



# ESP8266 FOTA Introduction

**Version 1.6**

Espressif Systems IOT Team

Copyright © 2016



### **Disclaimer and Copyright Notice**

Information in this document, including URL references, is subject to change without notice.

THIS DOCUMENT IS PROVIDED AS IS WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE. All liability, including liability for infringement of any proprietary rights, relating to use of information in this document is disclaimed. No licenses express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein.

The WiFi Alliance Member Logo is a trademark of the WiFi Alliance.

All trade names, trademarks and registered trademarks mentioned in this document are property of their respective owners, and are hereby acknowledged.

Copyright © 2016 Espressif Systems Inc. All rights reserved.



# Table of Contents

- 1. Preambles .....4**
- 2. Flash Layout.....4**
  - 2.1. 521KB Flash .....4
  - 2.2. 1024KB and larger Flash .....5
- 3. FOTA procedure .....8**
- 4. User Guide.....9**
  - 4.1. How to generate user1.bin and user2.bin .....9
  - 4.2. Burning into flash.....9
  - 4.3. Website User Guide .....10
  - 4.4. Curl Instructions .....13
- 5. Software APIs .....15**
  - 5.1. struct upgrade\_server\_info .....15
  - 5.2. Upgrade APIs .....15
- 6. Appendix.....18**
  - 6.1. Naming Template .....18
  - 6.2. Naming Rules .....19

## 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:

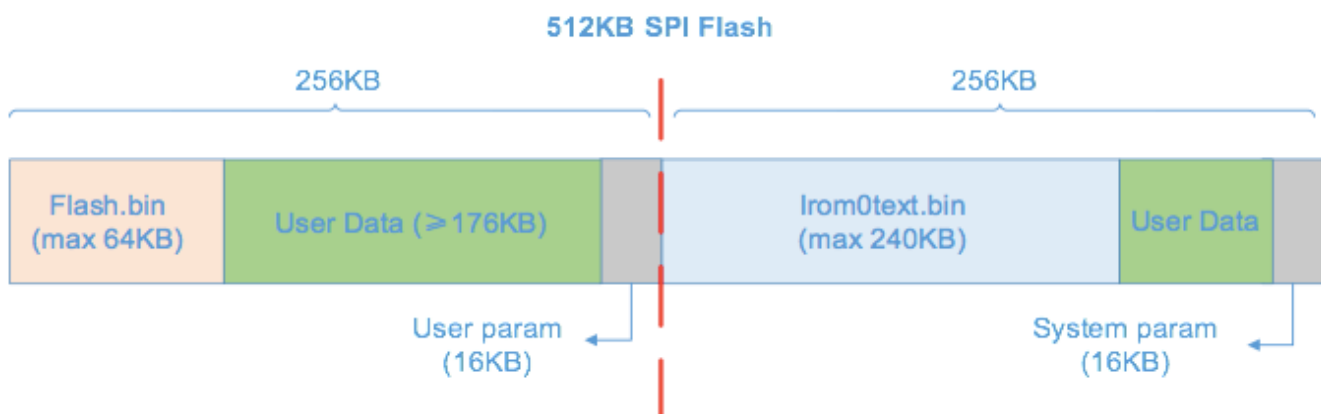
- `boot.bin` to be written into flash 0x00000;
- `user1.bin` to be written into flash 0x01000.

## 2. Flash Layout

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

### 2.1. 521KB Flash

For example, IOT\_Demo in ESP8266\_NONOS\_SDK without RTOS is using 512KB SPI Flash:



512KB SPI Flash Layout			
Sector	Description	Region	Size
Boot	Store <code>boot.bin</code>	0~4KB	4KB
User APP1	Store <code>user1.bin</code> (= <code>flash.bin</code> + <code>irom0text.bin</code> )	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 <code>user2.bin</code> (= <code>flash.bin</code> + <code>iram0text.bin</code> )	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 `iram0text.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 .

NOTE:

- 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).

