ESP-AT User Guide

Read the Docs

Jun 18, 2021

Contents

[]

This is the documentation for the ESP-AT.

Get Started	AT Binary Lists	AT Command Set
AT Command Examples	Compile and Develop	Customized AT Commands and Firmware

CHAPTER 1

Get Started

[]

This Get Started guide provides users with detailed information on what is ESP-AT, how to connect hardware, and how to download and flash AT firmware. It consists of the following parts:

1.1 What is ESP-AT

[]

ESP-AT is a solution developed by Espressif to integrate connectivity into customers' products, which can be quickly moved to mass production. It aims to reduce software development costs and quickly form products. With ESP-AT commands, you can quickly join the wireless network, connect to the cloud platform, realize data transmission and remote control functions, and realize the interconnection of everything through wireless communication easily.

ESP-AT is a project based on ESP-IDF or ESP8266_RTOS_SDK. It makes an ESP board work as a slave, and an MCU as a host. The host MCU sends AT commands to the ESP chip and receives AT responses back. ESP-AT provides a wide range of AT commands with different functions, such as Wi-Fi commands, TCP/IP commands, Bluetooth LE commands, Bluetooth commands, MQTT commands, HTTP commands, and Ethernet commands.

Note: Currently, ESP-AT is based on ESP-IDF or ESP8266_RTOS_SDK, not ESP8266 NonOS SDK.

AT commands start with "AT", which stand for "Attention", and end with a new line (CR LF). Every time you send a command, you will reveive an `OK` or `ERROR`, which indicates the final execution status of the current command. Please be noted that all commands are executed serially, which means only one AT command can be executed at a time. Therefore, you should wait for the previous command to be executed before sending out the next one. Otherwise, you will receive `busy P...`. For more details about AT commands, please refer to *AT Command Set*.

By default, the host MCU connects to the ESP board via UART, and sends/receives AT commands/responses through UART. But you can also use other interfaces, such as SDIO, according to your actual use scenario.

You can develop your own AT commands based on our ESP-AT project and implement more features according to your actual use scenario.

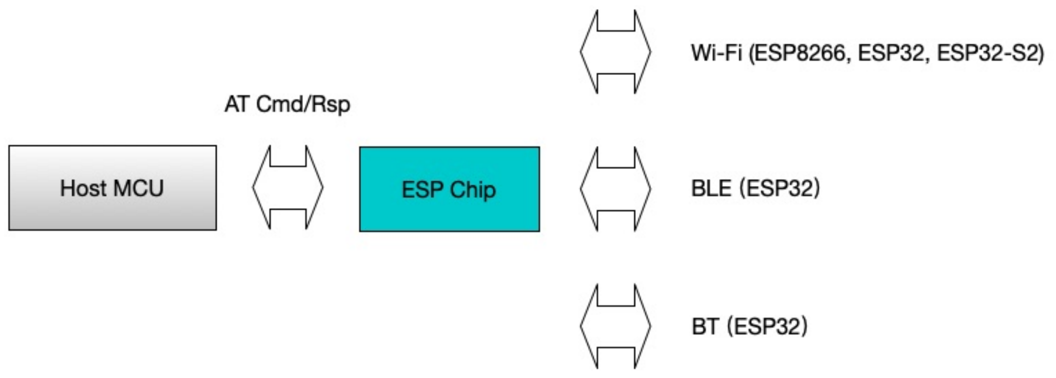


Fig. 1: ESP-AT Overview

1.2 Hardware Connection

[]

This document introduces what hardware you need to prepare and how to connect them in order to download AT firmware, send AT commands, and receive AT responses. It covers the following four ESP series of modules:

- *ESP32 Series*
- *ESP32-S2 Series*
- *ESP32-C3 Series*
- *ESP8266 Series*

For different series of modules, the commands supported by AT firmware are different. Please refer to *How to understand the differences of each type of module* for more details.

1.2.1 What You Need

Table 1: List of Components Required for ESP-AT Testing

Component	Function
ESP board	Slave MCU.
USB cable (ESP borad to PC)	Download/Log output connection.
PC	Act as Host MCU. Download firmware to Slave MCU.
USB cable (PC to serial port converter)	AT command/response connection.
USB to serial port converter	Convert between USB signals and TTL signals.
Jumper wires (serial port converter to ESP board)	AT command/response connection.

Please note that in the above picture, four jump wires are used to connect the ESP board and USB to serial converter. If you don't use hardware flow control, two wires connecting TX/RX and a simpler converter will be enough.

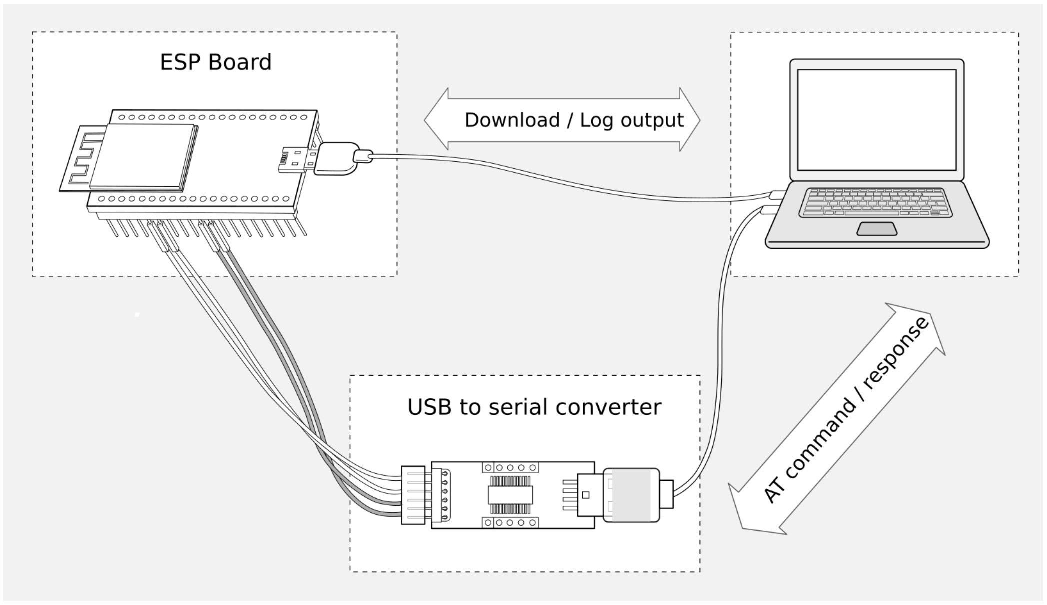


Fig. 2: Connection of Components for ESP-AT Testing

1.2.2 ESP32 Series

ESP32 AT uses two UART ports: UART0 is used to download firmware and log output; UART1 is used to send AT commands and receive AT responses.

