Smart Tuya Cloud integration with Raspberry PLC with Python

v. 2021-1

Shrivastava Amil

April 28, 2021

Introduction

Tuya smart is a global IoT platform that is enabling brands worldwide to produce Domotics and AI solutions and enabling OEMs, retailers and developers to come up with their own unique solutions and products using a very vast range of Tuya smart compatible electronics.

This manual will walk you through the process of connecting and using Tuya compatible WIFI Smart Devices (Plugs, Switches, lights etc.) with Industrial Raspberry PLC using python for AI and Domotics purposes.

Contents

1 Overview of the hardware used

1.1 Prerequisites

Before starting with this manual, the following requirements need to be fulfilled:

1. An Active internet connection to communicate with the cloud
2. A smart phone with Smart Life application
3. Tuya Smart compatible device
4. Industrial PLC that supports Python
5. For registering the tuya device to the network and the cloud, The smart phone and the device need to be in the same network. For controlling, the PLC and the device have to be in the same network where the device has been registered.

1.2 LoraTap Curtain Switch Module (2nd Gen.)

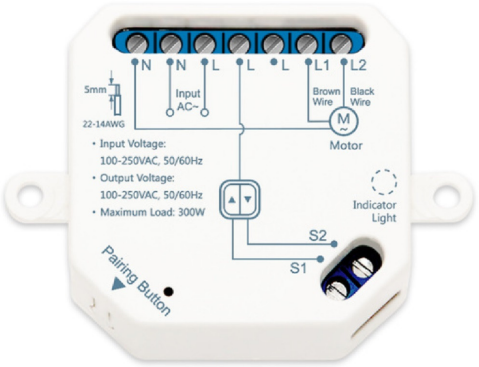

Figure 1: Loratap 2nd Gen. Curtain Switch

2nd generation LoraTap Curtain module is a wireless 4 wired curtain motor kinetic switch with a competitive price. It is compatible with Tuya smart life cloud services and Smart life mobile application. Its technical specifications and other details can be found on this link.

1.3 Industrial Raspberry PLC

Figure 2: Industrial Raspberry PLC.

Industrial Raspberry PLC is one of the most popular PLCs in Industrial Shields' line of PLCs. With the capability and flexibility of a Raspberry Pi and the Industrial compatibility and endurance with Industrial shields, this PLC is proving to be very versatile with its applications in different fields. The Industrial Raspberry PLC runs on the latest version of raspbian which is a debian based open source operating system.

More details about Industrial Raspberry PLC can be found on this link.

2 Module network Configuration

To run the module remotely in a network, the module needs to be first registered in Tuya cloud. This can only be done using the Smart life or Tuya mobile application. In this section, we will go through the steps to accomplish this requirement.

2.1 Smart life App Set up

1. Download the mobile application using either of the following two links:
   - https://itunes.apple.com/us/app/smart-life-smart-living/id1115101477?mt=8
   - https://play.google.com/store/apps/details?id=com.tuya.smartlife&hl=en
2. Create an account or Log in if you already have an account.

2.2 Module cloud registration

1. On the top right corner of the Home tab, tap on the add button to add new module or Add a device by tapping the "Add Device" button.

