ESP8266 FOTA Introduction

Version 1.5

Espressif Systems IOT Team

Table of Contents

1. Preambles

Herein, we introduce how to upgrade firmware through WiFi (FOTA: Firmware Over The Air) base on Espressif Cloud.

It contains the storage of firmware in flash , FOTA procedure and APIs introduction, as a reference for software developer.

For firmware support FOTA, we need to compile to generate images as below and burn them into flash to run:

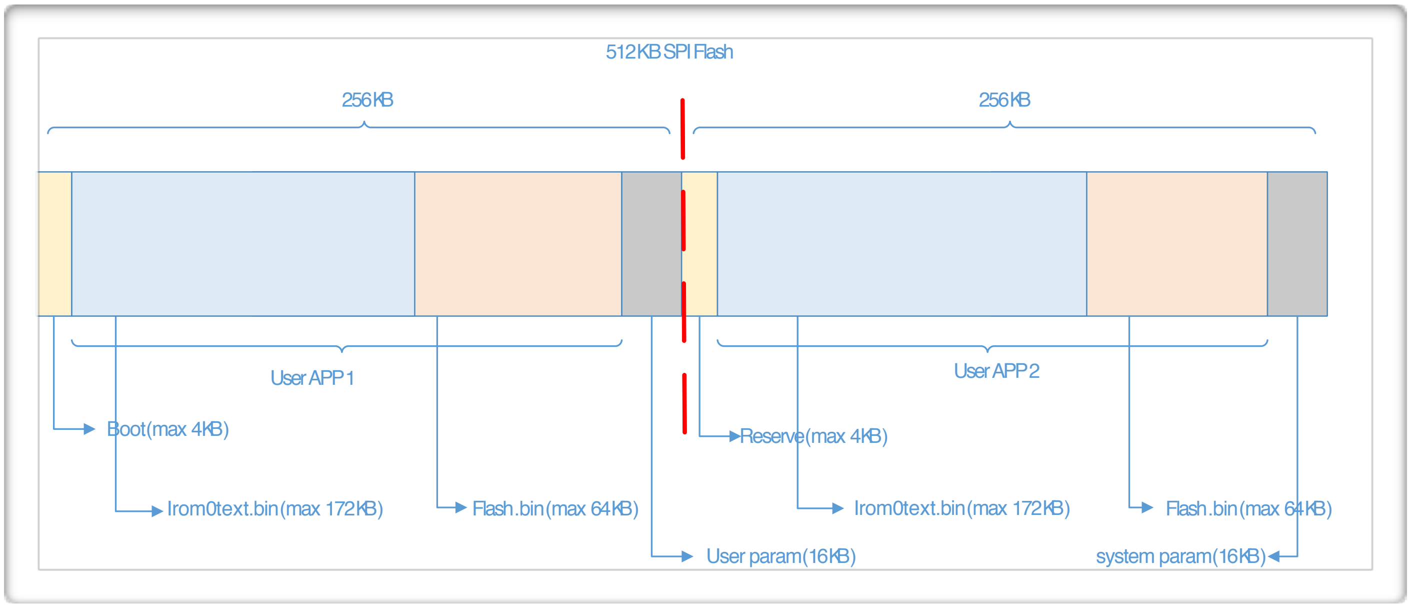
- `boot.bin` to be written into flash **0x00000**;
- `user1.bin` to be written into flash **0x01000**.

2. Flash Layout

We only introduce load region layout of firmware which support FOTA here.

2.1. 521KB Flash

For example, IOT_Demo in `esp_iot_sdk` without RTOS is using 512KB SPI Flash：



512KB SPI Flash Layout			
Sector	**Description**	**Region**	**Size**
Boot	Store `boot.bin`	0~4KB	4KB
User APP1	Store `user1.bin` (= `flash.bin` + `irom0text.bin`)	4KB~240KB	236KB
User param	Store user parameter （4 x 4KB）	240KB~256KB	16KB
Reverse	Reserved, as we need to make user2 and user1 have the same offset （0x01000）	256KB~260KB	4KB
User APP2	Store `user2.bin` (= `flash.bin` + `irom0text.bin`)	260KB~496KB	236KB
System param	Store system parameter （4 x 4KB）	496KB~512KB	16KB

User App1 and User App2 are two images generated by compiling the same application code (`user1.bin` and `user2.bin` when downloading).

The original `flash.bin` and `irom0text.bin` are combined to `user1.bin` (or `user2.bin`) for upgrading. One of the user.bin is also a backup for the other in case of upgrading failure.

Bootloader (`boot.bin`) will check the flag in System param area to decide whether execute User App1 (`user1.bin`) or User App2 (`user2.bin`).

Example:

(1) We download `boot.bin` and `user1.bin` (Version 1.0.0) to flash. Flag in system param area also be marked as `user1.bin` by default. In this case, after power up, it will always execute from `user1.bin`.

(2) We upload `user1.bin` and `user2.bin` of version 1.0.1 to server.

(3) The server will push a message of new SW available. If device got this message, it will check the flag first, flag sets as user1, so download `user2.bin` of v1.0.1 to flash region 260KB~496KB to upgrade.