All ESP32 modules use GPIO1 and GPIO3 as UART0, but they use different GPIOs as UART1. The following sections illustrate which GPIOs you should connect for each ESP32 series of modules.

For more details of ESP32 modules and boards, please refer to ESP32 Modules and Boards.

ESP32-WROOM-32 Series

Table 2: ESP32-WROOM-32 Series Hardware Connection Pinout

Function of Connection	ESP Board Pins	Other Device Pins
Download/Log output [1]	**UART0** • GPIO3 (RX) • GPIO1 (TX)	**PC** • TX • RX
AT command/response [2]	**UART1** • GPIO16 (RX) • GPIO17 (TX) • GPIO15 (CTS) • GPIO14 (RTS)	**USB to serial converter** • TX • RX • RTS • CTS

Note 1: Connection between individual pins of the ESP board and the PC is already established internally on the ESP board. You only need to provide USB cable between the board and PC.

Note 2: Connection between CTS/RTS is optional, depending on whether you want to use hardware flow control.

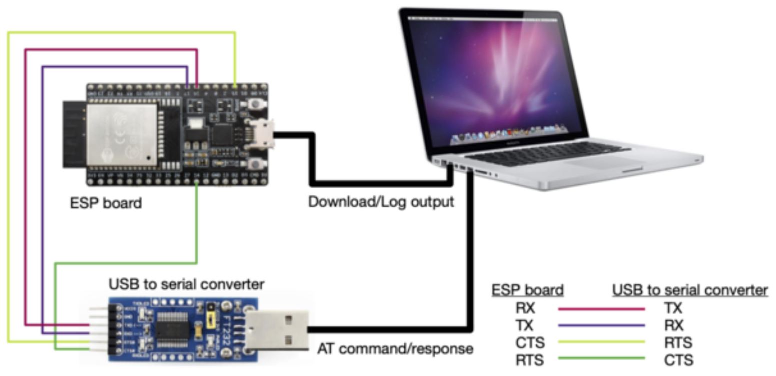


Fig. 3: ESP32-WROOM-32 Series Hardware Connection

If you want to connect your device directly with ESP32-WROOM-32 rather than the ESP board that integrates it, please refer to ESP32-WROOM-32 Datasheet for more details.

ESP32-WROVER Series

Table 3: ESP32-WROVER Series Hardware Connection Pinout

Function of Connection	ESP Board Pins	Other Device Pins
Download/Log output [1]	**UART0** • GPIO3 (RX) • GPIO1 (TX)	**PC** • TX • RX
AT command/response [2]	**UART1** • GPIO19 (RX) • GPIO22 (TX) • GPIO15 (CTS) • GPIO14 (RTS)	**USB to serial converter** • TX • RX • RTS • CTS

Note 1: Connection between individual pins of the ESP board and the PC is already established internally on the ESP board. You only need to provide USB cable between the board and PC.

Note 2: Connection between CTS/RTS is optional, depending on whether you want to use hardware flow control.

If you want to connect your device directly with ESP32-WROVER rather than the ESP board that integrates it, please refer to ESP32-WROVER Datasheet for more details.

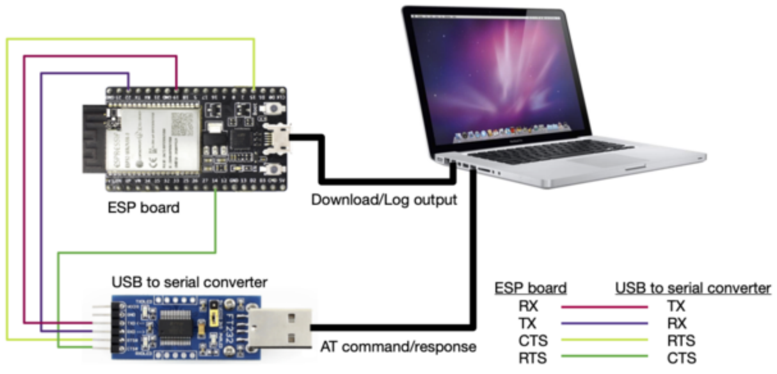


Fig. 4: ESP32-WROVER Series Hardware Connection

ESP32-PICO Series

Table 4: ESP32-PICO Series Hardware Connection Pinout

Function of Connection	ESP Board Pins	Other Device Pins
Download/Log output [1]	**UART0** • GPIO3 (RX) • GPIO1 (TX)	**PC** • TX • RX
AT command/response [2]	**UART1** • GPIO19 (RX) • GPIO22 (TX) • GPIO15 (CTS) • GPIO14 (RTS)	**USB to serial converter** • TX • RX • RTS • CTS

Note 1: Connection between individual pins of the ESP board and the PC is already established internally on the ESP board. You only need to provide USB cable between the board and PC.

Note 2: Connection between CTS/RTS is optional, depending on whether you want to use hardware flow control.

If you want to connect your device directly with ESP32-PICO-D4 rather than the ESP board that integrates it, please refer to ESP32-PICO-D4 Datasheet for more details.

