ESP8266 Non-OS SDK IoT_Demo Guide

Version 1.4

About This Guide

The document is structured as follows.

Chapter	Title	Content
Chapter 1	Overview	Introduction to the ***IoT_Demo***.
Chapter 2	***IoT_Demo*** Application	Introduction to compiling and using the ***IoT_Demo*** application.
Chapter 3	Curl Toolkit	Introduction to using curl commands.
Chapter 4	Functions In LAN	Introduction to the functions in LAN when running ***IoT_Demo***.
Chapter 5	Functions In WAN	Introduction to the functions through Espressif Cloud when running ***IoT_Demo***.

Release Notes

Date	Version	Release Notes
2016.04	V1.3	Initial Release.
2016.08	V1.4	Added introduction to using curl commands.

Table of Contents

1. Overview

Download ESP8266_NONOS_SDK:

http://www.espressif.com/support/download/sdks-demos

ESP8266_NONOS_SDK\examples\IoT_Demo provides simple demo implementations of three types of smart devices: Smart Light, Smart Power Plug, and Sensor Device. Featuring connectivity powered by Espressif Cloud, ESP8266 smart devices can be easily controlled through Wi-Fi with internet access and thus realise control and data collection operations.

This documentation demonstrates how to control an ESP8266 device based on ***IoT_Demo*** application by using curl commands, and the communication between ESP8266 device and Espressif Cloud.

> 📖 ***Notes:***
>
> - *Espressif Cloud http://iot.espressif.cn/#/.*
> - *Refer to http://iot.espressif.cn/#/help-en/ when using Espressif Cloud for the first time.*

2. *IoT_Demo* Application

2.1. *IoT_Demo* Introduction

The structure of ***ESP8266_NONOS_SDK\examples\IoT_Demo*** is as Figure 2-1.

Figure 2-1. *IoT_Demo* folder

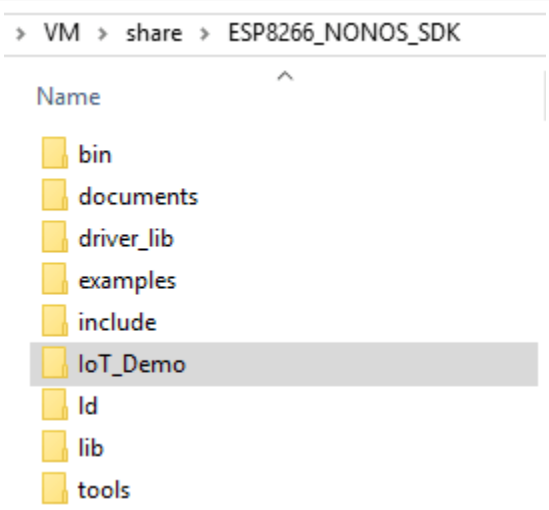
1. **user folder**
   - ***user_main.c:*** the `user_init` function in it is the application startup routine, developers can add functions for initialisation in it.
   - ***user_esp_platform.c:*** demo codes for communicating with Espressif Cloud.
   - ***user_esp_platform_timer.c:*** realises timer function with Espressif Cloud.
   - ***user_webserver.c:*** creates a TCP server.
   - ***user_devicefind.c:*** creates a UDP transmission.
   - ***user_sensor.c:*** example of ESP8266 sensor device.
   - ***user_plug.c:*** example of ESP8266 smart plug device.
   - ***user_light.c:*** example of ESP8266 smart light device.
   - ***user_json.c:*** demo codes of handling json packet.
2. **include folder**
   - header files of ***IoT_Demo***
3. **driver folder**
   - ***i2c_master.c***: demo codes of ESP8266 running as I2C master.
   - ***key.c***: example of using GPIO.

2.2. Compiling The Firmware

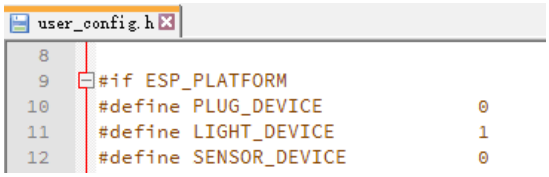
2.2.1. Modifying *IoT_Demo*

1. Download ***ESP8266_NONOS_SDK***:

   http://www.espressif.com/support/download/sdks-demos

Steps	Result
• Taking ***ESP8266_NONOS_SDK_V2.0.0_16_07_19*** as an example, users should download and unzip it. • Copy ***ESP8266_NONOS_SDK\examples\IoT_Demo*** folder (to be compiled) to the ***\ESP8266_NONOS_SDK*** root directory, as shown in the screenshot 👉.	

2. ***IoT_Demo*** provides demo application programs for three devices - Smart Light, Smart Plug and Sensor. The default device type is Smart Light.

Steps	Result
• Enable device type in ***ESP8266_NONOS_SDK\IoT_Demo\include\user_config.h***. • As is shown in the screenshot 👉, Smart Light is chosen as an example. ⚠ ***Note:*** *Please enable only one device type to implement at a time.*	

3. Modify the user parameter area location according to the actual flash size of the ESP8266 hardware module.

Steps	Result
• As is shown in the screenshot on the right 👉, taking 2048 KB flash and 512+512 map as an example, modify the value of ***#define PRIV_PARAM_START_SEC*** in ***ESP8266_NONOS_SDK\IoT_Demo\include\user_light.h***. ⚠ ***Note:*** *If Smart Plug device type is used, modify the value of* ***#define PRIV_PARAM_START_SEC*** *in* ***user_plug.h***.	user_light.h 14 #define PRIV_PARAM_START_SEC 0x7C 15 #define PRIV_PARAM_SAVE 0
• As is shown in the screenshot on the right 👉, taking 2048 kB flash and 512+512 map as an example, modify the value of ***#define ESP_PARAM_START_SEC*** in ***ESP8266_NONOS_SDK\IoT_Demo\include\user_esp_platform.h***. ⚠ ***Note:*** *Modify the same definition if Smart Plug or Sensor device type is used.*	user_esp_platform.h 4 /* NOTICE---this is for 512KB spi flash. 5 * you can change to other sector */ 6 #define ESP_PARAM_START_SEC 0x7D

Different flash maps correspond to different modified locations in header files, as listed in Table 2-1.