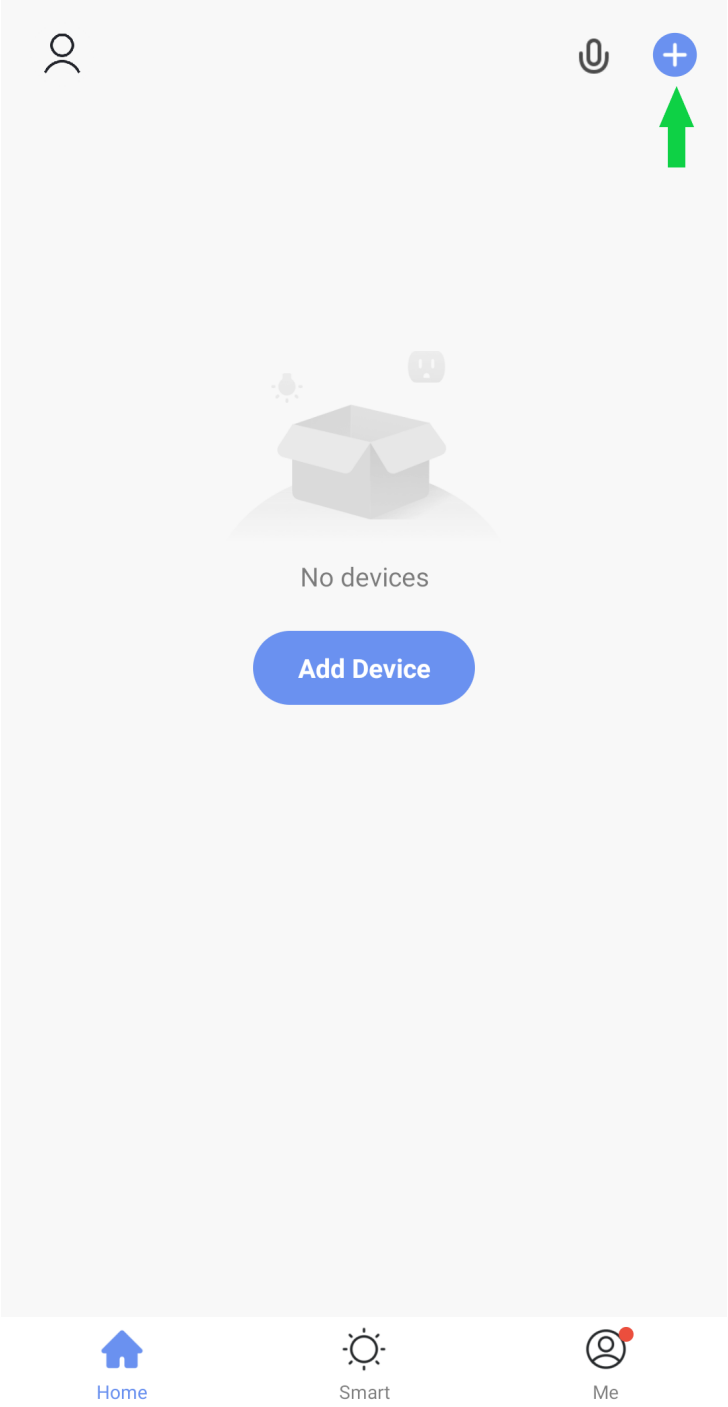

Figure 3: Adding a new device

2. Choose the appropriate device in this step. In our case we choose WIFI curtain Switch.

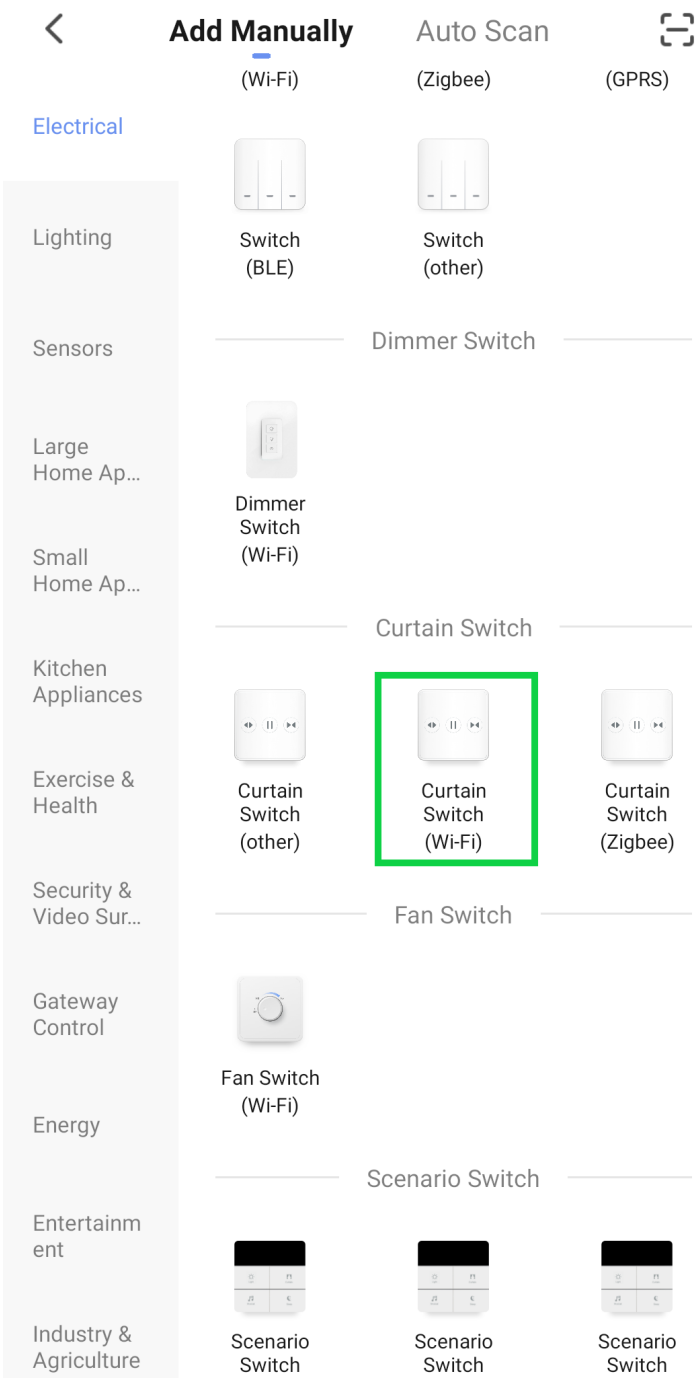

Figure 4: Choosing the device type

3. Follow the instructions on the screen. Usually it will ask you to press and hold a button on the module, or repeatedly press a button on the module.