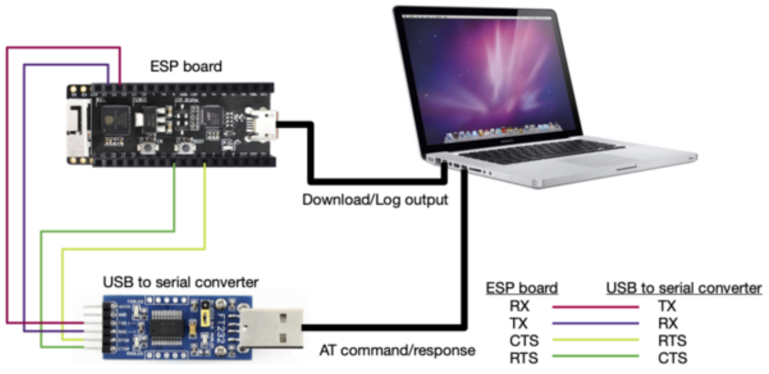


Fig. 5: ESP32-PICO Series Hardware Connection

ESP32-SOLO Series

Table 5: ESP32-SOLO Series Hardware Connection Pinout

Function of Connection	ESP Board Pins	Other Device Pins
Download/Log output [1]	**UART0** • GPIO3 (RX) • GPIO1 (TX)	**PC** • TX • RX
AT command/response [2]	**UART1** • GPIO16 (RX) • GPIO17 (TX) • GPIO15 (CTS) • GPIO14 (RTS)	**USB to serial converter** • TX • RX • RTS • CTS

Note 1: Connection between individual pins of the ESP board and the PC is already established internally on the ESP board. You only need to provide USB cable between the board and PC.

Note 2: Connection between CTS/RTS is optional, depending on whether you want to use hardware flow control.

If you want to connect your device directly with ESP32-SOLO-1 rather than the ESP board that integrates it, please refer to ESP32-SOLO-1 Datasheet for more details.

1.2.3 ESP32-S2 Series

ESP32-S2 AT uses two UART ports: UART0 is used to download firmware and log output; UART1 is used to send AT commands and receive AT responses.

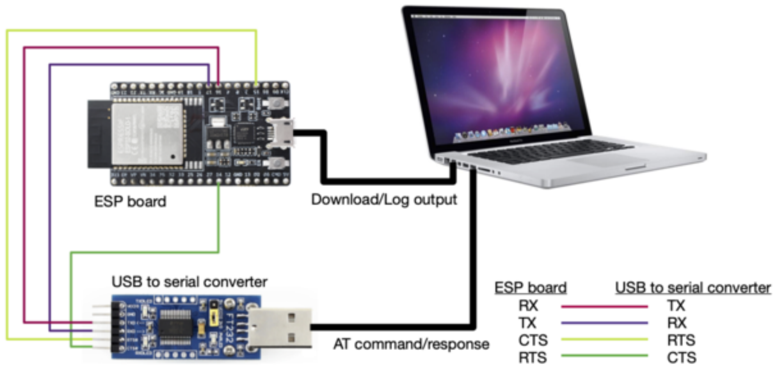


Fig. 6: ESP32-SOLO Series Hardware Connection

Table 6: ESP32-S2 Series Hardware Connection Pinout

Function of Connection	ESP Board Pins	Other Device Pins
Download/Log output [1]	**UART0** • GPIO44 (RX) • GPIO43 (TX)	**PC** • TX • RX
AT command/response [2]	**UART1** • GPIO21 (RX) • GPIO17 (TX) • GPIO20 (CTS) • GPIO19 (RTS)	**USB to serial converter** • TX • RX • RTS • CTS

Note 1: Connection between individual pins of the ESP board and the PC is already established internally on the ESP board. You only need to provide USB cable between the board and PC.

Note 2: Connection between CTS/RTS is optional, depending on whether you want to use hardware flow control.

If you want to connect your device directly with ESP32-S2-WROOM rather than the ESP board that integrates it, please refer to ESP32-S2-WROOM & ESP32-S2-WROOM-I Datasheet for more details.

1.2.4 ESP32-C3 Series

ESP32-C3 AT uses two UART ports: UART0 is used to download firmware and log output; UART1 is used to send AT commands and receive AT responses.

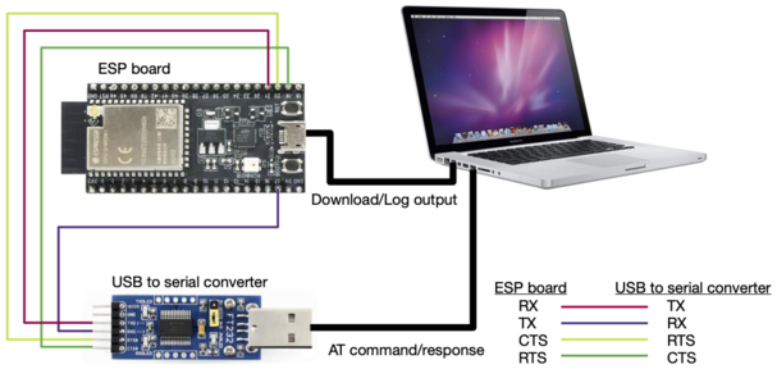


Fig. 7: ESP32-S2 Series Hardware Connection

Table 7: ESP32-C3 Series Hardware Connection Pinout

Function of Connection	ESP Board Pins	Other Device Pins
Download/Log output [1]	**UART0** • GPIO20 (RX) • GPIO21 (TX)	**PC** • TX • RX
AT command/response [2]	**UART1** • GPIO6 (RX) • GPIO7 (TX) • GPIO5 (CTS) • GPIO4 (RTS)	**USB to serial converter** • TX • RX • RTS • CTS

Note 1: Connection between individual pins of the ESP board and the PC is already established internally on the ESP board. You only need to provide USB cable between the board and PC.

Note 2: Connection between CTS/RTS is optional, depending on whether you want to use hardware flow control.

If you want to connect your device directly with ESP32-C3-MINI-1 rather than the ESP board that integrates it, please refer to ESP32-C3-MINI-1 Datasheet for more details.

1.2.5 ESP8266 Series

ESP8266 AT uses two UART ports: UART0 is used to download firmware and send AT commands and receive AT responses; UART1 is used to log output.