Table 2-1. Modifying the Field in include File (Unit: kB)

Default value (512)	Modified values					
	512	1024	2048 (512+512)	2048 (1024+1024)	4096 (512+512)	4096 (1024+1024)
0x3C	-	0x7C	0x7C	0xFC	0x7C	0xFC
0x3D	-	0x7D	0x7D	0xFD	0x7D	0xFD

⚠ ***Notice:***

The default baud rate for ESP8266 running ***IoT_Demo*** *is 74880.*

2.2.2. Compiling *IoT_Demo*

Figure 2-2 shows the process of compiling ***ESP8266_NONOS_SDK\IoT_Demo***. For more details, please refer to documentation "*ESP8266 SDK Getting Started Guide*".

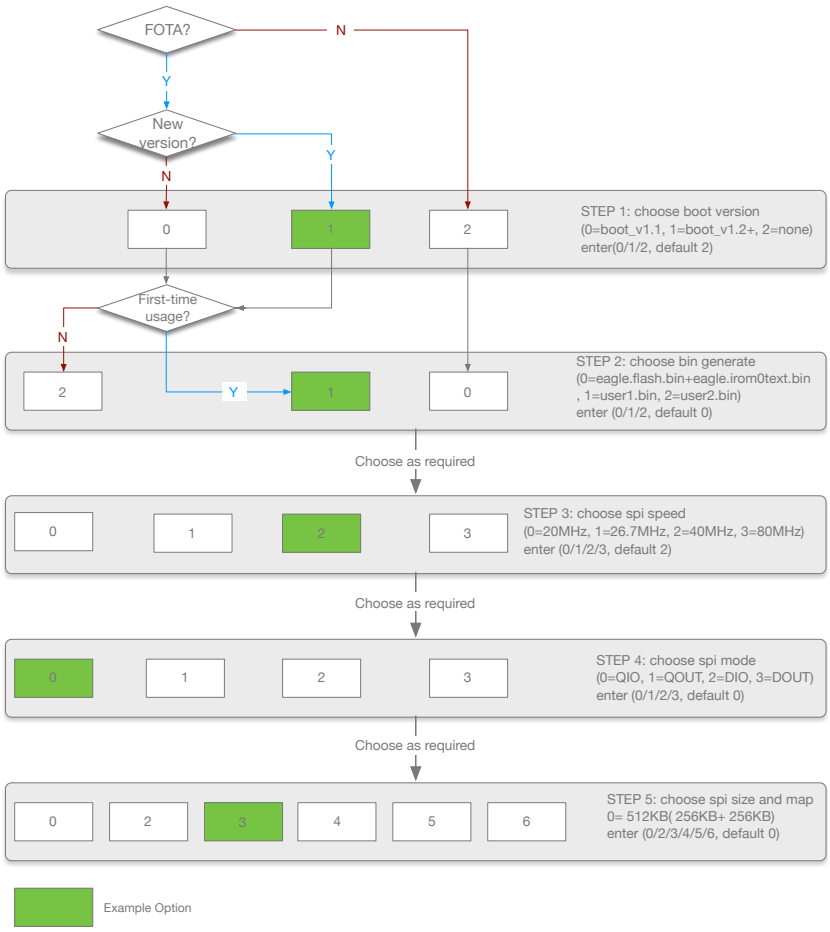


Figure 2-2. Compile *IoT_Demo*

> ⚠ ***Notice:***
>
> - *In Figure 2-2, the example options are marked in green. Users can select any option as required by the specific implementation.*
> - *Only **sdk_v1.1.0 + boot 1.4 + flash download tool_v1.2** and later versions support compiling option 5 and option 6 in STEP 5.*
> - ***user1.bin** and **user2.bin** are generated by compiling the same application code, but note to choose 2 in STEP 2 when generating **user2.bin**. More details are as below:*
>   - *Compile the application to generate **user1.bin***
>   - *Call "`make clean`" to clear the temporary files generated from the last step*
>   - *Compile the application code to generate **user2.bin** (make exactly the same choices in the compiling process except for STEP 2).*
> - ***user2.bin** is for the FOTA upgrade function, needs not to be downloaded into flash, more details are in documentation "ESP8266 FOTA Upgrade".*

2.3. Create Device On Espressif Cloud

To run ***IoT_Demo***, user needs to create a smart device on the Espressif Cloud with device type same as that in the definition in IoT_Demo source code.

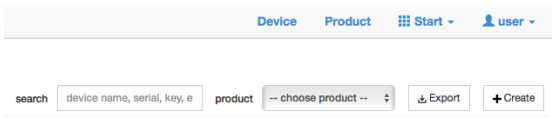
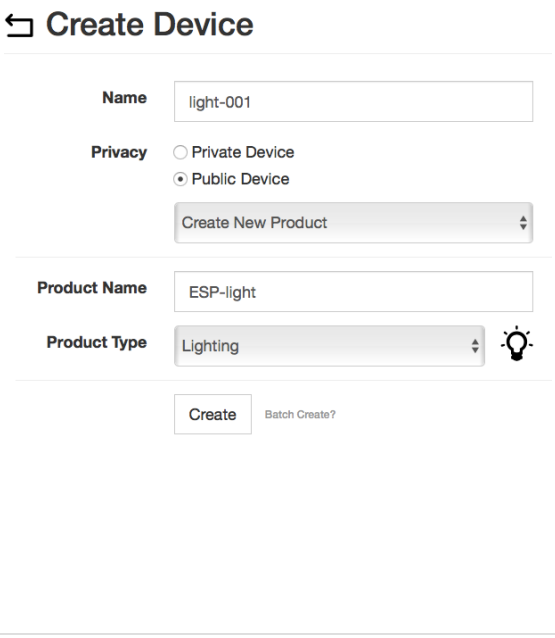
For example, if ***#define LIGHT_DEVICE*** is enabled in the ***ESP8266_NONOS_SDK\IoT_Demo\include\user_config.h*** (refer to ***Section 2.2.1***), user needs to create a smart light device on the Espressif Cloud.

2.3.1. Export *master_device_key.bin*

master_device_key is the device ID that is automatically assigned by Espressif Cloud Server when the developer builds a smart device on it. Each ID is unique. The device can get Espressif Cloud services with this ID.