(4) After download finish, end user will get FOTA message, decide whether they accept firmware upgrade. If they accept, then we'll modify the flag to user2 and reboot to run `user2.bin`. Otherwise nothing will happen.

(5) Next upgrade, starts at step 2, and download `user1.bin` of v1.0.2 to flash region 4KB~240KB covering the original `user1.bin` of v1.0.0 .

Notes:

- We need to generate both `user1.bin` and `user2.bin` during compiling time and upload both to server. Device will determine which bin it need to download.
- `user1.bin` and `user2.bin` are same software placed to different regions of flash. The only difference is address mapping on flash.
- We usually do not burn `user2.bin` into flash, we burn `user1.bin` into flash first, and upgrade to `user2.bin` through WiFi (FOTA).

2.2. 1MB or larger Flash

Flash layout of flash (1MB < size <4MB) is almost like flash (size = 1MB) except that the **"System param" area is always the last 16KB of flash**:

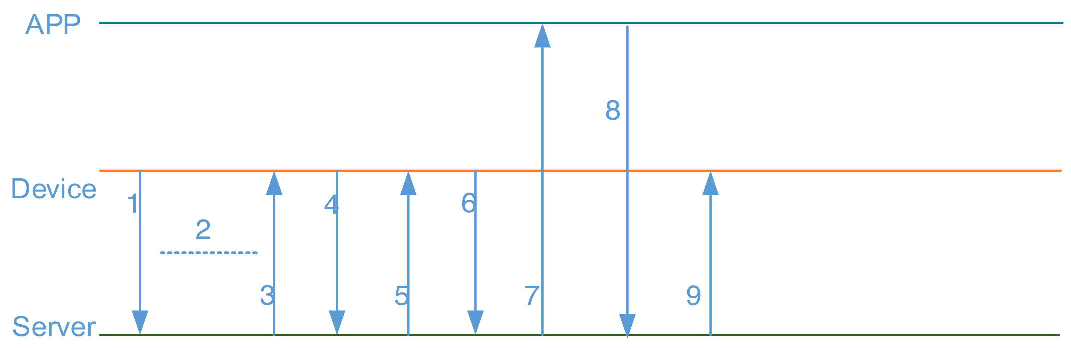
More details on BBS http://bbs.espressif.com/viewtopic.php?f=10&t=305

1MB SPI Flash Layout			
Sector	**Description**	**Region**	**Size**
Boot	Store `boot.bin`	0~4KB	4KB
User APP1	Store `user1.bin` (= `flash.bin` + `irom0text.bin`)	4KB~496KB	492KB
User param	Store user parameters（4 x 4KB)	496KB~512KB	16KB
Reverse	Reserved, as we need to make user2 and user1 have the same offset（0x01000)	512KB~516KB	4KB
User APP2	Store `user2.bin` (= `flash.bin` + `irom0text.bin`)	516KB~1008KB	492KB
System param	Store system parameters（4 x 4KB)	1008KB~1024KB	16KB

Instruction here is same as 512KB Flash.

3. FOTA procedure

FOTA procedure will be executed after new firmware bin files are uploaded to server and "upgrade" button is pressed. FOTA procedure is only executed if server firmware version is newer then device firmware version.



1. during device activation, device will upload its version information to server. Server will record device version information;

2. normal usage;

3. new firmware uploaded to server. Server push this message to device.

4. device request firmware according to its device key and upload path (`user1.bin` or `user2.bin`).

5. download firmware to flash.

6. send download finish message to server.

7. server will push this message to APP, APP display this information to end user.

8. APP send back end user decision to server.

9. server send end user decision to device. device proceed accordantly.

The communication between device, service and APP is encrypted by SSL. Device key is required from Step4. Therefore upgrade procedure is secure.

Notes:

Some device (e.g., the sensor device) will enter sleep mode in idle state. Such device do not support remote control. Our FOTA feature is only available on those device support remote control. For devices that don't support remote control, FOTA will be supported later.

4. User Guide

4.1. How to generate user1.bin and user2.bin

`user1.bin` and `user2.bin` are two images generated by compiling the same application code, they are same software placed to different regions of flash. The only difference is address mapping on flash. We compile them by choosing a boot:

- launch compiler, execute command "`./gen_misc.sh`", choose "`1=boot_v1.2+`" in STEP 1, choose "`1=user1.bin`" in STEP 2; then follow the tips choosing according to your actual configuration; `user1.bin` will be generated under "`\esp_iot_sdk\bin\upgrade`"
- execute command "make clean" to clean up all previous compilation
- execute command "`./gen_misc.sh`", choose "`1=boot_v1.2+`" in STEP 1, choose "`2=user2.bin`" in STEP 2; then follow tips choosing according to your actual configuration; `user2.bin` will be generated under "`\esp_iot_sdk\bin\upgrade`"

Notes:

MUST upload both `user1.bin` and `user2.bin` to server. Device will decide which one it will download.

If FOTA is not supported, please refer to " Espressif IoT SDK User Manual " for compile and download operations.