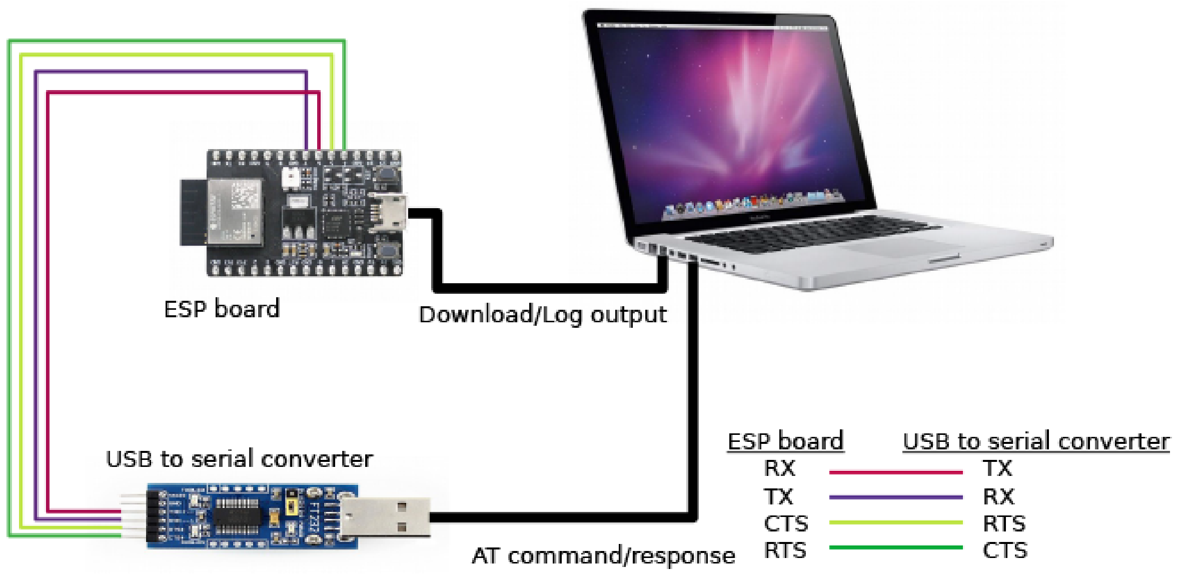


Fig. 8: ESP32-C3 Series Hardware Connection

Table 8: ESP8266 Series Hardware Connection Pinout

Function of Connection	ESP Board Pins	Other Device Pins
Download	**UART0** • GPIO3 (RX) • GPIO1 (TX)	**PC** • TX • RX
AT command/response [2]	**UART0** • GPIO13 (RX) • GPIO15 (TX) • GPIO3 (CTS) • GPIO1 (RTS)	**USB to serial converter** • TX • RX • RTS • CTS
Log output	**UART1** • GPIO2 (TX)	**USB to serial converter** • RX

Note 1: Connection between individual pins of the ESP board and the PC is already established internally on the ESP board. You only need to provide USB cable between the board and PC.

Note 2: Connection between CTS/RTS is optional, depending on whether you want to use hardware flow control.

Note: The default ESP8266 RTOS AT firmware for ESP-WROOM-02 swaps RX/TX with CTS/RTS. If you want to use hardware flow control, you need to disconnect UART1, desolder CP2102N chip from the ESP board, and connect the board with 3.3 V and GND of the converter to supply power.

If you want to connect your device directly with ESP-WROOM-02 or ESP-WROOM-02D/02U rather than the ESP

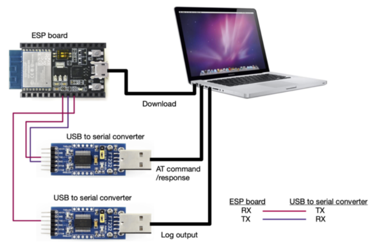


Fig. 9: ESP8266 Series Hardware Connection

board that integrates it, please refer to ESP-WROOM-02 Datasheet or ESP-WROOM-02D/02U Datasheet for more details.

For more details about ESP8266 modules, please refer to ESP8266 documentation.

1.3 Downloading Guide

[]

This Guide demonstrates how to download AT firmware and flash it into an ESP device by taking ESP32-WROOM-32 as an example. The Guide is also applicable to other ESP modules.

Before you start, please make sure you have already connected your hardware. For more details, see *Hardware Connection*.

For different series of modules, the commands supported by AT firmware are different. Please refer to *How to understand the differences of each type of module* for more details.

1.3.1 Download AT Firmware

To download AT firmware to your computer, please do as follows:

- Navigate to *AT Binary Lists*
- Find the firmware for your device
- Click the link to download it

Here, we download `ESP32-WROOM-32_AT_Bin_V2.1` for ESP32-WROOM-32. The list below describes the structure of this firmware and what each bin file contains. Other AT firmware has similar structure and bin files.

```
.
├── at_customize.bin                // secondary partition table
├── bootloader                      // bootloader
│   └── bootloader.bin
├── customized_partitions           // AT customized binaries
│   ├── ble_data.bin
│   ├── client_ca.bin
│   ├── client_cert.bin
│   ├── client_key.bin
│   ├── factory_param.bin
│   ├── factory_param_WROOM-32.bin
│   ├── mqtt_ca.bin
```

(continues on next page)

(continued from previous page)

```
│    ├── mqtt_cert.bin
│    ├── mqtt_key.bin
│    ├── server_ca.bin
│    ├── server_cert.bin
│    └── server_key.bin
├── download.config                  // configuration of downloading
├── esp-at.bin                       // AT application binary
├── factory                          // A combined bin for factory downloading
│    ├── factory_WROOM-32.bin
│    └── factory_parameter.log
├── flasher_args.json                // flasher arguments
├── ota_data_initial.bin             // ota data parameters
├── partition_table                  // primary partition table
│    └── partition-table.bin
└── phy_init_data.bin                // phy parameters
```

The file `download.config` contains the configuration to flash the firmware into multiple addresses:

```
--flash_mode dio --flash_freq 40m --flash_size 4MB
0x8000 partition_table/partition-table.bin
0x10000 ota_data_initial.bin
0xf000 phy_init_data.bin
0x1000 bootloader/bootloader.bin
0x100000 esp-at.bin
0x20000 at_customize.bin
0x24000 customized_partitions/server_cert.bin
0x39000 customized_partitions/mqtt_key.bin
0x26000 customized_partitions/server_key.bin
0x28000 customized_partitions/server_ca.bin
0x2e000 customized_partitions/client_ca.bin
0x30000 customized_partitions/factory_param.bin
0x21000 customized_partitions/ble_data.bin
0x3B000 customized_partitions/mqtt_ca.bin
0x37000 customized_partitions/mqtt_cert.bin
0x2a000 customized_partitions/client_cert.bin
0x2c000 customized_partitions/client_key.bin
```

- `--flash_mode dio` means the firmware is compiled with flash DIO mode.
- `--flash_freq 40m` means the firmware's flash frequency is 40 MHz.
- `--flash_size 4MB` means the firmware is using flash size 4 MB.
- `0x10000 ota_data_initial.bin` means downloading `ota_data_initial.bin` into the address `0x10000`.