> 📖 ***Note:***
>
> *Please refer to http://iot.espressif.cn/#/help-en/ when using Espressif Cloud for the first time.*

1. Register an account and log in to Espressif Cloud (*http://iot.espressif.cn/#/*), create a device.

Steps	Result
• As is shown in the screenshot 👉, log in to Espressif Cloud, click "**Device**" and then click "**+ Create**".	
• Create a smart light device, for example: - Name: light-001 - Set privacy level to "**Public Device**" which supports sharing. - Product Choice "**Create New Products**" - Product Name: ESP-light - Product Type: Lighting Then click "**Create**" at the bottom. • The process is shown in the screenshot 👉 ⚠️ ***Note:*** • *Developers can define (Device) "**Name**" and "**Product Name**" at their will.* • *If user wants to create another device type, please choose another "**Product Type**", for example, choose product type "**Plugs**" to create a smart plug device.*	

Steps	Result
• A new device page will show up on successful creation. • Master Device Key value can be seen in the device page, as shown in the screenshot 👉.	**Device** { id: 8316, serial: 3d770f9d } **light-001** Public Device Product { id: 764, name: ESP-light, serial: 604e2cb5 } Product Secret 7f126009e579e05b1cea584055605ece40e4f894 Device Secret dd1f9319cd79e8946492f6cbc3e275822a1a0e05 Master Device Key cdecad163d892f30392c2ab065b30e4c17ca7d7c

2. Export ***master_device_key.bin*** from Espressif Cloud.

Steps	Result
• Click "Download Key BIN" which is at the lower right corner of the "light-001" page, as shown in the screenshot 👉.	Download Key BIN
• The master_device_key.bin in “light-001” will be downloaded. - The name of bin file is the same as the value of Master Device Key of “light-001”.	cdecad163d892f30392c2ab065b30e4c17c... 8/5/2016 9:49 PM

2.3.2. Create Datastreams

The "Datastreams" on the Espressif Cloud can be user-defined function properties of a device type. Developer realise the functions in the application (for example, IoT_Demo) after parsing the "Datastreams" name.

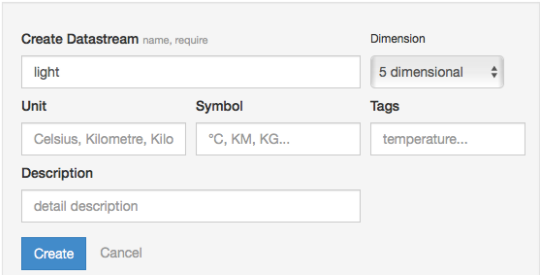
In the ***IoT_Demo***, data streams below are realised:

- For smart light device type, data stream "light" is realised to set the colour and brightness of light.
- For smart plug device type, data stream "plug-status" is realised to set the status of power plug.
- For temperature-humidity sensor device type, data stream "tem_hum" is realised to get the temperature and humidity of sensor.

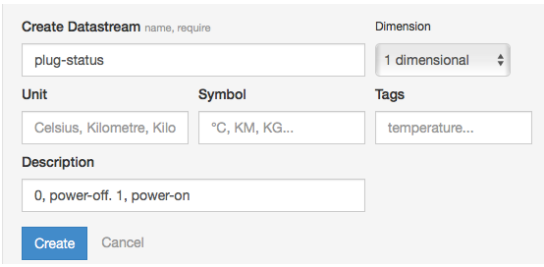
📖 ***Notes:***

- *Refer to the source code:* `IoT_Demo\user\user_esp_platform.c`.
- *On the Espressif Cloud, user can create "**Datastreams**" whether in the device page or in the product page, they are the same.*
- *Developer can realise other data streams in the application refer to **IoT_Demo**.*

1. For a smart light device, user should create "light" data stream on the Espressif Cloud.

Steps	Result
• Enter the "***light-001***" device page, click the "***+Create***" in ***Datastreams*** area.	**Datastreams** + Create
• Create a "light" data stream, for example: - Name: light - Dimension: 5 dimensional (it can has 5 parameters) - Other information is optional - Click "***Create***" • The process is shown in the screenshot 👉.	

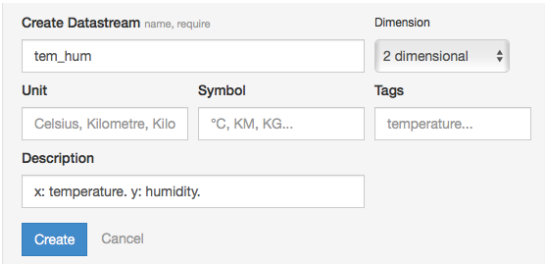
2. For a smart plug device, user should create "plug-status" data stream on the Espressif Cloud.

Steps	Result
• Create a smart plug device, according to ***Section 2.3.1***. • Enter the smart plug device page, click the "***+Create***" in ***Datastreams*** area.	**Datastreams** + Create
• Create a "plug-status" data stream, for example: - Name: plug-status - Dimension: 1 dimensional (it can has 1 parameter) - Other information is optional - Click "***Create***" • The process is shown in the screenshot 👉.	

3. For a temperature-humidity sensor device, user should create "tem_hum" data stream on the Espressif Cloud.

Steps	Result
• Create a temperature-humidity sensor device, according to ***Section 2.3.1***. • Enter the sensor device page, click the "***+Create***" in Datastreams area.	**Datastreams** + Create

- Create a "tem_hum" data stream, for example:
  - Name: tem_hum
  - Dimension: 2 dimensional (it can has 2 parameters)
  - Other information is optional
  - Click "***Create***"
- The process is shown in the screenshot 👉.



2.4. Download Addresses

Table 2-2 lists the download addresses for different flash sizes of ESP8266 module.

Table 2-2. The Download Addresses (Unit: KB)

Binaries	Download addresses for flash of different capacities					
	512	1024	2048		4096	
			512+512	1024+1024	512+512	1024+1024
master_device_key.bin	0x3E000	0x7E000	0x7E000	0xFE000	0x7E000	0xFE000
blank.bin (Partition 1)	0x7B000	0xFB000	0x1FB000		0x3FB000	
esp_init_data_default.bin	0x7C000	0xFC000	0x1FC000		0x3FC000	
blank.bin (Partition 2)	0x7E000	0xFE000	0x1FE000		0x3FE000	
boot.bin	0x00000					
user1.bin	0x01000					

Table 2-3. FOTA Firmware Description

Binaries	Description
master_device_key.bin	Applied from Espressif Cloud for Espressif Cloud services. Stored in user parameter area, and the storage address is defined by user's application program. The download address in Table 2-2 is an example set in ***IoT_Demo*** according to ***Section 2.2.1***.
blank.bin (Partition 1)	Initializes the ***RF_CAL*** parameter area. Download address is defined by function ***user_rf_cal_sector_set*** in application. The download address in Table 2-2 is an example set in ***IoT_Demo***. Provided by Espressif and is under the path ***ESP266_SDK\bin***.