4.2. Burning into flash

bin	Address	Description
master_device_key.bin	0x3E000	Obtained from Espressif Cloud by users themselves to get Espressif Cloud service; to be written into the third sector of flash user parameter area which is 0x3E000 in IOT_Demo, can be changed by user. If using 1MB or larger flash, recommend to change it to 0x7E000, refer to BBS http://bbs.espressif.com/viewtopic.php?f=10&t=305
blank.bin	0x7E000	Stores default system parameter values, provided in SDK
boot.bin	0x00000	Boot loader, provided in SDK, recommend to use the latest version
user1.bin	0x01000	Compiled by the steps said above

More details about burning into flash are in documentation "Espressif IoT SDK User Manual" user2.bin need not to be burned into flash, it can be download through WiFi as upgrade.

For future updates, please upload both user1.bin and user2.bin to the server and the server will send update information to users. If users choose to update, then the device will select and download user1.bin or user2.bin, whichever is necessary for cloud update.

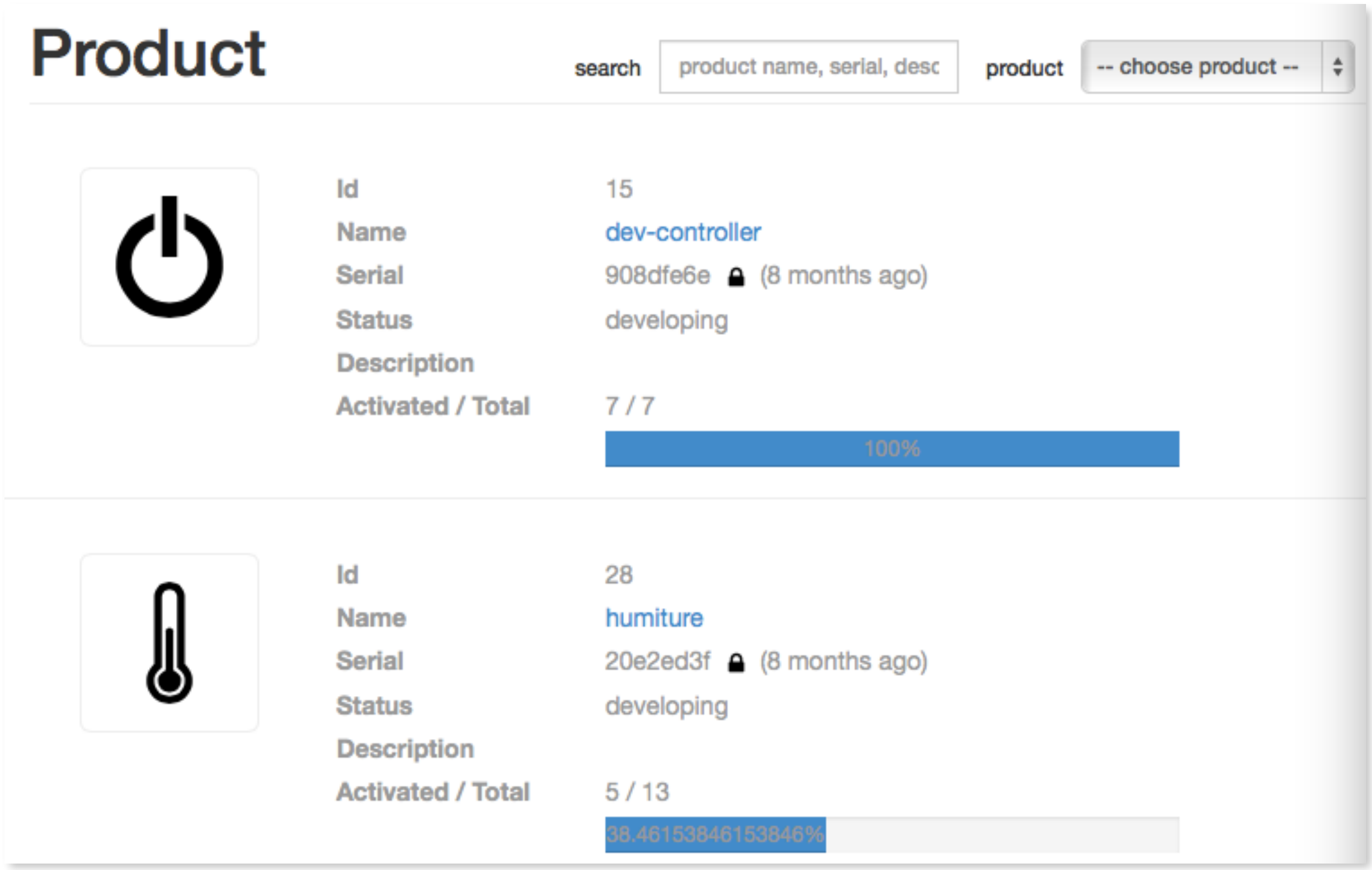
4.3. Website User Guide

Note:

User starter guide of Espressif Cloud please refer to http://iot.espressif.cn/#/help-en/

More details in document "Espressif Cloud introduction".