1.3.2 Flash AT Firmware into Your Device

Follow the instructions below for your operating system.

Windows

Before starting to flash, you need to download Flash Download Tools for Windows. For more details about the tools, please see `readme.pdf` or the `doc` folder in the zip folder.

- Open the ESP Flash Download Tool.

- Select a mode according to your need. (Here, we select `Developer Mode`.)

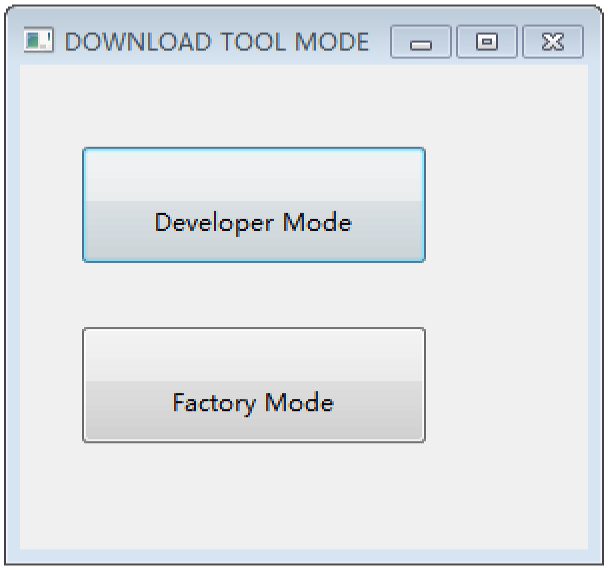


Fig. 10: Flash Download Tools Modes

- Select your target chip. For example, choose "ESP8266 DownloadTool" for ESP8266 chip; choose "ESP32-S2 DownloadTool" for ESP32-S2 chip. (Here, we select ESP32 DownloadTool.)
- Flash AT firmware into your device. You can select either of the two ways below.
  - To download one combined factory bin to address 0, select "DoNotChgBin" to use the default configuration of the factory bin.
  - To download multiple bins separately to different addresses, set up the configurations according to the file `download.config` and do NOT select "DoNotChgBin".

In case of flashing issues, please verify what the COM port number of download interface of the ESP board is and select it from "COM:" dropdown list. If you don't know the port number, you can refer to Check port on Windows for details.

When you finish flashing, please *Check Whether AT Works*.

Linux or macOS

Before you start to flash, you need to install esptool.py.

You can select either of the two ways below to flash AT firmware into your device.

- To download the bins separately into multiple addresses, enter the following command and replace `PORTNAME` and `download.config`:

```
esptool.py --chip auto --port PORTNAME --baud 115200 --before default_reset --
↪after hard_reset write_flash -z download.config
```

  Replace `PORTNAME` with your port name. If you don't know it, you can refer to Check port on Linux and macOS for details.

  Replace `download.config` with the content inside the file.

  Below is the example command for ESP32-WROOM-32.