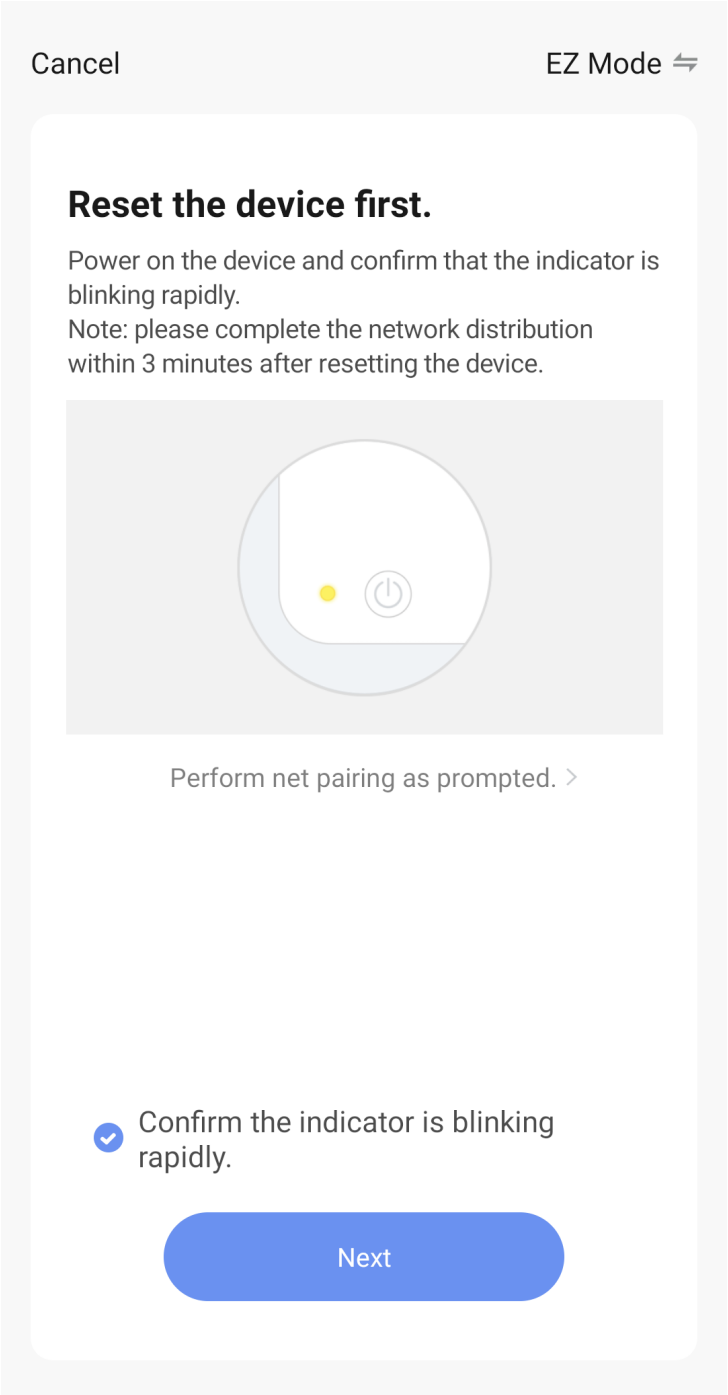

Figure 5: Follow the instructions shown on your app screen

4. The module will automatically register itself on the cloud

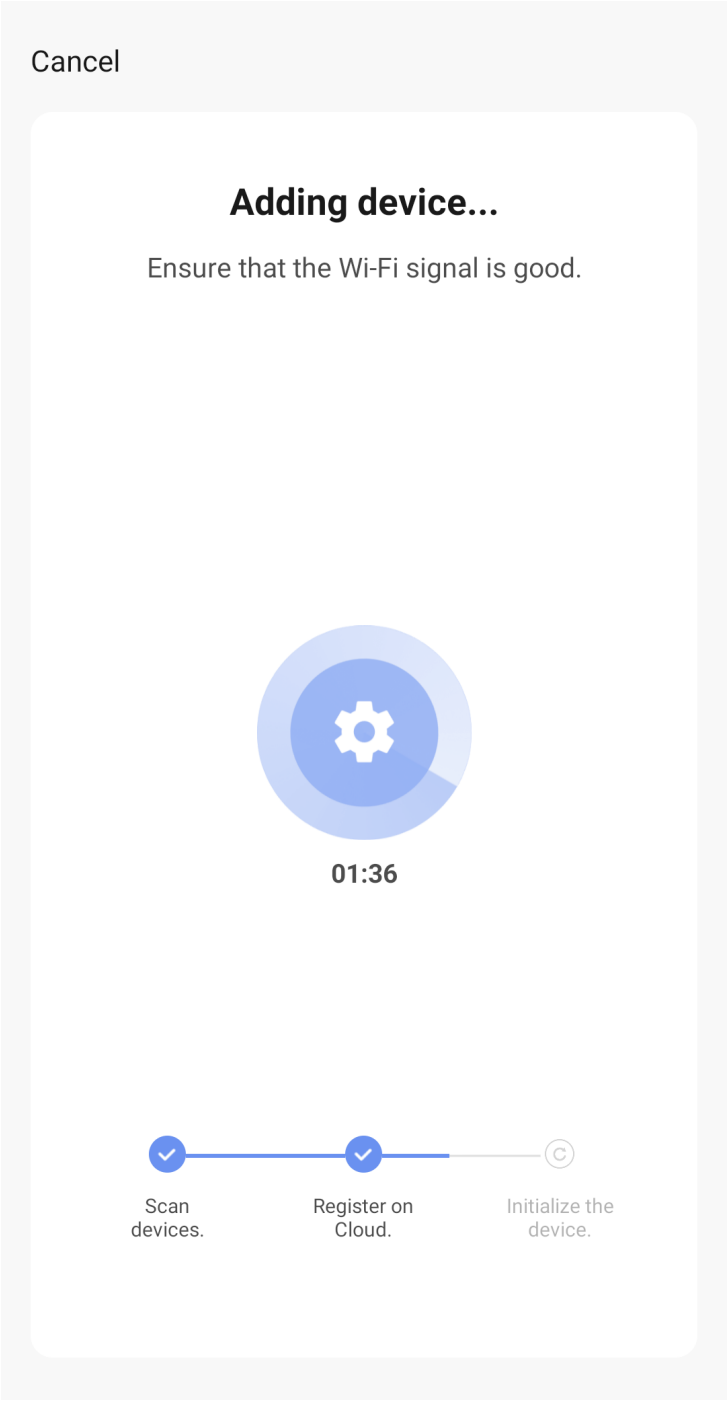

Figure 6: Device getting registered on the cloud automatically

5. Test the module using the controls of the application