(1) Login into http://iot.espressif.cn/#/ , click "Product". It will list all your products.

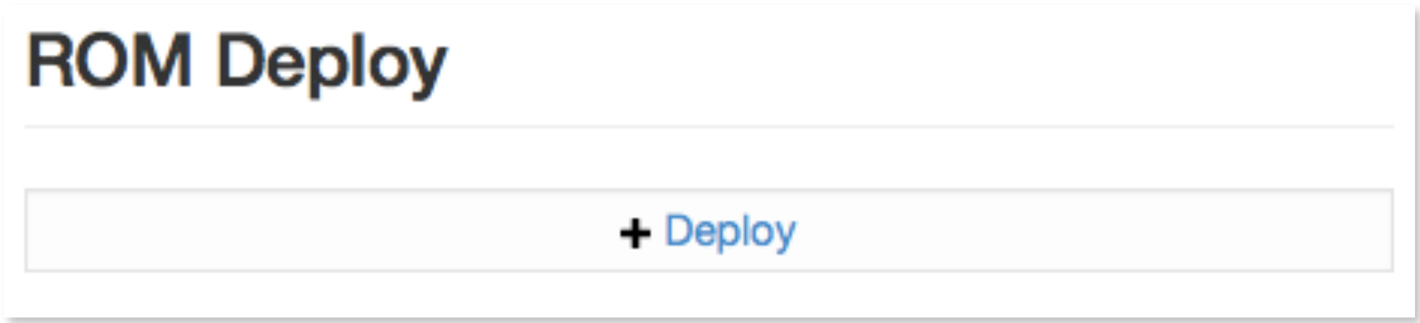


Id	15
Name	dev-controller
Serial	908dfe6e (8 months ago)
Status	developing
Description	
Activated / Total	7 / 7
	100%

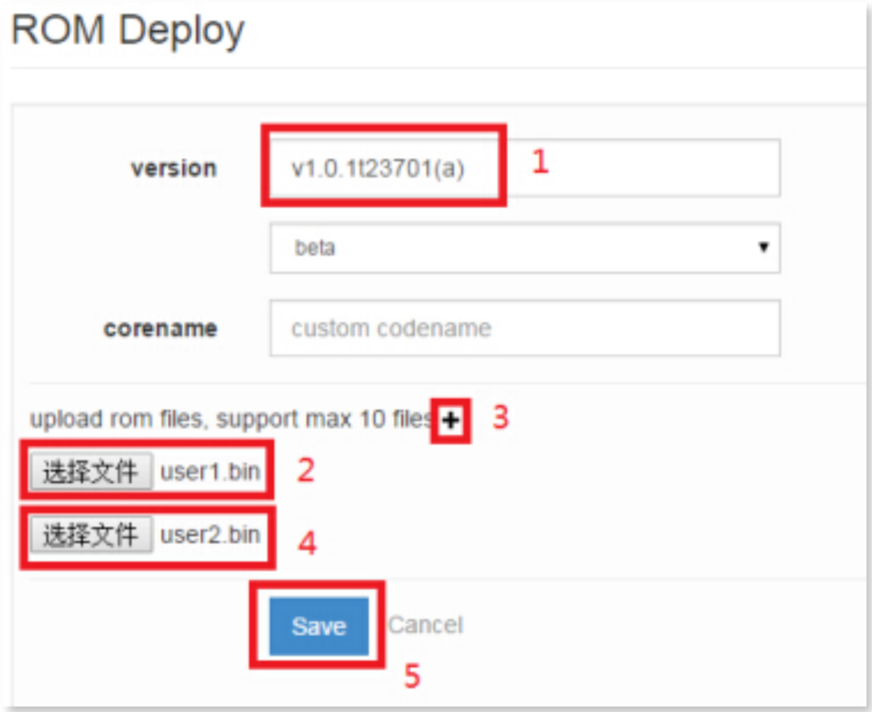
Id	28
Name	humiture
Serial	20e2ed3f (8 months ago)
Status	developing
Description	
Activated / Total	5 / 13
	38.46153846153846%

(2) Click a product which need to update firmware, find "ROM Deploy" at the right side of webpage. For example, we click "dev-controller" at above picture, and find "ROM Deploy".

(3) Click "+ Deploy" to upload a new version of firmware. Follow steps showing below. ROM version



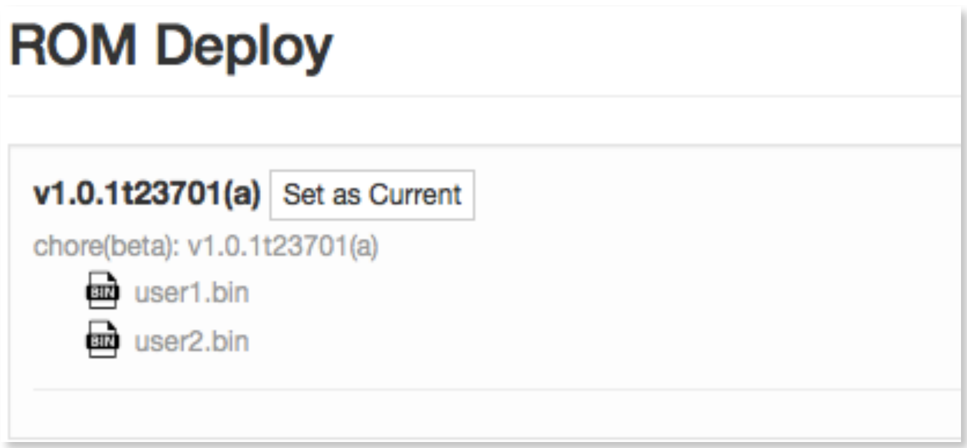
format as "v1.0.1t23701(a)" according to appendix.