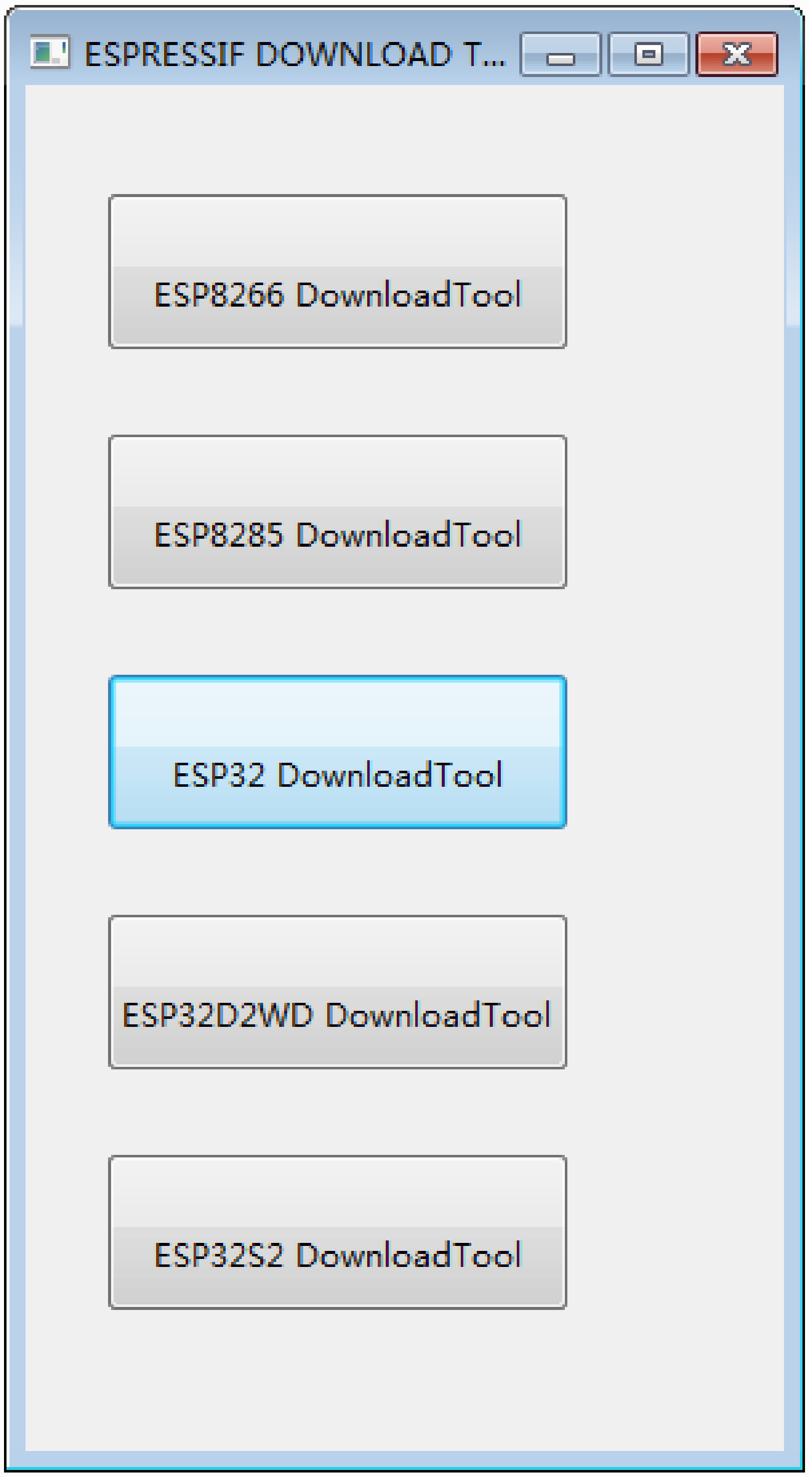
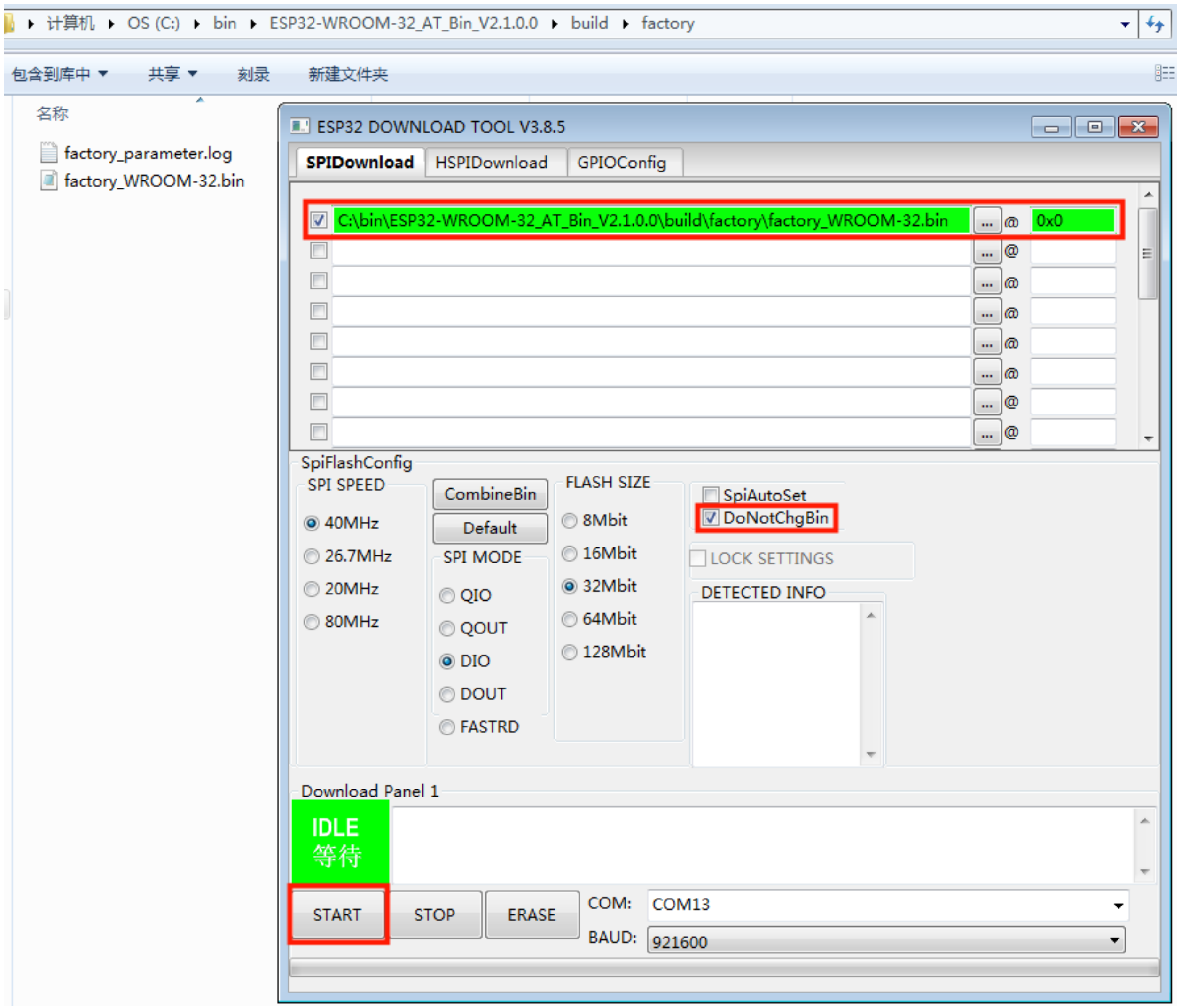