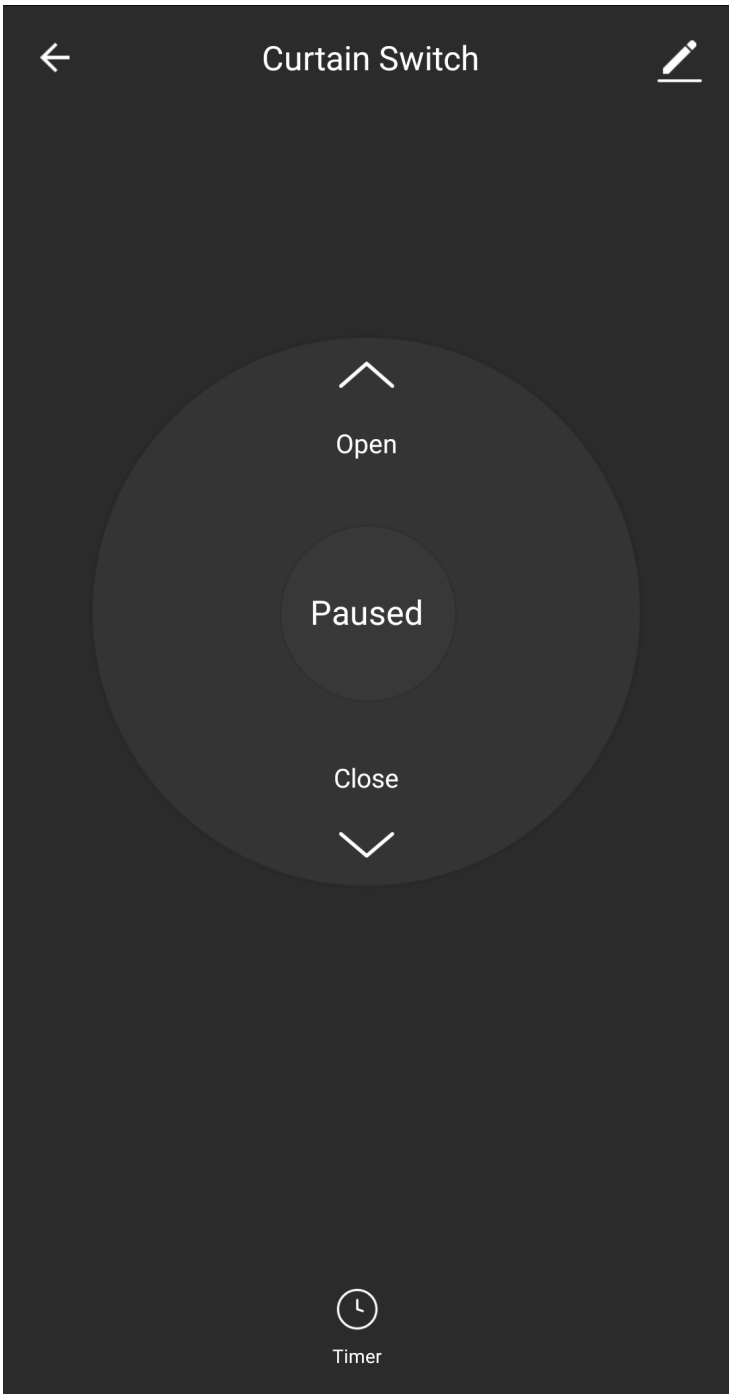

Figure 7: App Controls

Once done, the module can be used from any network

3 Tuya Development Environment Set up

In order to be able to be develop solutions with the module and control it via your own applications, a developer's account is needed in the Tuya's IoT developer's Platform.

