
python -m tinytuya snapshot
```

```
# The sames thing as above but with a non-interactive JSON response
python -m tinytuya json

# List all register devices discovered from Wizard and poll them
python -m tinytuya devices
```

Wizard Cloud

Vous permettra de rapatrier vos devices depuis le CLOUD Tuya.

```
python -m tinytuya wizard
```

L'outils vous demandera ces éléments pour se connecter:

```
    "apiKey": "xxxxxxxxxxxxxxxxxxxx",
    "apiSecret": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
    "apiRegion": "eu",
    "apiDeviceID": "scan"
```

A la fin de Wizard, celui-ci enregistrera ces infos dans un fichier tinytuya.json à l'endroit ou le script à été exécuté

Il générera aussi un ficher contenant toutes les infos de vos devices enregistré dans le Cloud

```
tuya-raw.json
```

Script Cloud

voici un exemple de script pour établir une connexion via le Cloud et afficher les valeurs d'un équipement définit.

```
import tinytuya

# Connect to Tuya Cloud
# c = tinytuya.Cloud()  # uses tinytuya.json
c = tinytuya.Cloud(
        apiRegion="us",
        apiKey="xxxxxxxxxxxxxxxxxxx",
        apiSecret="xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
        apiDeviceID="xxxxxxxxxxxxxxxxxID")

# Display list of devices
devices = c.getdevices()
print("Device List: %r" % devices)

# Select a Device ID to Test
id = "xxxxxxxxxxxxxxxxxID"
```

```
# Display Properties of Device
result = c.getproperties(id)
print("Properties of device:\n", result)

# Display Status of Device
result = c.getstatus(id)
print("Status of device:\n", result)

# Send Command - Turn on switch
commands = {
    "commands": [
        {"code": "switch_1", "value": True},
        {"code": "countdown_1", "value": 0},
    ]
}
print("Sending command...")
result = c.sendcommand(id,commands)
print("Results\n:", result)
```

Script Local

```
import tinytuya

# Connect to Device
d = tinytuya.OutletDevice(
    dev_id='DEVICE_ID_HERE',
    address='IP_ADDRESS_HERE',      # Or set to 'Auto' to auto-discover IP
address
    local_key='LOCAL_KEY_HERE',
    version=3.3)

# Get Status
data = d.status()
print('set_status() result %r' % data)

# Turn On
d.turn_on()

# Turn Off
d.turn_off()
```

TinyTuya Classes et Fonctions

```
Global Functions
    devices = deviceScan()             # Returns dictionary of devices found
```

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

  Device Commands:

    status()                            # Fetch status of device (json
payload)
    subdev_query(nowait)                # query sub-device list and online
status (only for gateway devices)
    detect_available_dps()              # Return list of DPS available from
device
    set_status(on, switch=1, nowait)    # Control status of the device to
'on' or 'off' (bool)
                                        # nowait (default False) True to send
without waiting for response
```

```
    set_value(index, value, nowait)    # Send and set value of any DPS/index
on device.
    set_multiple_values(index_value_dict, nowait)
                                       # Set multiple values with a single
request
                        # Note: Some devices do not like this!
    heartbeat(nowait)                  # Send heartbeat to device
    updatedps(index=[1], nowait)       # Send updatedps command to device to
refresh DPS values
    turn_on(switch=1, nowait)          # Turn on device / switch #
    turn_off(switch=1, nowait)         # Turn off device
    set_timer(num_secs, nowait)        # Set timer for num_secs on devices
(if supported)
    generate_payload(command, data)    # Generate TuyaMessage payload for
command with data
    send(payload)                      # Send payload to device (do not wait
for response)
    receive()                          # Receive payload from device

    OutletDevice:
        set_dimmer(percentage):

    CoverDevice:
        open_cover(switch=1):
        close_cover(switch=1):
        stop_cover(switch=1):

    BulbDevice
        set_colour(r, g, b, nowait):
        set_hsv(h, s, v, nowait):
        set_white(brightness, colourtemp, nowait):
        set_white_percentage(brightness=100, colourtemp=0, nowait):
        set_brightness(brightness, nowait):
        set_brightness_percentage(brightness=100, nowait):
        set_colourtemp(colourtemp, nowait):
        set_colourtemp_percentage(colourtemp=100, nowait):
        set_scene(scene, nowait):              # 1=nature, 3=rave, 4=rainbow
        set_mode(mode='white', nowait):        # white, colour, scene, music
        result = brightness():
        result = colourtemp():
        (r, g, b) = colour_rgb():
        (h,s,v) = colour_hsv():
        result = state():

    Cloud
        setregion(apiRegion)
    cloudrequest(url, action=[POST if post else GET], post={}, query={})
        getdevices(verbose=False)
        getstatus(deviceid)
        getfunctions(deviceid)
        getproperties(deviceid)
```

```
        getdps(deviceid)
        sendcommand(deviceid, commands [, uri])
    getconnectstatus(deviceid)
    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={})
```

Documentation avancée

Projet github
