Hardware APIs

Command interface between Home Automation hardware and Smartphone application

1.1 Introduction:

Here in following document we introduced our developers to the APIs for hardware interface. To be more specific we used UDP for local hardware and app communication.

1.2 UDP communication:

UDP used for only local hardware and app communication. UDP is comparatively lightweight and quick response protocol compared to TCP, though we need one to many type communication so UDP serves our purpose well. The first thing about our hardware UDP connection is our hardware will always broadcast all commands/responses on UDP port 13000. So if you are planning to communicate with hardware you need to send commands to UDP port 13001 to that particular hardware IP or you can also perform broadcast (though we recommend you to send to specific IP it will reduce un necessary work load to other hardware devices). The second thing about our hardware is it will respond to any command came from either UDP or MQTT on local and over internet. Therefore, any action taken by one app user will be reflect on all other user’s smartphone as well. For debug purpose, you can use “Packet Sender” software on Windows/Mac PC. Keeping in mind that our hardware will respond same response multiple time on UDP to resolve problem of package drop, but on MQTT it will be single command only.

Command flow

2.1 Add new hardware process:

Add process of our hardware can only be perform on LAN. To identify if hardware is present In LAN or not, you need to broadcast MST command on UDP. So all connected hardware will respond its name and hardware IDs and if LAN includes multiple devices connected then response will be multiple. By this manner, you can also identify the individual IPs of that hardware device. (UDP API includes sender's IP).

2.1.1 MST command:

Following is the sample of MST command to be send by application

{“cmd”:”MST”}

The response for above command by hardware will be as below.

{"cmd":"MST","device_id":2695322,"m_name":"Software Team","vendor":"VDT","wifi_ver":"1.0.4","hw_ver":"8.1.3","serial":182,"type":"standalone"}

device_id: a unique hardware id we need this if we need to login this hardware for identification

m_name: device's user define name, in this case: Software Team.

vendor: to which vendor this hardware is dedicated to. (VDT/WhiteLion/Etc)

wifi_ver: current firmware version of wifi chip.

Hw_ver: Hardware detail to identify which features are supported in that hardware

serial: simple counter which keeps increase after every new command, also helps to detect UDP multiple responses for single command

type: if hardware is direct wifi based than you will get “standalone”. Else it might be “master”/”slave”

2.1.2 LIN command:

This command is used to login to device through Administrator's account it can be done only on UDP. Below is the sample command to be send by app to login a device. In this command app need to add “device_id” parameter which it got by MST command response. The command below can be broadcast and only matching device_id hardware will only respond to this login request.

{"cmd":"LIN","user":"Admin","pin":"1234","device_id":"2695322"}

user: user name for admin account.

pin: pin number of admin account.

Below is the response for a login request by app. It includes parameters which app must need for further command's additional parameters.

If username and password is incorrect then device will respond as follows.

{"cmd":"LIN","wifi_ver":"1.0.4","hw_ver":"8.1.3","device_id":2695322,"status":"error","type":"unknown","serial":238}

In addition, if login credentials are correct then hardware will response the command below.

{"cmd":"LIN","wifi_ver":"1.0.4","hw_ver":"8.1.3","device_id":2695322,"slave":"2018121812161008","status":"success","type":"standalone","encryption_key":"695427554f8772846b5191e6fb40b222","topic":"2018121812161008","token":"14e50a705a14256f18b8","serial":229}

slave: unique device's id serial number (app must send this with all other commands).

device id: a response from which this hardware it came from (as app send in its login request)

status: success/error (if success then and then only u will receive other parameters below)

type: standalone/master/slave (currently only standalone device are ready rest of are for future)

encryption_key: AES 256 encryption key (currently all data are unencrypted. will be use in future)

topic: MQTT topic for over internet communication

token: a unique login token. App must send this with all rest of commands. (It will only change if anyone modify either username or pin so any old token/logged-in app will be ignored by hardware)

2.1.3 STS command:

This command is used to get current information of all nodes. This information includes ON/OFF status of each individual nodes, how many nodes are configure as dimmer or normal, dimmer values if any of them are configured as dimmer, Touch lock status of each individual node, User lock status of each individual node, total number of schedules set for each individual nodes.To get status of any logged in hardware send following command. This commands includes a parameters which we got in response of successful login

{"cmd":"STS","token":"14e50a705a14256f18b8","slave":"2018121812161008"}

The response for above command will be as follows.