Table 2-3. FOTA Firmware Description

Binaries	Description
esp_init_data_default.bin	Initializes other RF parameters area, downloaded at least once. When ***RF_CAL*** parameter area is initialized, this binary needs to be burnt as well. Provided by Espressif and is under the path ***ESP266_SDK\bin***.
blank.bin (Partition 2)	Initializes system parameter area. Provided by Espressif and is under the path ***ESP266_SDK\bin***.
boot.bin	Boot file provided by Espressif, under the path ***ESP266_SDK\bin***.
user1.bin	Main program generated by compiling application program, under the path ***ESP266_NONOS_SDK\bin\upgrade***.

3. Curl Toolkit

Download curl tool: *http://curl.haxx.se/download.html*

> 📖 ***Notes:***
>
> - *If using Windows curl tool, please refer to the curl command examples marked as "Windows curl" in the following chapters.*
> - *If using Linux or Cygwin curl tool, please refer to the curl command examples marked as "Linux / Cygwin curl" in the following chapters.*
> - *If there is no mark as "Windows curl" or "Linux/Cygwin curl", it means that the command example is suitable for both the platforms.*

Common errors when using curl command:

- Note that the curl commands are case sensitive - command may fail.
- Only English punctuation can be used in curl command, if Chinese punctuation is used, the command will fail.
- Wrong number of blank spaces will cause curl commands to fail. If blank spaces are missed, or if there are extra blank spaces, the curl command may fail.
- Please use the corresponding command format according to your tool (Windows curl or Linux/Cygwin curl). Do NOT interchange them.
- There is one-to-one correspondence between an ESP8266 device and the random token it uses to communicate with Espressif Cloud. A random token cannot be shared with multiple ESP8266 devices.

4. Functions in LAN

> 📖 ***Notes:***
>
> - *The default IP address of ESP8266 SoftAP is 192.168.4.1.*
> - *The IP address of ESP8266 Station is assigned by AP (router) after connecting to it.*
> - *The "ip" in curl commands below needs to be the actual IP address of ESP8266 SoftAP or Station.*

4.1. General Function

4.1.1. Get Version Information

PC connects to ESP8266 SoftAP as a Station and sends a curl command as below to get the version information. The "ip" in the curl command needs to be the actual IP address of ESP8266 SoftAP.

```
curl -X GET http://ip/client?command=info
```

Response:

```
{
      "Version":{
            "hardware":"0.1",
            "sdk_version":"2.0.0(656edbf)",
            "iot_version":"v1.0.5t45772(a)"
      },
      "Device":{
            "product":"Light",
            "manufacturer":"Espressif Systems"
      }
}
```

4.1.2. Set ESP8266 Station to Connect to an AP

The default mode of ESP8266 device is SoftAP+Station mode when running IoT_Demo. PC connects to ESP8266 SoftAP as a Station, and sends a curl command as below to set ESP8266 to connect to an AP as a Station.

Linux/Cygwin curl:

```
curl -X POST -H "Content-Type:application/json" -d '{"Request":
{"Station":{"Connect_Station":
```

```
{"ssid":"tenda","password":"1234567890","token":
"12345678901234567890123456789012345678901234567890"}}}}' http://192.168.4.1/
config?command=wifi
```

Windows curl:

```
curl -X POST -H "Content-Type:application/json" -d "{\"Request\":
{\"Station\":{\"Connect_Station\":{\"ssid\":\"tenda\",\"password\":
\"1234567890\",\"token\":
\"12345678901234567890123456789012345678901234567890\"}}}}" http://192.168.4.1/
config?command=wifi
```

> ⚠ ***Notice:***
>
> The red token in the curl command above is a random HEX string of 40 bytes, it cannot be shared with other devices.
>
> - ESP8266 device sends the random token to Espressif Cloud for activation and identification.
> - Then user will apply for control access to the ESP8266 device by sending the same random token to Espressif Cloud.
> - So there is one-to-one correspondence between ESP8266 device and random token. A random token cannot be shared with multiple ESP8266 devices.

Special AP configuration

If an AP is encrypted as WEP HEX, the password needs to be converted into HEX in the curl command.

For example, SSID of AP is "wifi_1", password is "tdr0123456789", encrypt as "WEP". To connect to this AP, the curl command should be as below:

Linux/Cygwin curl:

```
curl -X POST -H "Content-Type:application/json" -d '{"Request":
{"Station":{"Connect_Station":
{"ssid":"wifi_1","password":"7464723031323334353637383 9","token":
"12345678901234567890123456789012345678901234567890"}}}}' http://192.168.4.1/
config?command=wifi
```

Windows curl:

```
curl -X POST -H "Content-Type:application/json" -d "{\"Request\":
{\"Station\":{\"Connect_Station\":{\"ssid\":\"wifi_1\",\"password\":
\"7464723031323334353637383 9 \" , \"token\":
\"12345678901234567890123456789012345678901234567890\"}}}}" http://192.168.4.1/
config?command=wifi
```

Query the connection status

During the connection, the curl command below can be sent through PC to query the status of ESP8266 connecting to AP.

```
curl -X GET http://ip/client?command=status
```

The definition of "status" is as below:

```
enum {
    Station_IDLE = 0,
    Station_CONNECTING,
    Station_WRONG_PASSWORD,
    Station_NO_AP_FOUND,
    Station_CONNECT_FAIL,
    Station_GOT_IP
};
enum {
    DEVICE_CONNECTING = 40,
    DEVICE_ACTIVE_DONE,
    DEVICE_ACTIVE_FAIL,
    DEVICE_CONNECT_SERVER_FAIL
};
```

4.1.3. Set ESP8266 SoftAP Configuration

The curl command below can be used to set the configuration of ESP8266 SoftAP. For example, set the SSID, password of ESP8266 SoftAP.

Linux/Cygwin curl:

```
curl -X POST -H "Content-Type:application/json" -d '{"Request":
{"SoftAP":{"Connect_SoftAP":{"authmode":"OPEN", "channel":6,
"ssid":"ESP_IOT_SoftAP", "password":""}}}}' http://192.168.4.1/
config?command=wifi
```

Windows curl:

```
curl -X POST -H "Content-Type:application/json" -d "{\"Request\":
{\"SoftAP\":{\"Connect_SoftAP\":{\"authmode\":\"OPEN\",\"channel\":
6,\"ssid\":\"ESP_IOT_SoftAP\",\"password\":\"\"}}}}" http://
192.168.4.1/config?command=wifi
```

> ⚠ ***Notice:***
>
> - The "authmode" can be: OPEN, WPAPSK, WPA2PSK, WPAPSK/WPA2PSK.
> - The "password" has to be 8 bytes or longer, if the ESP8266 SoftAP is encrypted.