ROM Deploy

version: v1.0.1t23701(a)
beta
corename: custom codename
upload rom files, support max 10 files +
选择文件 user1.bin
选择文件 user2.bin

Note: Both user1.bin and user2.bin need to be uploaded on server.

(4) After "Save", we can find it on this page. Click "Set as Current" to use the new firmware. Then server will notify all devices of product "dev-controller" that there is a new version of firmware, users can choose to upgrade or not.



ROM Deploy

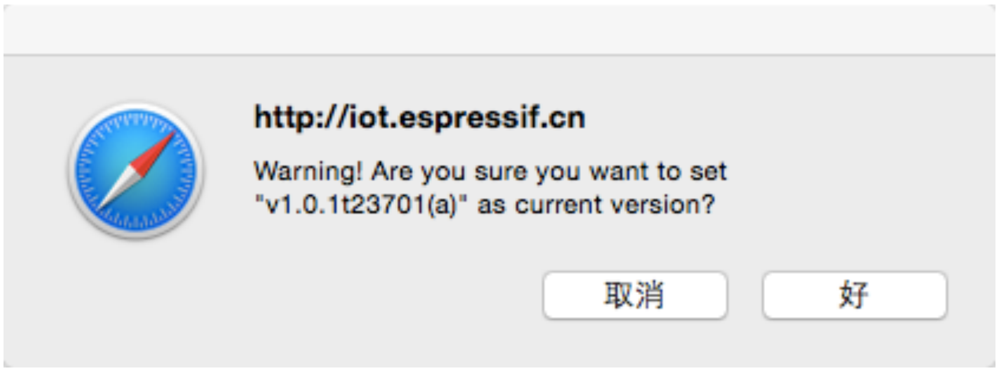
v1.0.1t23701(a) Set as Current

chore(beta): v1.0.1t23701(a)

user1.bin

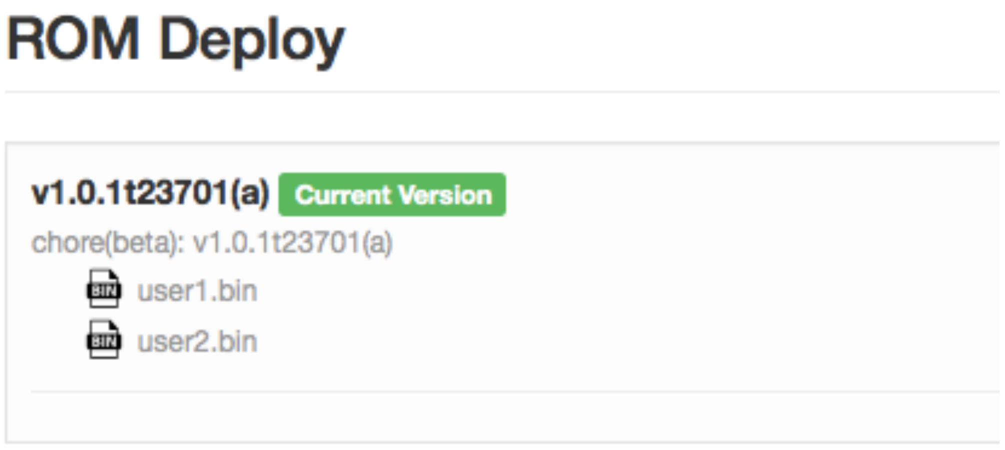
user2.bin

(5) Click "OK" to update the firmware.



http://iot.espressif.cn

Warning! Are you sure you want to set "v1.0.1t23701(a)" as current version?



ROM Deploy

v1.0.1t23701(a) Current Version

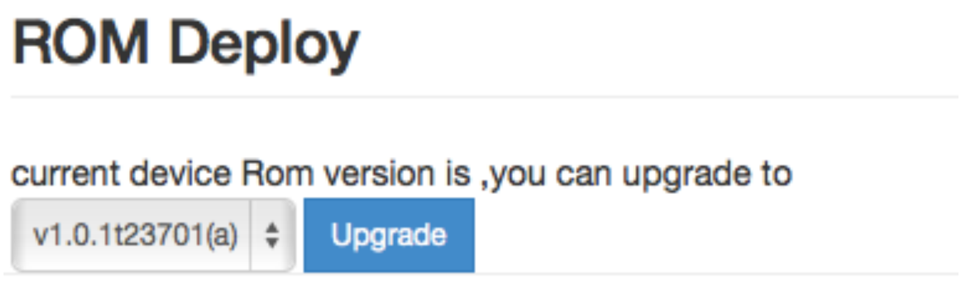
chore(beta): v1.0.1t23701(a)

user1.bin

user2.bin

(6) Users will get a message of firmware update.