3.1 Sign up and project creation

1. Go to this link and sign up a developer's account
2. Go to cloud and create a project and fill up the required details

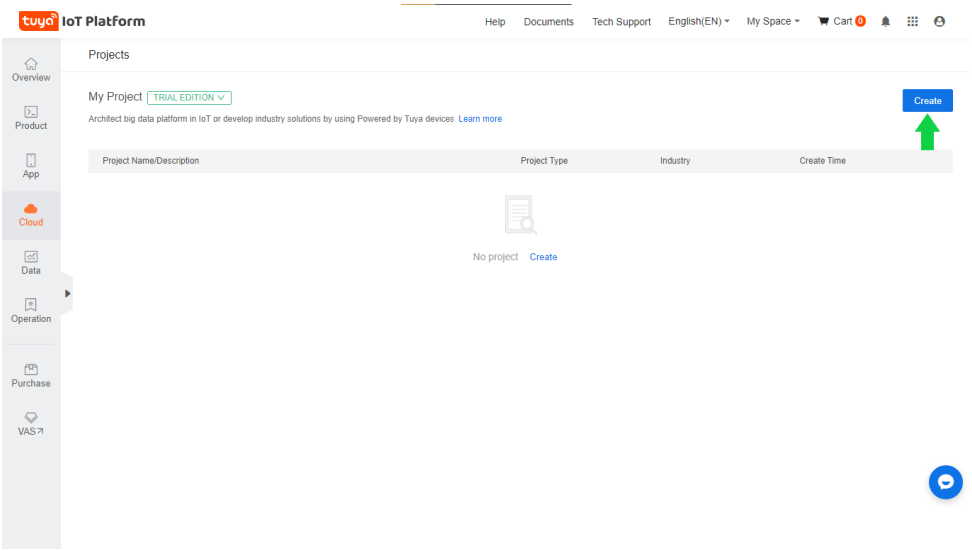

Figure 8: Create Project

3. Take note of Access ID and Access Secret. We will need it in the next steps.

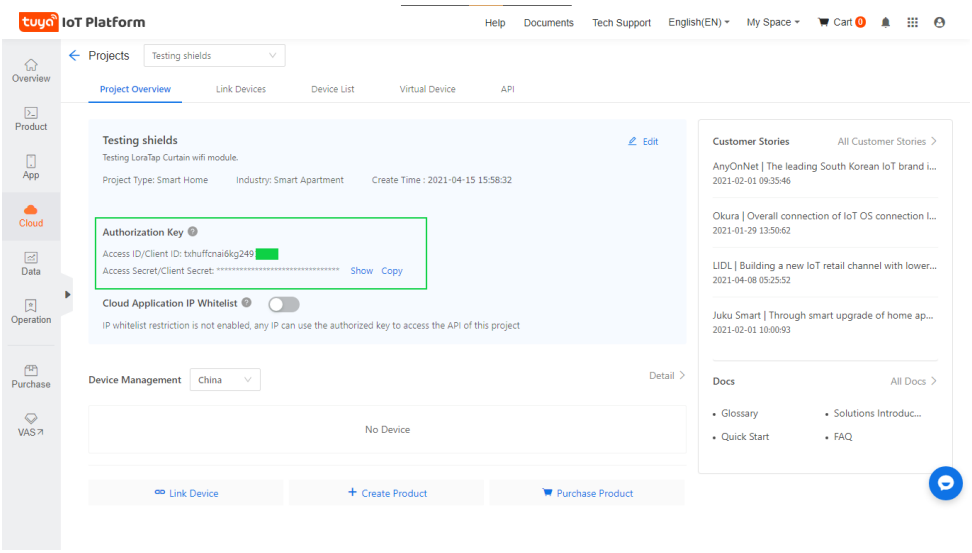

Figure 9: Project Dashboard

3.2 Adding API products

1. Go to API Products and subscribe to the following. It is free and does not require any form of payment at the time of writing this manual.