4.2. Reboot or Sleep Function

- For smart power plug or smart light device, the curl command below can be used to set the device to reboot.

```
curl -X POST http://ip/config?command=reboot
```

- For sensor device, the curl command below can be used to set the device to sleep.

```
curl -X POST http://ip/config?command=sleep
```

The sensor device will wakeup automatically after sleeping 30 seconds.

4.3. Search Devices in LAN

Users can find ESP8266 devices in the LAN by sending UDP broadcast packets to port 1025 through PC. Steps are as below:

- PC sends UDP broadcast message "Are You Espressif IOT Smart Device?" to port 1025 using a network debug tool.
- ESP8266 devices are listening to port 1025, if the message "Are You Espressif IOT Smart Device?" is received, ESP8266 devices will respond with its information.

> 📖 ***Note:***
> *The source code to realise this function is in **IoT_Demo\user\user_devicefind.c**.*

Response:

- If it is an ESP8266 smart power plug device, it will respond as shown:

```
I'm Plug.xx:xx:xx:xx:xx:xx yyy.yyy.yyy.yyy
```

- If it is an ESP8266 smart light device, it will respond as shown:

```
I'm Light.xx:xx:xx:xx:xx:xx yyy.yyy.yyy.yyy
```

- If it is an ESP8266 temperature-humidity sensor, it will respond as shown:

```
I'm Humiture.xx:xx:xx:xx:xx:xx yyy.yyy.yyy.yyy
```

- If it is an ESP8266 flammable gas sensor, it will respond as shown:

```
I'm Flammable Gas.xx:xx:xx:xx:xx:xx yyy.yyy.yyy.yyy
```

> 📖 ***Notes:***
> - "`xx:xx:xx:xx:xx:xx`" *is the actual MAC address of a ESP8266 device.*
> - "`yyy.yyy.yyy.yyy`" *is the actual IP address of a ESP8266 device.*

4.4. Smart Power Plug Device

Get the status of smart power plug device

If it is an ESP8266 smart power plug device, the curl command below can be sent to the device to query its status.

```
curl -X GET http://ip/config?command=switch
```

Response:

```
{
      "Response": {
            "status": 0
      }
}
```

> 📖 ***Note:***
>
> *"status": 0 means that the power plug is power-off, 1 means that the power plug is power-on.*

Set the status of smart power plug device

If it is an ESP8266 smart power plug device, the curl command below can be sent to the device to set its status.

Linux/Cygwin curl:

```
curl -X POST -H "Content-Type:application/json" -d '{"Response":
{"status":1}}' http://ip/config?command=switch
```

Windows curl:

```
curl -X POST -H "Content-Type:application/json" -d "{\"Response\":
{\"status\":1}}" http://ip/config?command=switch
```

4.5. Smart Light Device

Get the information of smart light device

If it is an ESP8266 smart light device, the curl command below can be used to get the information of the device.

```
curl -X GET http://ip/config?command=light
```

Response:

```
{
      "period":1000,
      "status":3,
```

```
            "color":{
                  "red":0,
                  "green":0,
                  "blue":0,
                  "white":255
            },
            "mdev_mac":”5CCF7F0A1454"
      }
```

📖 ***Notes:***

- *"period": the period of PWM, unit: millisecond.*
- *"status": the status of smart light*
  - *0: power-off*
  - *1: power-on*
  - *2: disable the white light, but change the colour of red light, blue light and green light*
  - *3: disable the red light, blue light and green light, but change the brightness of the white light*
- *"rgb": the range of colour depends on the PWM duty*
  - *version **ESP8266_NONOS_SDK_V2.0** and earlier, the range is [0, 22222]*
  - *versions later than **ESP8266_NONOS_SDK_V2.0**, the range is [0, 255]*
  - *version **ESP8266_RTOS_SDK_V1.4** and earlier, the range is [0, 1023]*
  - *versions later than **ESP8266_RTOS_SDK_V1.4**, the range is [0, 255]*

Set configuration of smart light device

If it is an ESP8266 smart light device, the curl command below can be used to set configuration of the device.

- For versions later than ESP8266_NONOS_SDK_V2.0, the curl command is as below:

Linux/Cygwin curl:

```
curl -X POST -H "Content-Type:application/json" -d '{"period":
1000,"status":3,"color":{"red":0,"green":0,"blue":0,"white":255}'
http://ip/config?command=light
```

Windows curl:

```
curl -X POST -H "Content-Type:application/json" -d “{\”period\":
1000,\”status\":3,\”color\":{\”red\":0,\”green\":0,\”blue\":0,\”white
\":255}' http://ip/config?command=light
```

- For version ESP8266_NONOS_SDK_V2.0 and earlier, the curl command is as follows:

Linux/Cygwin curl:

```
curl -X POST -H "Content-Type:application/json" -d '{"period":1000,
"rgb":{"red":200, "green":0, "blue”:0, “cwhite”:0, “wwhite”:0}}'
http://ip/config?command=light
```

Windows curl:

```
curl -X POST -H "Content-Type:application/json" -d "{\"period\":
1000,\"rgb\":{\"red\":200,\"green\":0,\"blue\":0}}" http://ip/config?
command=light
```

4.6. Sensor Device

ESP8266 sensor device running IoT_Demo cannot accessed within a local network. The status of the sensor must be obtained from Espressif Cloud through the internet.

5. Functions in WAN

> 📖 ***Notes:***
>
> - *"Device" below means that ESP8266 device will communicate with the Espressif Cloud automatically.*
> - *"PC" below means that user can send curl commands through PC to control ESP8266 device.*

5.1. ESP8266 Device Activation

Device

When the ESP8266 device has access to the internet via a router (refer to Section 4.1.2), the ESP8266 device will connect to Espressif Cloud automatically.

If it is the first time that ESP8266 device connects to the Espressif Cloud, the device will send TCP packet below to Espressif Cloud (port 8000) for activation.

```
{"path": "/v1/device/activate/", "method": "POST", "meta":
{"Authorization": "token HERE_IS_THE_MASTER_DEVICE_KEY"}, "body":
{"encrypt_method": "PLAIN", "bSSID": "18:fe:34:70:12:00", "token":
"1234567890123456789012345678901234567890"}}
```

> 📖 ***Notes:***
>
> - `"HERE_IS_THE_MASTER_DEVICE_KEY"` *means the actual master device key of ESP8266.*
> - *The red* `token` *in the curl command is the random token set to connect to AP in Section 4.1.2.*

Espressif Cloud will respond as follows:

```
{"status": 200, "device": {device}, "key": {key}, "token":
{token}}
```

PC

User can apply for device control from Espressif Cloud by sending curl command as below. Please note that in order to send command to Espressif Cloud, PC needs to have internet access.