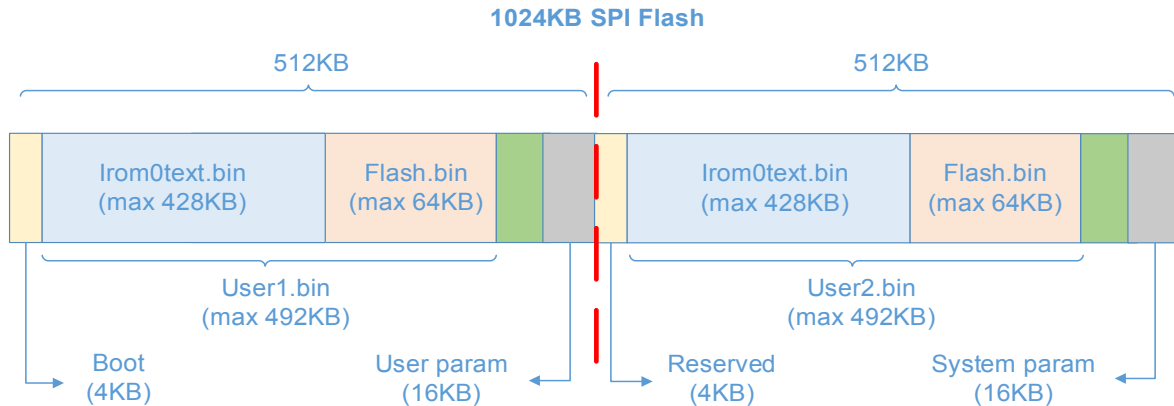
## 2.2. 1024KB and larger Flash

During compilation, choose 1 `boot_v1.2+` in STEP 1 to support FOTA, generate `user1.bin` and `user2.bin`. Then choose different flash map in STEP 5 according to your actual SPI flash. Instruction here is same as 512KB Flash.

- User Data area ( green area in pictures below ) means the flash area that may be available, if program area ( `user1.bin` and `user2.bin` ) doesn't reach the maximum size, remaining area can be used to save user data.
- "System param" area is always the last 16KB of flash.

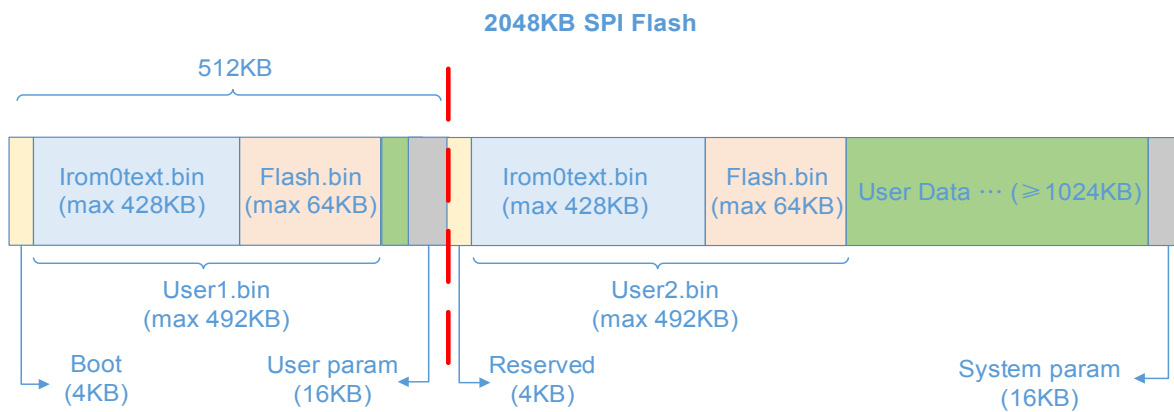
### 1. 1024KB flash

If choose 1 `boot_v1.2+` in STEP 1, choose 2 `1024KB` in STEP 5, flash map will be as below

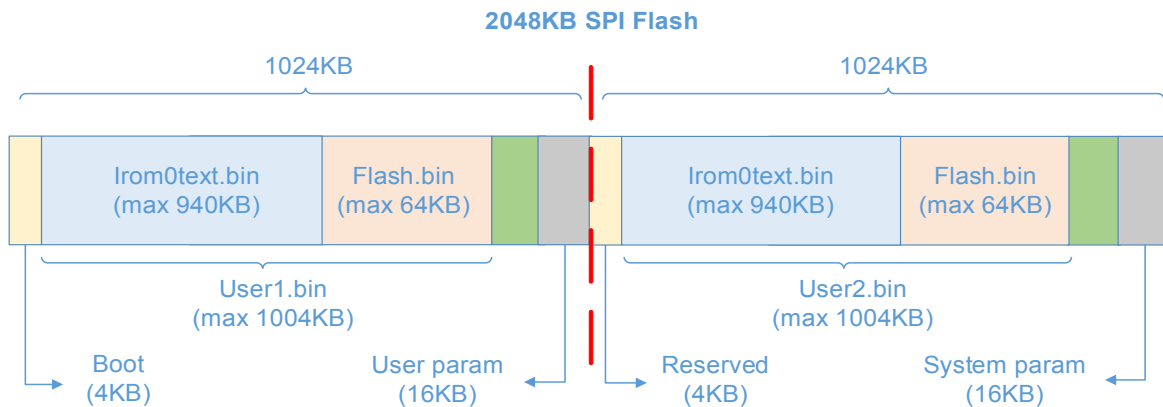


### 2. 2048KB flash

If choose 1 boot\_v1.2+ in STEP 1, choose 3 2048KB in STEP 5, only the first 1024KB to be the program area ( 512KB + 512KB ), flash map will be as below