(7) Click "Device", choose a device of product "dev-controller", find "ROM Deploy" in the device's page, choose the version of firmware, click "Upgrade".

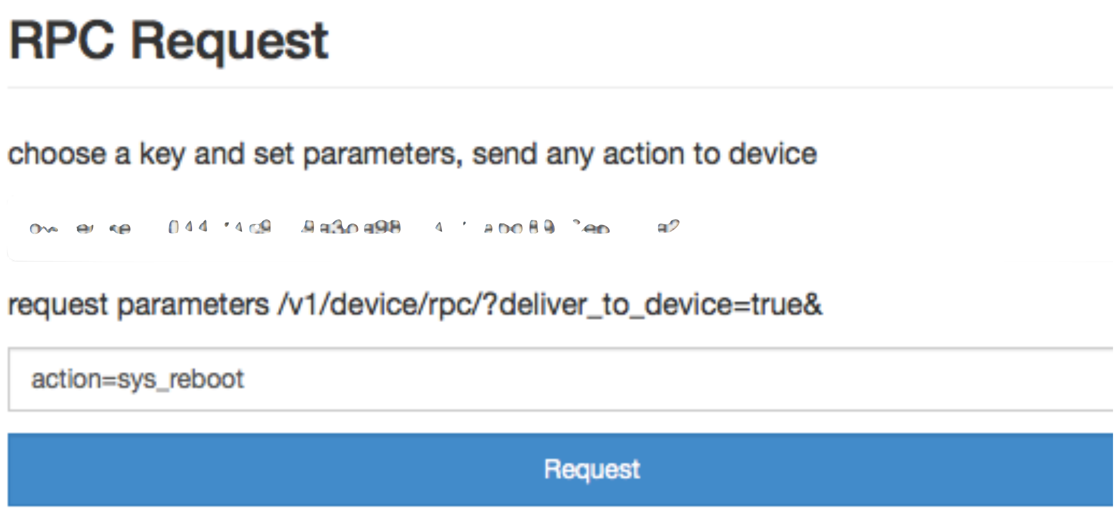


ROM Deploy

current device Rom version is ,you can upgrade to

(8) Then device will download the new firmware. If device is running `user1.bin` right now, it will download `user2.bin` from server; otherwise, download `user1.bin`.

(9) After finish downloading, we can send command from website to device, to make device reboot and run the new firmware. In the page of device, find "RPC Request", follow "action=" with "`sys_reboot`", then click "Request", this device will receive the request and reboot to run new firmware.



RPC Request

choose a key and set parameters, send any action to device

[illegible]

request parameters /v1/device/rpc/?deliver_to_device=true&

action=sys_reboot

(10) Device receives the request and reboots to run new firmware.

4.4. Curl Instructions

Note:

“Device " refers to the operation which that the device runs by itself and needs no user intervention.

"PC” refers to the commands that users can send to the device to run.

1. Request to upgrade

◇ PC

Send a request by curl to upgrade：

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=sys_upgrade&version=v1.0.1t23701(a)'
```

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=sys_upgrade&version=v1.0.1t23701(a)"
```

Note: the red v1.0.1t23701(a) above is an example, you need to input the real version info that you want to upgrade to.

◇ Device

Data packet that device will receive：

```
{"body": {}, "nonce": 855881582, "get": {"action": "sys_upgrade", "version": "v1.0.1t23701(a)", "deliver_to_device": "true"},
"token": "HERE_IS_THE_OWNER_KEY", "meta": {"Authorization": "token HERE_IS_THE_OWNER_KEY"}, "path": "/
v1/device/rpc/", "post": {}, "method": "GET", "deliver_to_device": true}
```

2. Download Finish

◇ Device

When device receives request to upgrade, it starts to download the new version of firmware (bin file) from server. While finish downloading, device will send data below to server, notify that download succeed.

```
{"path": "/v1/messages/", "method": "POST", "meta": {"Authorization": "token
HERE_IS_THE_MASTER_DEVICE_KEY"},"get":{"action":"device_upgrade_success"},"body":
{"pre_rom_version":"v1.0.0t23701(a)","rom_version":"v1.0.1t23701(a)"}}
```

3. Restart to update

◇ PC

While device notifies that download succeed, PC send a command to control device restart to use the new firmware.

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=sys_reboot'
```

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=sys_reboot"
```

◇ Device

Device will receive data packet below:

```
{"body": {}, "nonce": 856543282, "get": {"action": "sys_reboot", "deliver_to_device": "true"}, "token":
    "HERE_IS_THE_OWNER_KEY", "meta": {"Authorization": "token
    HERE_IS_THE_OWNER_KEY"}, "path": "/v1/device/rpc/", "post": {}, "method": "GET",
    "deliver_to_device": true}
```

Device get this "`sys_reboot`", it will restart and run the new firmware.