Linux/Cygwin curl:

```
curl -X POST -H "Authorization:token
c8922638bb6ec4c18fcf3e44ce9955f19fa3ba12" -d '{"token":
"1234567890123456789012345678901234567890"}' http://iot.espressif.cn/
v1/key/authorize/
```

Windows curl:

```
curl -X POST -H "Authorization:token
c8922638bb6ec4c18fcf3e44ce9955f19fa3ba12" -d "{\"token\":
```

```
\"1234567890123456789012345678901234567890\"}" http://
iot.espressif.cn/v1/key/authorize/
```

> 📖 ***Note:***
>
> "c8922638bb6ec4c18fcf3e44ce9955f19fa3ba12" *in the curl command means the user key in Espressif Cloud. After registering on the Espressif Cloud, the user key can be obtained as follows:*
>
> - *Login to Espressif Cloud (http://iot.espressif.cn/) with user name and password.*
> - *Click the user name in the top right corner.*
> - *Click "**Settings**".*
> - *Click "**Developer**" to find the user key.*

Espresso Cloud will respond as follows:

```
{"status": 200,  "key": {"updated": "2014-05-12 21:22:03",
"user_id": 1,  "product_id": 0,  "name": "device activate share
token",  "created": "2014-05-12 21:22:03",  "source_ip": "*",
"visibly": 1,  "id": 149,  "datastream_tmpl_id": 0,  "token":
"e474bba4b8e11b97b91019e61b7a018cdbaa3246",  "access_methods": "*",
"is_owner_key": 1,  "scope": 3,   "device_id": 29,
"activate_status": 1,  "datastream_id": 0,  "expired_at": "2288-02-22
20:31:47"}}
```

> 📖 ***Note:***
>
> "e474bba4b8e11b97b91019e61b7a018cdbaa3246" *in the response means the owner key that Espressif Cloud provided to the user, makes the user the owner of the ESP8266 device. User can control the ESP8266 device through Espressif Cloud by using the owner key.*

5.2. ESP8266 Device Identification

Device

After activation, every time ESP8266 device accesses the Espressif Cloud (including the current time of activation), the device will send TCP packet below to Espressif Cloud (port 8000) for identification.

```
{"nonce": 560192812,  "path": "/v1/device/identify",  "method":
"GET",  "meta": {"Authorization": "token
HERE_IS_THE_MASTER_DEVICE_KEY"}}
```

> 📖 ***Notes:***
>
> - "nonce" *is a random number. Espressif Cloud will respond with the same* "nonce" *value of the corresponding device message.*
> - *"HERE_IS_THE_MASTER_DEVICE_KEY" means the actual master device key of ESP8266.*

If the identity authentication passed, the Espressif Cloud will respond as follows:

```
{"device": {"productbatch_id": 0,  "last_active": "2014-06-19
10:06:58",  "ptype": 12335,  "activate_status": 1,  "serial":
"334a8481",  "id": 130,  "bSSID": "18:fe:34:97:d5:33",  "last_pull":
"2014-06-19 10:06:58",  "last_push": "2014-06-19 10:06:58",
"location": "",  "metadata": "18:fe:34:97:d5:33 temperature",
"status": 2,  "updated": "2014-06-19 10:06:58",  "description":
"device-description-79eba060",  "activated_at": "2014-06-19
10:06:58",  "visibly": 1,  "is_private": 1,  "product_id": 1,
"name": "device-name-79eba060",  "created": "2014-05-28 17:43:29",
"is_frozen": 0,  "key_id": 387},  "nonce": 560192812,  "message":
"device identified",  "status": 200}
```


> 📖 ***Note:***
>
> *The identification is necessary for ESP8266 smart power plug and smart light devices.*

5.3. PING Espressif Cloud

Device

In order to keep the connection between the ESP8266 device and the Espressif Cloud, the device will send TCP packet below to Espressif Cloud (port 8000) every 50 seconds.

```
{"path": "/v1/ping/",  "method": "POST", "meta": {"Authorization":
"token HERE_IS_THE_MASTER_DEVICE_KEY"}}
```

The Espressif Cloud will respond as follows:


```
{"status": 200,  "message": "ping success",  "datetime": "2014-06-19
09:32:28",  "nonce": 977346588}
```


> 📖 ***Note:***
>
> *The ping mechanism is necessary for ESP8266 smart power plug and smart light devices.*

5.4. Smart Power Plug Device

Device:

- When ESP8266 smart power plug device receives the GET command from the Espressif Cloud, it will respond with its status.

The GET command from the Espressif Cloud is as shown:


```
{"body": {},  "nonce": 33377242,  "is_query_device": true,  "get":
{},  "token": "HERE_IS_THE_OWNER_KEY",  "meta": {"Authorization":
"token HERE_IS_THE_OWNER_KEY"},  "path": "/v1/datastreams/plug-
status/datapoint/",  "post": {},  "method": "GET"}
```

ESP8266 smart power plug device will respond as follows:

```
{"status": 200,  "datapoint": {"x": 0},  "nonce": 33377242,
"is_query_device": true}
```

- When ESP8266 smart power plug device receives the POST command from the Espressif Cloud, it will change its status according to the command.

For example, the Espressif Cloud sends command as shown below to power-up the smart power plug device:

```
{"body": {"datapoint": {"x": 1}},  "nonce": 620580862,
"is_query_device": true,  "get": {},  "token":
"HERE_IS_THE_OWNER_KEY",  "meta": {"Authorization": "token
HERE_IS_THE_OWNER_KEY"},  "path": "/v1/datastreams/plug-status/
datapoint/",  "post": {},  "method": "POST", "deliver_to_device":
true}
```

The ESP8266 smart power plug device will respond as shown below after changing its status:

```
{"status": 200,  "datapoint": {"x": 1},  "nonce": 620580862,
"deliver_to_device": true}
```

> 📖 ***Note:***
>
> *The* `"nonce"` *value in the response should be the same as the* `"nonce"` *value in the corresponding command from Espressif Cloud.*

PC:

Get the status of smart power plug device

If it is an ESP8266 smart power plug device, the curl command below can be sent to the Espressif Cloud to get the status of the device.

```
curl -X GET -H "Content-Type:application/json" -H "Authorization:
token HERE_IS_THE_OWNER_KEY" http://iot.espressif.cn/v1/datastreams/
plug-status/datapoint/
```

The Espressif Cloud will respond as shown:

```
{"status": 200,  "nonce": 11432809,  "datapoint": {"x": 1},
"deliver_to_device": true}
```

Set the status of smart power plug device

If it is an ESP8266 smart power plug device, the curl command below can be sent to the Espressif Cloud to set the status of the device.