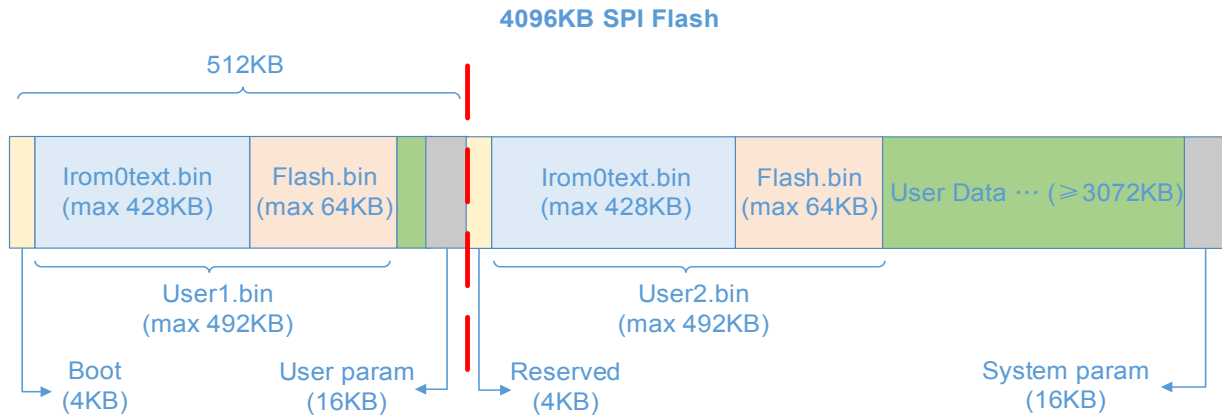
If choose 1 boot\_v1.2+ in STEP 1, choose 5 2048KB in STEP 5 (only be supported since sdk\_v1.1.0 + boot 1.4 and later version), the program area is 1024KB + 1024KB, flash map will be as below



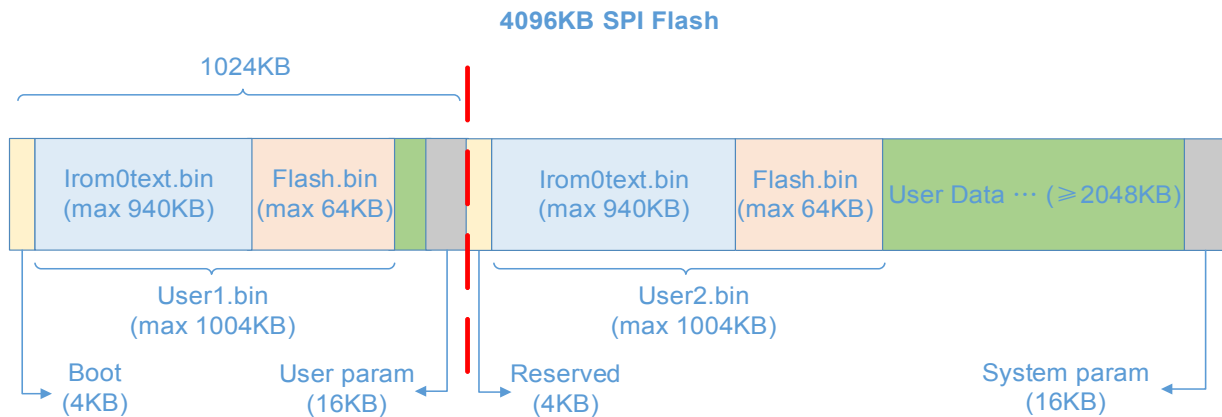


### 3. 4096KB flash

If choose 1 boot\_v1.2+ in STEP 1, choose 4 4096KB in STEP 5, only the first 1024KB to be the program area ( 512KB + 512KB ), flash map will be as below