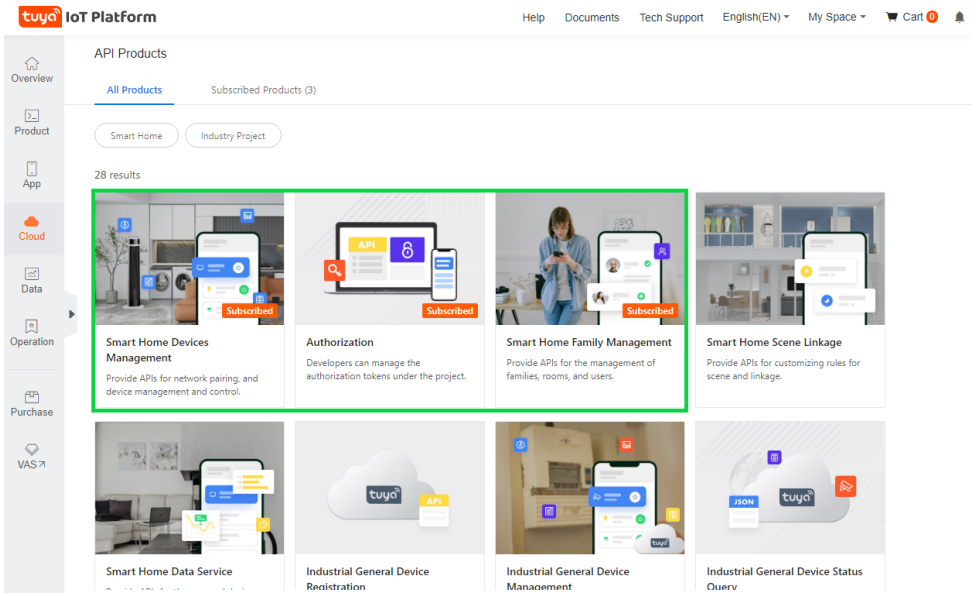

Figure 10: API Subscriptions

You will have to authorize them to be linked with your project.

2. Select your project and go to the API tab to link the APIs with your project.

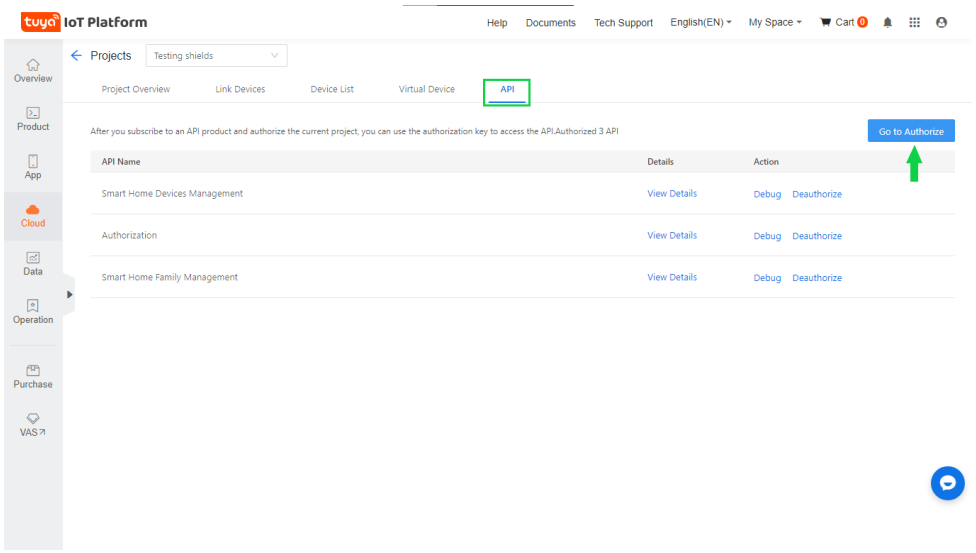

Figure 11: Authorizing APIs

3. Once, that is done, click on the Link Devices option as shown, followed by Link devices by App account.

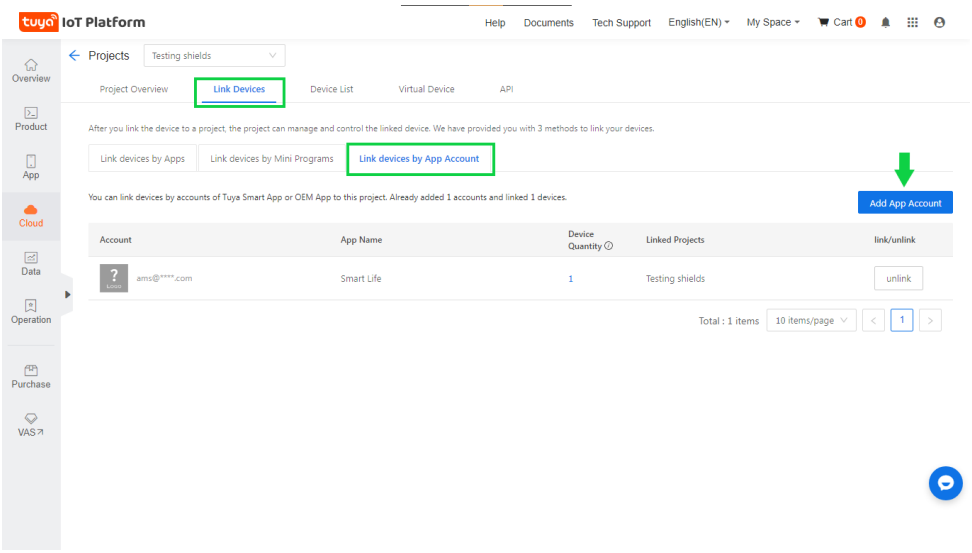

Figure 12: Linking devices by app account

4. Click on the Add App account option and scan the QR code with your mobile application.
5. You can now click on Device List tab, choose App account and your region from the drop down list and your devices linked with your app account should be listed below.