5. Software APIs

5.1. struct upgrade_server_info

```
struct upgrade_server_info{
    uint8 ip[4];             // IP address
    uint16 port;             // port number
    uint8 upgrade_flag;      // true – upgrade succeed; false – upgrade fail
    uint32 check_times;      // time out (ms)
    uint8 *url;
    upgrade_states_check_callback check_cb;
    struct espconn *pespconn;
};
```

5.2. Upgrade APIs

Since esp_iot_sdk_v0.8 we support upgrade through WiFi (FOTA):

system_upgrade_userbin_check	: Check current running bin, user1 or user2
system_upgrade_start	: Start upgrading (through HTTP)
system_upgrade_reboot	: Restart device to run new firmware
system_upgrade_flag_set	: Sets upgrade status flag
system_upgrade_flag_check	: Gets upgrade status flag

Users can refer to IOT_Demo：

user_esp_platform_upgrade_begin	: Start downloading user bin from Espressif Cloud.
user_esp_platform_upgrade_rsp	: Upgrade callback. Maybe succeed, maybe timeout.

1. system_upgrade_userbin_check

Function：	Check which firmware is running now, user1 or user2 If running user1.bin right now, we will download user2.bin from server to update. If running user2.bin right now, we will download user1.bin from server to update.
Prototype：	`uint8 system_upgrade_userbin_check()`
Return：	0x00： UPGRADE_FW_BIN1， means user1.bin 0x01： UPGRADE_FW_BIN2， means user2.bin

2. system_upgrade_start

Function：	Start downloading firmware (user bin) from server (HTTP)
Prototype：	`bool system_upgrade_start (struct upgrade_server_info *server)`
Parameter	`struct upgrade_server_info *server` - server related structure
Return：	true ： start to upgrade false： upgrading now, can not start again。

3. system_upgrade_reboot

Function:	Reboot system and use new version. If download user bin succeed, call api to restart and run new firmware.
Prototype:	`void system_upgrade_reboot (void)`
Return:	none

4. system_upgrade_flag_set

Function:	Sets upgrade status flag.
Note:	If you using `system_upgrade_start` to upgrade, this API need not be called. If you using `spi_flash_write` to upgrade firmware yourself, this flag need to be set to `UPGRADE_FLAG_FINISH`, then call `system_upgrade_reboot` to reboot to run new firmware.
Prototype:	`void system_upgrade_flag_set(uint8 flag)`

Parameter:	`uint8 flag:`
	`#define UPGRADE_FLAG_IDLE      0x00`
	`#define UPGRADE_FLAG_START     0x01`
	`#define UPGRADE_FLAG_FINISH    0x02`

Return:	none

5. system_upgrade_flag_check

Function:	Gets upgrade status flag.
Prototype:	`uint8 system_upgrade_flag_check()`

Return:	`#define UPGRADE_FLAG_IDLE      0x00`
	`#define UPGRADE_FLAG_START     0x01`
	`#define UPGRADE_FLAG_FINISH    0x02`

6. user_esp_platform_upgrade_begin

Function:	Begin to upgrade， set structure upgrade_server_info.
Prototype:	`user_esp_platform_upgrade_begin (struct espconn *pespconn, struct upgrade_server_info *server, char *version)`
Parameter:	`struct espconn` - server connection, to send response to Espressif Cloud. `struct upgrade_server_info *server` - server related structure `char *version` - version information
Return:	none

7. user_esp_platform_upgrade_rsp

Function:	Upgrade callback. Register as a callback in `user_esp_platform_upgrade_begin`. Enter this callback, if upgrade timout (check_times) or finish.
Prototype:	`esp_platform_upgrade_rsp(void *arg)`
Parameter:	`void *arg` - data pointer
Return:	none

Note:

Users can build their own server , call APIs listed here and download firmware according to HTTP, refer to IOT_Demo.

6. Appendix

When you upload device sdk firmware (user bin) to Espressif Cloud, you have to name it according to the rule, or it will be an illegal name and upgrade fail.

6.1. Naming Template

Prototype: `[v|b]Num1.Num2.Num3.tPTYPE([o|l|a|n])`

Example: v1.0.2t45772(a)

Analysis of Version Template				
v	1.0.2	t	45772	(a)
Software Type	**Version Number**	**Tag**	**Device Type**	**FOTA or not**
variable	variable	constant	variable	variable

1. Software Type: v or b

v: means official version

b: means beta version

Note: The same version number cannot have both official version and beta version. For example, there can't be both v1.0.2 and b1.0.2.