{"dimmer":[255,72,44,39,255,255,255,255],"cmd":"STS","slave":"2019020715452008","button":"0102030405060708","val":"0AAA000A","dval":"X521XXXX","touch_lock":"NNNNNNNN","user_locked":"NNNNNNNN","schedule_info":"0000000000000000","m_name":"Software","wifi_ver":"1.0.5","hw_ver":"8.1.3","tag":"formal","serial":1080}

slave: from which slave we got this response.

button: total number of nodes in this hardware.

val: 0 = off, A = on. Status of each node.

dval: dimmer values for node, X = not dimmer, 0-5 = dimmer value

touch_lock = touch lock status of individual node. N = no, Y = yes.

user_lock = user lock status of individual nodes, N = no, Y=yes.

schedule_info = total number of schedule used in each individual nodes (2 byte/node).

m_name = user defined hardware name

tag = reason why status sent. (Possible value = formal/turned on/schedule/scene)

dimmer: this parameter will only appear if hardware version is above 4.1.2/8.1.3. This will give dimmer values in 0-100% range and if node is non dimmable it will send 255 as value.

2.2 Node action commands

Node action commands are used to turn on/off nodes, touch lock/unlock, and user lock/unlock. These commands. Below are the sample commands and its response from our hardware. In all command below these two parameters are necessary to include slave and token.

2.2.1 UPD command:

This command is used to make any node turn on/off and to change dimmer value (if dimmer node). The response for UPD command is “SET” command. Keeping in mind that any manual change done by user (by changing node’s statue through physical switches on board) in that case our device will respond same SET command as well.

Following is the sample for this command

```
{"cmd":"UPD","slave":"2019020715452008","token":"e22f43e5675b2a8f52c0","node":4,"val":"A","d_val":1}
```

node = node number

val = value A = on, 0=off.

d_val : this parameter shows dimmer value in 0-100%

Response for above command will be as follows

```
{"cmd":"SET","slave":"2019020715452008","dimmer":28,"button":"04","val":"A","dval":"1","serial":1088}
```

Button: node number

val: A for on, 0 for off.

dval: x for no dimmer and 0 to 5 for dimmer value.

dimmer: this parameter shows dimmer value in 0-100%

2.2.2 000 (all zeros) cmd (Scene Command)

This command is used to set on/off and dimmer value of all nodes at once. This is more useful for scene/mood feature. Therefore, user can set all its appliances as its desired mood and in app; it can execute pre-defined scene/mood within single click.

Following is the example of ALL OFF scene command.

{"dimmer":[255,72,44,39,255,255,255,255],"cmd":"000","slave":"2018062817385708","data":"A1A1AXAX0X0X0X0X","token":"9196aaf9cfc2 c1f7e957"}

Data: 0X0X0XAX0X0X0X0X

Where 0 represents off, and A represents on

X represents normal node (on/off only) and 0-5 represent dimmer value

dimmer is array of dimmer values 0-100. 255 means the particular node is not

dimmer

Response for above command will be as follows

{"dimmer":[255,72,44,39,255,255,255,255],"cmd":"STS","slave":"2018062817385708","button":"0102030405060708","val":"0AAA000A","dval":"X521XXXX","touch_lock":"NNNNNNNN","user_locked":"NNNNNNNN","schedule_info":"0000000000000000","m_name":"Software","wifi_ver":"1.0.5","hw_ver":"8.1.3","tag":"formal","serial":1080}

Where slave: from which slave we got this response.

button: total number of nodes in this hardware.

val: 0 = off, A = on. Status of each node.

dval: dimmer values for node, X = not dimmer, 0-5 = dimmer value

touch_lock = touch lock status of individual node. N = no, Y = yes.

user_lock = user lock status of individual nodes, N = no, Y=yes.

schedule_info = total number of schedule used in each individual nodes (2 byte/node).

tag = why this status command is sent. Here its scene execution.