Linux/Cygwin curl:

```
curl -X POST -H "Content-Type:application/json" -H "Authorization:
token HERE_IS_THE_OWNER_KEY" -d '{"datapoint":{"x":1}}' http://
iot.espressif.cn/v1/datastreams/plug-status/datapoint/?
deliver_to_device=true
```

Windows curl:

```
curl -X POST -H "Content-Type:application/json" -H "Authorization:
token HERE_IS_THE_OWNER_KEY" -d "{\"datapoint\":{\"x\":1}}" http://
iot.espressif.cn/v1/datastreams/plug-status/datapoint/?
deliver_to_device=true
```

The Espressif Cloud will respond as shown:

```
{"status": 200, "nonce": 11432809, "datapoint": {"x": 1},
"deliver_to_device": true}
```

5.5. Smart Light Device

Device:

When ESP8266 smart light device receives the GET command from the Espressif Cloud, it will respond with its status.

The GET command from the Espressif Cloud is as follows:

```
{"body": {},  "nonce": 8968711,  "is_query_device": true,  "get": {},
"token": "HERE_IS_THE_OWNER_KEY",  "meta": {"Authorization": "token
HERE_IS_THE_OWNER_KEY"},  "path": "/v1/datastreams/light/datapoint/",
"post": {},  "method": "GET"}
```

- For versions later than ***ESP8266_NONOS_SDK_V2.0***, the ESP8266 smart light device will respond as follows:

```
{"nonce": 5619936,  "datapoint": {"x": 1, "y": 1000, "z":{"red": 0,
"green": 0, "blue": 0, "white": 255}}, "deliver_to_device": true}
```

📖 ***Notes:***

- *"x": the status of smart light*
  - *0: power-off*
  - *1: power-on*
  - *2: disable the white light, but change the colour of red light, blue light and green light*
  - *3: disable the red light, blue light and green light, but change the brightness of the white light*
- *"y": the period of PWM, unit: millisecond.*
- *"z": the colour information*

- For version ***ESP8266_NONOS_SDK_V2.0*** and earlier, the ESP8266 smart light device will respond as shown:

```
{"nonce": 5619936,  "datapoint": {"x": 1, "y": 1000, "z":100, “k":1,
“l":2}, "deliver_to_device": true}
```

When ESP8266 smart light device receives the POST command from the Espressif Cloud, it will change its status according to the command.

For example, the Espressif Cloud sends command as shown below to change the colour of the smart light device:

```
{"body": {"datapoint": {"y": 2, "x": 1, "k": 4, "z": 3, "l": 5}},
"nonce": 65470541, "mdev_mac": "18fe34ed861c", "meta":
{"Authorization": "token HERE_IS_THE_OWNER_KEY", "Time-Zone": "Asia/
Kashgar"}, "path": "/v1/datastreams/light/datapoint/", "method":
"POST", "deliver_to_device": true}
```

The ESP8266 smart light device will respond as shown below after changing its configuration.

- For versions later than ***ESP8266_NONOS_SDK_V2.0***, the ESP8266 smart light device will respond as shown:

```
{"status": 200,"nonce": 65470541, "datapoint": {"x": 1,"y": 1000,"z":
{"red": 0, "green": 0, "blue": 0, "white": 255}},"deliver_to_device":
true,"mdev_mac":"18FE34ED861C"}
```

> 📖 ***Notes:***
>
> - *"x": the status of smart light*
>   - *0: power-off*
>   - *1: power-on*
>   - *2: disable the white light, but change the colour of red light, blue light and green light*
>   - *3: disable the red light, blue light and green light, but change the brightness of the white light*
> - *"y": the period of PWM, unit: millisecond.*
> - *"z": the colour information*

- For the version ***ESP8266_NONOS_SDK_V2.0*** and earlier, the ESP8266 smart light device will respond as shown:

```
{"status": 200,"nonce": 65470541, "datapoint": {"x": 1,"y": 1000,”z":
1, “k":2, “l":3},"deliver_to_device":true,"mdev_mac":"18FE34ED861C"}
```

PC:

Get the information of smart light device

If it is an ESP8266 smart light device, the curl command below can be sent to the Espressif Cloud to get the information of the device.

```
curl -X GET -H "Content-Type:application/json" -H "Authorization:
token HERE_IS_THE_OWNER_KEY" http://iot.espressif.cn/v1/datastreams/
light/datapoint/
```

Response:

- For versions later than ***ESP8266_NONOS_SDK_V2.0***, the light device will respond as shown:

```
{"status": 200, "datapoint": {"updated": "2016-07-05 15:06:17",
"created": "2016-07-05 15:06:17", "datatype": 0, "k": 0.0, "visibly":
1, "datastream_id": 7969, "at": "2016-07-05 15:06:17", "y": 1000,
"x": 2, "z": {"red": 0, "green": 255, "blue": 0, "white": 0} , "id":
4131591, "l": 0.0}}
```

📖 ***Notes:***

- *"x": the status of smart light*
  - *0: power-off*
  - *1: power-on*
  - *2: disable the white light, but change the colour of red light, blue light and green light*
  - *3: disable the red light, blue light and green light, but change the brightness of the white light*
- *"y": the period of PWM, unit: millisecond.*
- *"z": the colour information*

- For the version ***ESP8266_NONOS_SDK_V2.0*** and earlier, the light device will respond as shown:

```
{"status": 200, "datapoint": {"updated": "2016-07-05 15:06:17",
"created": "2016-07-05 15:06:17", "datatype": 0, "k": 0.0, "visibly":
1, "datastream_id": 7969, "at": "2016-07-05 15:06:17", "y": 0.0, "x":
1000.0, "z": 0.0, "id": 4131591, "l": 0.0}}
```

Set configuration of smart light device

If it is an ESP8266 smart light device, the curl command below can be sent to the Espressif Cloud to set the configuration of the light device.