Fig. 11: Flash Download Tools Target Chip

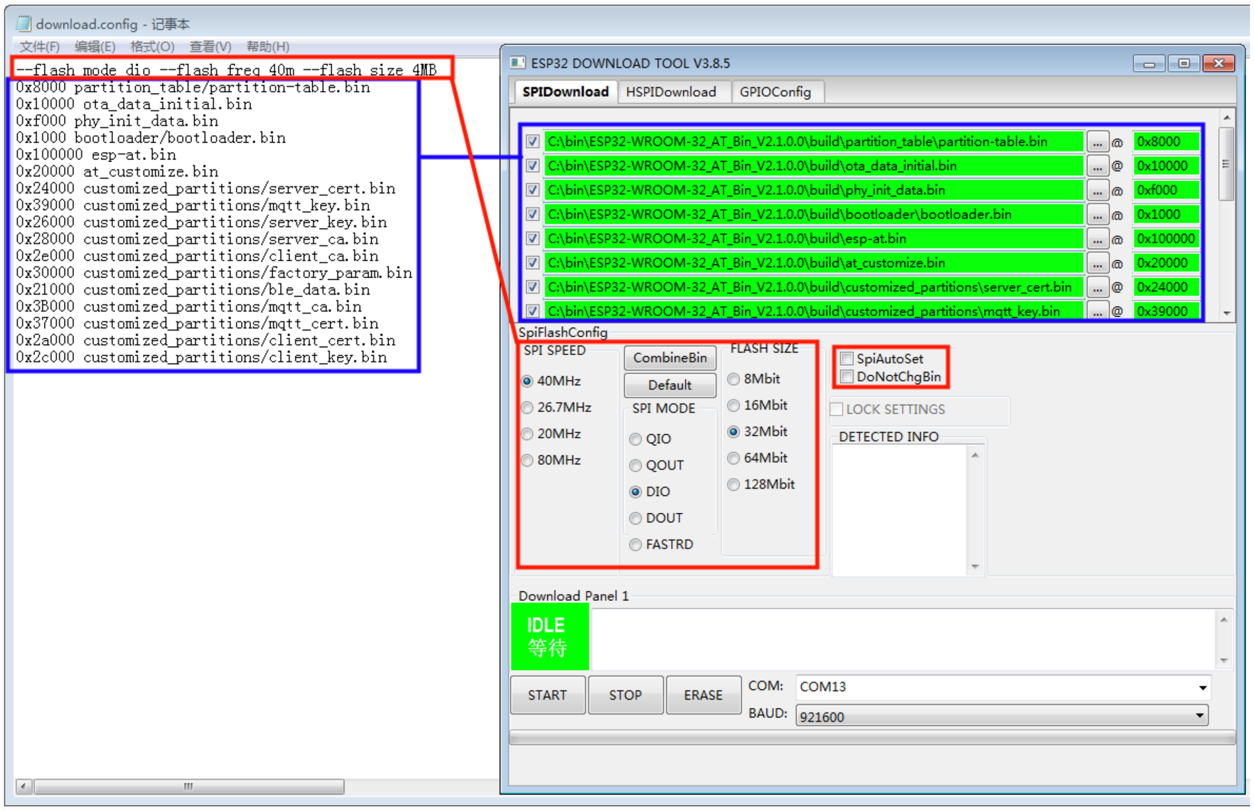

计算机 ▸ OS (C:) ▸ bin ▸ ESP32-WROOM-32_AT_Bin_V2.1.0.0 ▸ build ▸ factory
包含到库中 共享 刻录 新建文件夹
名称
factory_parameter.log
factory_WROOM-32.bin
ESP32 DOWNLOAD TOOL V3.8.5
SPIDownload HSPIDownload GPIOConfig
CombineBin
Default
IDLE 等待
START STOP ERASE

☑ C:\bin\ESP32-WROOM-32_AT_Bin_V2.1.0.0\build\factory\factory_WROOM-32.bin @ 0x0

SpiFlashConfig
SPI SPEED: [● 40MHz] 26.7MHz 20MHz 80MHz
SPI MODE: QIO QOUT [● DIO] DOUT FASTRD
FLASH SIZE: 8Mbit 16Mbit [● 32Mbit] 64Mbit 128Mbit
☐ SpiAutoSet
☑ DoNotChgBin
☐ LOCK SETTINGS
DETECTED INFO

Download Panel 1
COM: COM13
BAUD: 921600

Fig. 12: Download to One Address



```
--flash_mode dio --flash_freq 40m --flash_size 4MB
0x8000 partition_table/partition-table.bin
0x10000 ota_data_initial.bin
0xf000 phy_init_data.bin
0x1000 bootloader/bootloader.bin
0x100000 esp-at.bin
0x20000 at_customize.bin
0x24000 customized_partitions/server_cert.bin
0x39000 customized_partitions/mqtt_key.bin
0x26000 customized_partitions/server_key.bin
0x28000 customized_partitions/server_ca.bin
0x2e000 customized_partitions/client_ca.bin
0x30000 customized_partitions/factory_param.bin
0x21000 customized_partitions/ble_data.bin
0x3B000 customized_partitions/mqtt_ca.bin
0x37000 customized_partitions/mqtt_cert.bin
0x2a000 customized_partitions/client_cert.bin
0x2c000 customized_partitions/client_key.bin
```

```
☑ C:\bin\ESP32-WROOM-32_AT_Bin_V2.1.0.0\build\partition_table\partition-table.bin @ 0x8000
☑ C:\bin\ESP32-WROOM-32_AT_Bin_V2.1.0.0\build\ota_data_initial.bin @ 0x10000
☑ C:\bin\ESP32-WROOM-32_AT_Bin_V2.1.0.0\build\phy_init_data.bin @ 0xf000
☑ C:\bin\ESP32-WROOM-32_AT_Bin_V2.1.0.0\build\bootloader\bootloader.bin @ 0x1000
☑ C:\bin\ESP32-WROOM-32_AT_Bin_V2.1.0.0\build\esp-at.bin @ 0x100000
☑ C:\bin\ESP32-WROOM-32_AT_Bin_V2.1.0.0\build\at_customize.bin @ 0x20000
☑ C:\bin\ESP32-WROOM-32_AT_Bin_V2.1.0.0\build\customized_partitions\server_cert.bin @ 0x24000
☑ C:\bin\ESP32-WROOM-32_AT_Bin_V2.1.0.0\build\customized_partitions\mqtt_key.bin @ 0x39000
```

SpiFlashConfig

SPI SPEED: [✓ 40MHz] 26.7MHz 20MHz 80MHz

SPI MODE: QIO QOUT [✓ DIO] DOUT FASTRD

FLASH SIZE: 8Mbit 16Mbit [✓ 32Mbit] 64Mbit 128Mbit

☐ SpiAutoSet
☐ DoNotChgBin
☐ LOCK SETTINGS

COM: COM13
BAUD: 921600

Fig. 13: Download to Multiple Addresses

```
esptool.py --chip auto --port /dev/tty.usbserial-0001 --baud 115200 --before␣
↪default_reset --after hard_reset write_flash -z --flash_mode dio --flash_freq␣
↪40m --flash_size 4MB 0x8000 partition_table/partition-table.bin 0x10000 ota_
↪data_initial.bin 0xf000 phy_init_data.bin 0x1000 bootloader/bootloader.bin␣
↪0x100000 esp-at.bin 0x20000 at_customize.bin 0x24000 customized_partitions/
↪server_cert.bin 0x39000 customized_partitions/mqtt_key.bin 0x26000 customized_
↪partitions/server_key.bin 0x28000 customized_partitions/server_ca.bin 0x2e000␣
↪customized_partitions/client_ca.bin 0x30000 customized_partitions/factory_param.
↪bin 0x21000 customized_partitions/ble_data.bin 0x3B000 customized_partitions/
↪mqtt_ca.bin 0x37000 customized_partitions/mqtt_cert.bin 0x2a000 customized_
↪partitions/client_cert.bin 0x2c000 customized_partitions/client_key.bin
```

- To download the bins together to one address, enter the following command and replace `PORTNAME` and `FILEDIRECTORY`:

```
esptool.py --chip auto --port PORTNAME --baud 115200 --before default_reset --
↪after hard_reset write_flash -z --flash_mode dio --flash_freq 40m --flash_size␣
↪4MB 0x0 FILEDIRECTORY
```

  Replace `PORTNAME` with your port name. If you don't know it, you can refer to Check port on Linux and macOS for details.