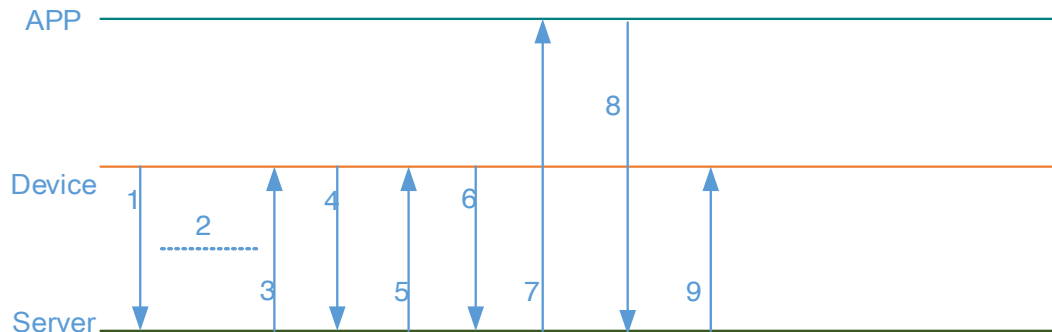
If choose 1 boot\_v1.2+ in STEP 1, choose 6 4096KB in STEP 5 (only be supported since sdk\_v1.1.0 + boot 1.4 and later version), the first 2048KB to be the program area (1024KB + 1024KB), flash map will be as below





### 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 than 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 can be encrypted by SSL. Device key is required from Step4. Therefore upgrade procedure is secure.

Note:

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 `"\ESP8266_NONOS_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 `"\ESP8266_NONOS_SDK\bin\upgrade"`

Note:

**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
<b>master_device_key.bin</b>	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.
<b>blank.bin</b>	0x7E000	Stores default system parameter values, provided in SDK
<b>boot.bin</b>	0x00000	Boot loader, provided in SDK, recommend to use the latest version
<b>user1.bin</b>	0x01000	Compiled by the steps said above

More details about burning into flash are in documentation `"2A-ESP8266-SDK_Getting_Started_Guide"`

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.

The screenshot shows the 'Product' management interface. At the top, there is a search bar with the text 'product name, serial, desc' and a dropdown menu for selecting a product. Below this, two product cards are displayed. Each card includes an icon, a list of attributes (Id, Name, Serial, Status, Description), and a progress bar for 'Activated / Total'.

Product Name	Id	Status	Activated / Total	Progress
dev-controller	15	developing	7 / 7	100%
humiture	28	developing	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”.

## 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

The screenshot shows the 'ROM Deploy' form with the following elements and annotations:

- version** input field containing 'v1.0.1t23701(a)' with a red box and the number '1' next to it.
- A dropdown menu showing 'beta' with a downward arrow.
- corename** input field containing 'custom codename'.
- A section titled 'upload rom files, support max 10 files' with a red box around the '+' icon and the number '3' next to it.
- Two file selection rows: '选择文件 user1.bin' with a red box and the number '2' next to it, and '选择文件 user2.bin' with a red box and the number '4' next to it.
- A 'Save' button with a red box and the number '5' next to it, and a 'Cancel' button next to it.

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.

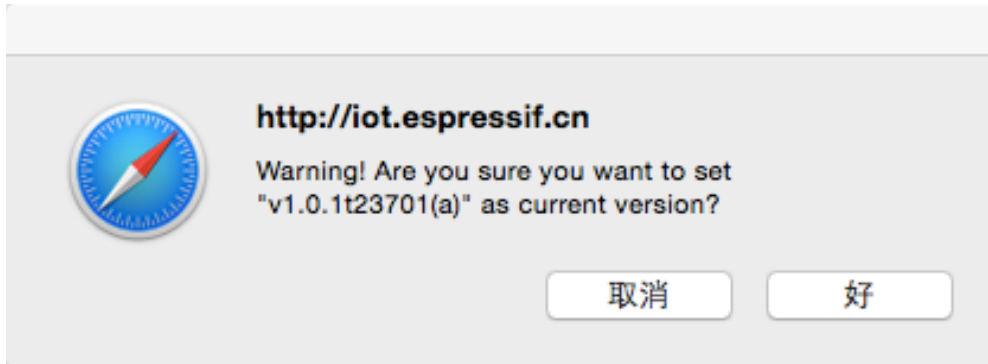
### ROM Deploy

The screenshot shows the 'ROM Deploy' page after saving, with the following elements:

- The version 'v1.0.1t23701(a)' is displayed in bold, followed by a 'Set as Current' button.
- The text 'chore(beta): v1.0.1t23701(a)' is shown below.
- Two file icons with labels 'user1.bin' and 'user2.bin' are listed.