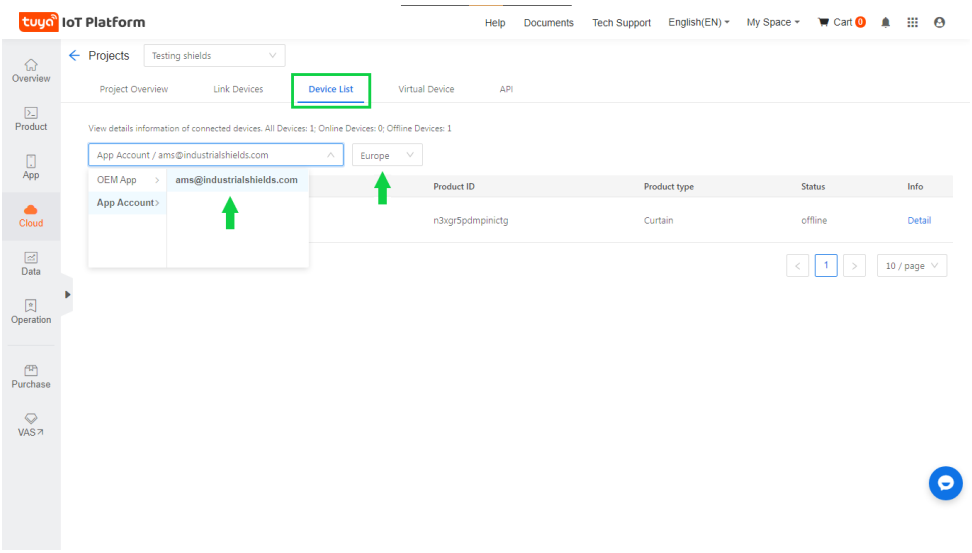

Figure 13: Create Project

4 Industrial PLC configuration and Code

Controlling and monitoring Tuya devices on a network requires the following parameters:

- IP - The network address (IPv4) of the device e.g. 10.0.1.100
- ID - The unique identifier for the Tuya device
- VERSION - The Tuya protocol version used (3.1 or 3.3)
- LOCAL_KEY - The security key created to encrypt and decrypt communication. Devices running the latest protocol version 3.3 (e.g. Firmware 1.0.5 or above) will require a device_KEY to read the status. Both 3.1 and 3.3 devices will require a device LOCAL_KEY to control the device.

  The next few sections will explain how to obtain these parameters.

4.1 Installing TinyTuya libraries

The TinyTuya library needs to be installed in the PLC via command line interface to be able to use its features for our applications. To install it run the following commands:

```
# Install PIP
  $  sudo apt-get install python-crypto python-pip
# Install TinyTuya
  $  python -m pip install tinytuya
```

4.2 Commands

4.2.1 Scan the network for online Tuya devices

TinyTuya has a built in network scanner that can be used to find Tuya Devices on your local network. It will show IP, ID and VERSION for each device.

```
$ python -m tinytuya scan
```

4.2.2 Setup Wizard to acquire the required parameters

TinyTuya has a built in setup Wizard that uses the Tuya IoT Cloud Platform to generate a JSON list (devices.json) of all your registered devices. This includes the secret LOCAL_KEY as well as the Name of each device.

```
$ python -m tinytuya wizard
# use -nocolor for non-ANSI-color terminals
```

- The Wizard will ask for the API ID key, API Secret, API Region (us, eu, cn or in) from your Tuya IoT project mentioned in step 3 of section 3.1 . It will also ask for a sample Device ID. Use one from the scanning step in section 4.2.1 or found in the Device List on your Tuya IoT project.

- The Wizard will poll the Tuya IoT Cloud Platform and print a JSON list of all your registered devices with the "name", "id" and "key" of your registered device(s). The "key"s in this list are the Devices' LOCAL_KEY you will use to access your device.
- In addition to displaying the list of devices, Wizard will create a local file devices.json. TinyTuya will use this file to provide additional details to scan results from tinytuya.scanDevices() or when running python -m tinytuya to scan your local network.
- The Wizard will ask if you want to poll all the devices. If you do, it will display the status of all devices on records and create a snapshot.json file with the results.

> Re-pairing or Re-setting the device will reset the LOCAL_KEY and will need to repeat the above mentioned procedure to obtain the key.

4.3 Programming with TinyTuya

Creating a device handle for the device we want to read or control:

```
import tinytuya

d = tinytuya.OutletDevice('DEVICE_ID_HERE',\
        'IP_ADDRESS_HERE', 'LOCAL_KEY_HERE')
d.set_version(3.3)
data = d.status()
print('set_status() result %r' % data)
```