2. Version Number: NUM1.NUM2.NUM3

Num: Range from 0 to 999

Example: 1.0.2

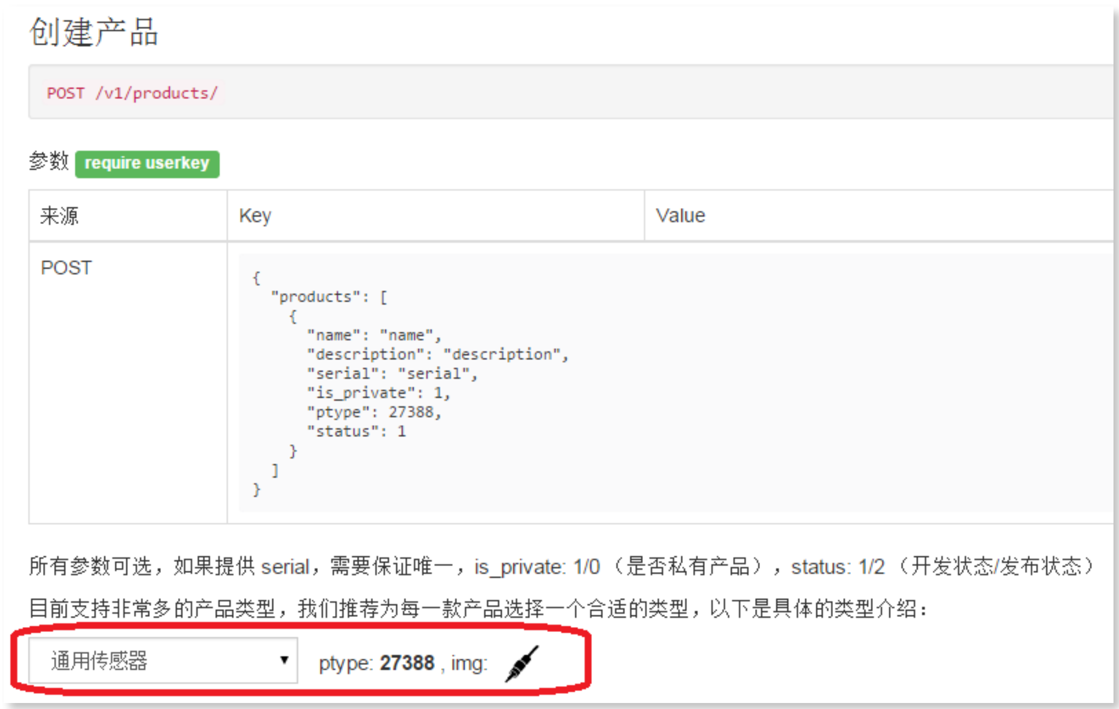
3. Tag: t

t: Type Tag, followed by device's product type number (PTYPE)

4. Device Type: PTYPE

PTYPE: ptype (product type) number, get from Espressif Cloud , as follow

Enter http://iot.espressif.cn/#/api/#api-product-create



创建产品

```
POST /v1/products/
```

参数 require userkey

来源	Key	Value
POST		

```
{
  "products": [
    {
      "name": "name",
      "description": "description",
      "serial": "serial",
      "is_private": 1,
      "ptype": 27388,
      "status": 1
    }
  ]
}
```

所有参数可选，如果提供 serial，需要保证唯一，is_private: 1/0（是否私有产品），status: 1/2（开发状态/发布状态）

目前支持非常多的产品类型，我们推荐为每一款产品选择一个合适的类型，以下是具体的类型介绍：

通用传感器 ▾ ptype: **27388** , img:

5. Firmware upgrade through WiFi (FOTA)：

o: online , supports online Upgrade

l: local , supports local Upgrade

a: all , supports both online Upgrade and local Upgrade

n: not support , can't Upgrade

6.2. Naming Rules

1. Value of Version

Refer to Device SDK version：[v|b]Num1.Num2.Num3.tPTYPE([o|l|a|n])

Value of Version Number is Num1*1000*1000 + Num2*1000 + Num3

Example：

Get ptype number of device "Light" being 45772 on Espressif Cloud, and device "Switch" is 23701. So

SDK v1.0.2t45772(a) is an official software of device "Light" of version 1.0.2, it supports both online and local upgrade; value of version number is 1*1000*1000+0*1000+2 = 1000002

SDK b1.0.3t23701(l) is a beta software of device "Switch" of version 1.0.2, it supports only local upgrade; value of version number is 1*1000*1000+0*1000+3 = 1000003

2. Rule of Version Value

(1) Same ptype, same version value, cannot have more than one upgrade type version (o,l,a)

- For example, if b1.0.3t45772(o) exists, then v1.0.3t45772(o), v1.0.3t45772(l), v1.0.3t45772(a), b1.0.3t45772(l), b1.0.3t45772(a) are not allowed.

(2) Same ptype, same version value, can only have one non-upgrade type version (n)

- For example, if b1.0.3t45772(n) exists, then v1.0.3t45772(n) is not allowed.

(3) Same ptype with version (n), cannot have more than one upgrade type version (o,l,a)

- For example, if b1.0.3t45772(n) exists, then b1.0.3t45772(o), b1.0.3t45772(l), b1.0.3t45772(a), v1.0.3t45772(o), v1.0.3t45772(l), v1.0.3t45772(a) can only have one at most.

(4) Different ptype can have the same version value.

- For example, if b1.0.3t45772(n) exists, b1.0.3t12335(n) can exist, too.