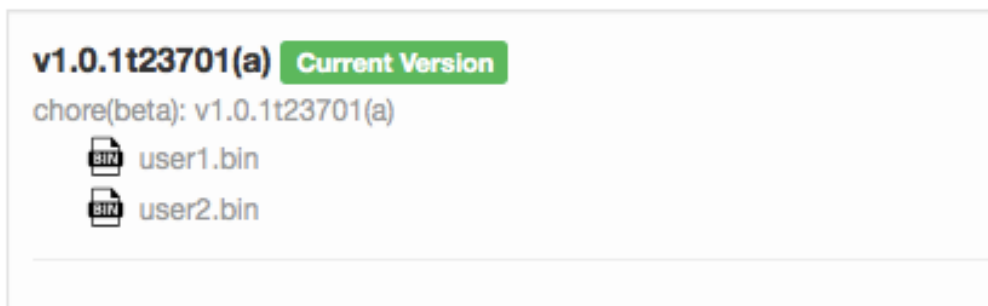


(5) Click “OK” to update the firmware.



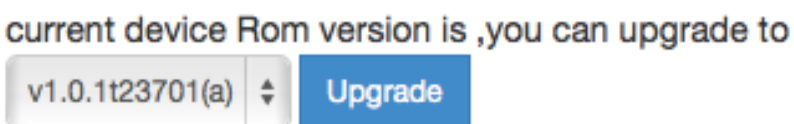
(6) Users will get a message of firmware update.

## ROM Deploy



(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



(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

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

`action=sys_reboot`

Request

- (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 [ESP8266\\_NONOS\\_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

<b>Function:</b>	<p>Check which firmware is running now, user1 or user2</p> <p>If running <code>user1.bin</code> right now, we will download <code>user2.bin</code> from server to update.</p> <p>If running <code>user2.bin</code> right now, we will download <code>user1.bin</code> from server to update.</p>
<b>Prototype:</b>	<code>uint8 system_upgrade_userbin_check()</code>



<b>Return:</b>	<code>0x00</code> : UPGRADE_FW_BIN1 , means <code>user1.bin</code> <code>0x01</code> : UPGRADE_FW_BIN2 , means <code>user2.bin</code>
----------------	--

### 2. system\_upgrade\_start

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

### 3. system\_upgrade\_reboot

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

### 4. system\_upgrade\_flag\_set

<b>Function:</b>	Sets upgrade status flag.
<b>Note:</b>	If you using <code>system_upgrade_start</code> to upgrade, this API need not be called. If you using <code>spi_flash_write</code> to upgrade firmware yourself, this flag need to be set to <code>UPGRADE_FLAG_FINISH</code> , then call <code>system_upgrade_reboot</code> to reboot to run new firmware.
<b>Prototype:</b>	<code>void system_upgrade_flag_set(uint8 flag)</code>
<b>Parameter:</b>	<code>uint8 flag</code> : <code>#define UPGRADE_FLAG_IDLE</code> <code>0x00</code> <code>#define UPGRADE_FLAG_START</code> <code>0x01</code> <code>#define UPGRADE_FLAG_FINISH</code> <code>0x02</code>
<b>Return:</b>	none





### 5. system\_upgrade\_flag\_check

<b>Function:</b>	Gets upgrade status flag.
<b>Prototype:</b>	<code>uint8 system_upgrade_flag_check()</code>
<b>Return:</b>	<code>#define UPGRADE_FLAG_IDLE 0x00</code> <code>#define UPGRADE_FLAG_START 0x01</code> <code>#define UPGRADE_FLAG_FINISH 0x02</code>

### 6. user\_esp\_platform\_upgrade\_begin

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

### 7. user\_esp\_platform\_upgrade\_rsp

<b>Function:</b>	Upgrade callback. Register as a callback in <code>user_esp_platform_upgrade_begin</code> . Enter this callback, if upgrade timeout (check_times) or finish.
<b>Prototype:</b>	<code>esp_platform_upgrade_rsp(void *arg)</code>
<b>Parameter:</b>	<code>void *arg</code> - data pointer
<b>Return:</b>	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		<pre>{   "products": [     {       "name": "name",       "description": "description",       "serial": "serial",       "is_private": 1,       "ptype": 27388,       "status": 1     }   ] }</pre>

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：

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.