- For versions later than ***ESP8266_NONOS_SDK_V2.0***, the curl command is as shown:

Linux/Cygwin curl:

```
curl -X POST -H "Content-Type:application/json" -H "Authorization:
token HERE_IS_THE_OWNER_KEY" -d '{"datapoint":{"x": 2,  "y": 1000,
"z": {"red": 0, "green": 255, "blue": 0, "white": 0}}} ' http://
iot.espressif.cn/v1/datastreams/light/datapoint/?deliver_to_device
=true
```

Windows curl:

```
curl -X POST -H "Content-Type:application/json" -H "Authorization:
token HERE_IS_THE_OWNER_KEY" -d "{\"datapoint\":{\"x\": 100, \"y\":
200, \"z\": {\"red\": 0, \"green\": 255, \"blue\": 0, \"white\":
0}}}" http://iot.espressif.cn/v1/datastreams/light/datapoint/?
deliver_to_device=true
```

The Espressif Cloud will respond as shown:

```
{"nonce": 28541403.0, "status": 200, "mdev_mac": "5CCF7F14C682",
"datapoint": {"updated": "2016-07-28 13:51:54", "created":
"2016-07-28 13:51:54", "datatype": 0, "k": 0, "visibly": 1, "id":
4807512, "at": "2016-07-28 13:51:54", "y": 1000, "x": 2, "z": {"red":
0, "green": 255, "blue": 0, "white": 0}, "datastream_id": 7969, "l":
50}}
```

- For the version ***ESP8266_NONOS_SDK_V2.0*** and earlier, the curl command is as shown here:

Linux/Cygwin curl:

```
curl -X POST -H "Content-Type:application/json" -H "Authorization:
token HERE_IS_THE_OWNER_KEY" -d '{"datapoint":{"x": 100,  "y": 200,
"z": 0,  "k": 0, "l": 50}} ' http://iot.espressif.cn/v1/datastreams/
light/datapoint/?deliver_to_device=true
```

Windows curl:

```
curl -X POST -H "Content-Type:application/json" -H "Authorization:
token HERE_IS_THE_OWNER_KEY" -d "{\"datapoint\":{\"x\": 100, \"y\":
200, \"z\": 0, \"k\": 0, \"l\": 50}}" http://iot.espressif.cn/v1/
datastreams/light/datapoint/?deliver_to_device=true
```

The Espressif Cloud will respond with the following:

```
{"nonce": 28541403.0, "status": 200, "mdev_mac": "5CCF7F14C682",
"datapoint": {"updated": "2016-07-28 13:51:54", "created":
"2016-07-28 13:51:54", "datatype": 0, "k": 0, "visibly": 1, "id":
4807512, "at": "2016-07-28 13:51:54", "y": 200, "x": 100, "z": 0,
"datastream_id": 7969, "l": 50}}
```

5.6. Sensor Device

Device:

Running ***IoT_Demo***, ESP8266 temperature-humidity sensor device will upload its datapoint to the Espressif Cloud automatically. The packet ESP8266 sent will be as shown:

```
{"nonce": 153436234,  "path": "/v1/datastreams/tem_hum/datapoint/",
"method": "POST",  "body": {"datapoint": {"x": 35, "y": 32}},
"meta": {"Authorization": "token  HERE_IS_THE_MASTER_DEVICE_KEY"}}
```

The Espressif Cloud will respond with the following:

```
{"status": 200,  "datapoint": {"updated": "2014-05-14 18:42:54",
"created": "2014-05-14 18:42:54",  "visibly": 1,  "datastream_id":
16,  "at": "2014-05-14 18:42:54",  "y": 32,  "x": 35,  "id": 882644}}
```

📖 ***Notes:***

- *"x": the datapoint of temperature*
- *"y": the datapoint of humidity*
- *The latest time stamp will be in the response of Espressif Cloud.*

PC:

If it is an ESP8266 temperature-humidity sensor device, the curl command below can be sent to the Espressif Cloud to get the latest temperature and humidity information of the sensor.

```
curl -X GET -H "Content-Type:application/json" -H "Authorization:
token HERE_IS_THE_OWNER_KEY" http://iot.espressif.cn/v1/datastreams/
tem_hum/datapoint
```

📖 ***Note:***

If the Espressif Cloud respond with "remote device is disconnect or busy", it is normal, the reason being that the temperature-humidity sensor cannot be remotely controlled.

If it is an ESP8266 temperature-humidity sensor device, the curl command below can be sent to the Espressif Cloud to get the historical temperature and humidity information of the sensor.

```
curl -X GET -H "Content-Type:application/json" -H "Authorization:
token HERE_IS_THE_OWNER_KEY" http://iot.espressif.cn/v1/datastreams/
tem_hum/datapoints
```

5.7. User-defined Reverse Control

The Espressif Cloud supports user-defined reverse control actions. User can send an action to the ESP8266 device through the Espressif Cloud with additional parameters to realise flexible reverse control.

The curl command format of user-defined reverse controlling is as follows:

Linux/Cygwin curl:

```
curl -X GET -H "Content-Type:application/json" -H "Authorization:
token HERE_IS_THE_OWNER_KEY" 'http://iot.espressif.cn/v1/device/rpc/?
deliver_to_device=true&action=your_custom_action&any_parameter=any_va
lue'
```

Windows curl:

```
curl -X GET -H "Content-Type:application/json" -H "Authorization:
token HERE_IS_THE_OWNER_KEY" "http://iot.espressif.cn/v1/device/rpc/?
```

```
deliver_to_device=true&action=your_custom_action&any_parameter=any_va
lue"
```

The packet that ESP8266 device receives will be as shown here:

```
{"body": {},  "nonce": 872709859,  "get": {"action":
"your_custom_action",  "any_parameter": "any_value",
"deliver_to_device": "true"},  "token": "HERE_IS_THE_DEVICE_KEY",
"meta": {"Authorization": "token HERE_IS_THE_DEVICE_KEY "},  "path":
"/v1/device/rpc/",  "post": {},  "method": "GET",
"deliver_to_device": true}
```

📖 ***Notes:***

- *The* `"your_custom_action"`, `"any_parameter"`, `"any_value"` *marked red above denote the user-defined configuration of reverse controlling.*
- *Developer needs to add functions in the* ***IoT_Demo*** *to parse the action and parameter of reverse controlling, and realise the corresponding operation.*
- *The user-defined reverse controlling actions will not be saved in the Espressif Cloud, it does not record history.*

Espressif IOT Team

www.espressif.com