TinyTuya Module Classes and Functions

```
    Global Functions
    devices = scanDevices()         # returns dictionary of devices
        found on local network
    scan()                          # interactive scan of local
        network
    wizard()                        # interactive setup wizard

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)

 Functions:

    json = status()                 # returns json payload
    set_version(version)            # 3.1 [default] or 3.3
```

```
set_socketPersistent(False/True) # False [default] or True
set_socketNODELAY(False/True)   # False or True [default]
set_socketRetryLimit(integer)   # retry count limit [default 5]
set_socketTimeout(self, s)      # set connection timeout in
    seconds [default 5]
set_dpsUsed(dpsUsed)            # set data points (DPs)
set_retry(retry=True)           # retry if response payload is
    truncated
set_status(on, switch=1)        # Set status of the device to
    'on' or 'off' (bool)
set_value(index, value)         # Set int value of any index.
heartbeat()                     # Send heartbeat to device
updatedps(index=[1])            # Send updatedps command to
    device
turn_on(switch=1)               # Turn on device / switch #
turn_off(switch=1)              # Turn off
set_timer(num_secs)             # Set timer for num_secs
set_debug(toggle, color)        # Activate verbose debugging
    output
set_sendWait(num_secs)          # Seconds to wait after sending
    for response
detect_available_dps()          # Return list of DPS available
    from device
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):
    set_hsv(h, s, v):
    set_white(brightness, colourtemp):
    set_white_percentage(brightness=100, colourtemp=0):
    set_brightness(brightness):
    set_brightness_percentage(brightness=100):
    set_colourtemp(colourtemp):
    set_colourtemp_percentage(colourtemp=100):
    set_scene(scene):          # 1=nature, 3=rave, 4=rainbow
    set_mode(mode='white'):    # white, colour, scene, music
    result = brightness():
    result = colourtemp():
    (r, g, b) = colour_rgb():
```

```
(h,s,v) = colour_hsv()
result = state():
```

4.4 TinyTuya Error Codes

Starting with v1.2.0 TinyTuya functions will return error details in the JSON data responses instead of raising exceptions. The format for this response:

```
{ "Error":"Invalid JSON Payload", "Err":"900","Payload":"{Tuya
   Message}" }'
```

The "Err" number will be one of these:

- 900 (ERR_JSON) - Invalid JSON Response from Device
- 901 (ERR_CONNECT) - Network Error: Unable to Connect
- 902 (ERR_TIMEOUT) - Timeout Waiting for Device
- 903 (ERR_RANGE) - Specified Value Out of Range
- 904 (ERR_PAYLOAD) - Unexpected Payload from Device
- 905 (ERR_OFFLINE) - Network Error: Device Unreachable
- 906 (ERR_STATE) - Device in Unknown State
- 907 (ERR_FUNCTION) - Function Not Supported by Device

4.5 Industrial PLC Code

For ease of understanding and demonstration, we will run an application on Industrial raspberry PLC that will change the status of the Tuya device on inputting the appropriate input via the CLI. In our case, we are working with a curtain switch, which has a total of 3 states

- Open - 1
- Pause - 2
- Close - 3

This code can be modified and have automated responses or different trigger methods like counters, timers , external buttons etc.

Code

```
import tinytuya
import time
import os

#Device details
d=tinytuya.OutletDevice('0665207370039f108cc5','192.168.177.115','921b031edd016258')
d.set_version(3.3)
data= d.status()
```

```
print('Dictionary %r' %data)
print('State (bool,true is ON) %r' %data ['dps']['1'])

switch_state= data['dps']['1']
print("We are going to do some testing")
time.sleep(1)

ip=input("Type 1-- Open ;2-- Pause ;3-- Close : ")


while True:

   inpu1= int(ip)

   if inpu1 == 1:
      print("Switch Open")
      data= d.set_status(1,1) # Open
      print("status changed %r" %data)
#In built LEDs
      os.system("sudo ./set-digital-output Q0.0 1 && sudo
          ./set-digital-output Q0.1 0 && sudo ./set-digital-output
          Q0.2 0")
        ip=input("Type 1-- Open ;2-- Pause ;3-- Close : ")


   elif inpu1 == 2:
      print("Switch Pause")
      data= d.set_status(2,1) # Pause
      os.system("sudo ./set-digital-output Q0.0 0 && sudo
          ./set-digital-output Q0.1 1 && sudo ./set-digital-output
          Q0.2 0")
      print("status changed %r" %data)
      ip=input("Type 1-- Open ;2-- Pause ;3-- Close : ")


   elif inpu1 == 3 :
      print("Switch close")
      data= d.set_status(3,1) # Close
      os.system("sudo ./set-digital-output Q0.0 0 && sudo
          ./set-digital-output Q0.1 0 && sudo ./set-digital-output
          Q0.2 1")
      print("status changed %r" %data)
      ip=input("Type 1-- Open ;2-- Pause ;3-- Close : ")

   else :
      print("Wrong input; Testing ended")
########################End#############################
```

For more information or troubleshooting, please visit the links in the References section.

To use the inputs and outputs of the PLC in the python code, the related files must be in the same folder as the python code. These files are found in the test folder. The links to the tutorial of using inputs and outputs of Industrial raspberry PLC are in the next section.

4.6 Other useful links.

- How to access the Raspberry Pi industrial
- Basics about digital inputs of a Raspberry PLC
- Basics about digital outputs of Raspberry PLC
- Basics about analog inputs of Raspberry PLC
- Basics about Raspberry Pi PLC analog outputs
- How to connect industrial Raspberry PLC to Wi-Fi

References

[1] https://developer.tuya.com/en/docs/iot

[2] https://pypi.org/project/tinytuya/