  Replace `FILEDIRECTORY` with the file directory you would flash to the address `0x0`. It is normally factory/XXX.bin.

  Below is the example command for ESP32-WROOM-32.

```
esptool.py --chip auto --port /dev/tty.usbserial-0001 --baud 115200 --before␣
↪default_reset --after hard_reset write_flash -z --flash_mode dio --flash_freq␣
↪40m --flash_size 4MB 0x0 factory/factory_WROOM-32.bin
```

When you finish flashing, please *Check Whether AT Works*.

1.3.3 Check Whether AT Works

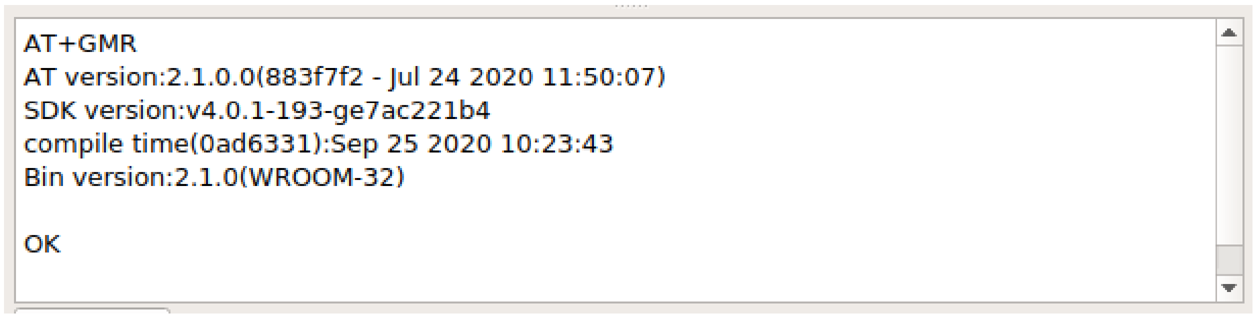
To check whether AT works, do as follows:

- Open a serial port tool, such as SecureCRT;
- Select the Port attached to "AT command/response" line (see *Hardware Connection* for details);
- Set Baudrate to 115200;
- Set Data Bits to 8;
- Set Parity to None;
- Set Stop Bits to 1;
- Set Flow Type to None;
- Enter the command "AT+GMR" with a new line (CR LF).

If the response is OK as the picture below shows, it means that AT works.

Otherwise, you need to check your ESP startup log, which is visible on PC over "Download/Log output connection". If it is like the log below, it means that ESP-AT firmware have been initalized correctly.

ESP32 startup log:



```
AT+GMR
AT version:2.1.0.0(883f7f2 - Jul 24 2020 11:50:07)
SDK version:v4.0.1-193-ge7ac221b4
compile time(0ad6331):Sep 25 2020 10:23:43
Bin version:2.1.0(WROOM-32)

OK
```

Fig. 14: Response from AT

```
ets Jun  8 2016 00:22:57
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0030,len:4
load:0x3fff0034,len:7184
ho 0 tail 12 room 4
load:0x40078000,len:13200
load:0x40080400,len:4564
entry 0x400806f4
I (30) boot: ESP-IDF v4.2 2nd stage bootloader
I (31) boot: compile time 11:23:19
I (31) boot: chip revision: 0
I (33) boot.esp32: SPI Speed      : 40MHz
I (38) boot.esp32: SPI Mode       : DIO
I (42) boot.esp32: SPI Flash Size : 4MB
I (47) boot: Enabling RNG early entropy source...
I (52) boot: Partition Table:
I (56) boot: ## Label            Usage          Type ST Offset   Length
I (63) boot:  0 phy_init         RF data          01 01 0000f000 00001000
I (71) boot:  1 otadata          OTA data         01 00 00010000 00002000
I (78) boot:  2 nvs              WiFi data        01 02 00012000 0000e000
I (86) boot:  3 at_customize     unknown          40 00 00020000 000e0000
I (93) boot:  4 ota_0            OTA app          00 10 00100000 00180000
I (101) boot:  5 ota_1            OTA app          00 11 00280000 00180000
I (108) boot: End of partition table
I (112) esp_image: segment 0: paddr=0x00100020 vaddr=0x3f400020 size=0x2a300 (172800)␣
↪map
I (187) esp_image: segment 1: paddr=0x0012a328 vaddr=0x3ffbdb60 size=0x039e8 ( 14824)␣
↪load
I (194) esp_image: segment 2: paddr=0x0012dd18 vaddr=0x40080000 size=0x00404 (  1028)␣
↪load
I (194) esp_image: segment 3: paddr=0x0012e124 vaddr=0x40080404 size=0x01ef4 (  7924)␣
↪load
I (206) esp_image: segment 4: paddr=0x00130020 vaddr=0x400d0020 size=0x10a470␣
↪(1090672) map
I (627) esp_image: segment 5: paddr=0x0023a498 vaddr=0x400822f8 size=0x1c3a0 (115616)␣
↪load
I (678) esp_image: segment 6: paddr=0x00256840 vaddr=0x400c0000 size=0x00064 (   100)␣
↪load
I (695) boot: Loaded app from partition at offset 0x100000
I (695) boot: Disabling RNG early entropy source...
```

(continues on next page)

(continued from previous page)

```
max tx power=78,ret=0
2.1.0
```

ESP32-S2 startup log:

```
ESP-ROM:esp32s2-rc4-20191025
Build:Oct 25 2019
rst:0x1 (POWERON),boot:0x8 (SPI_FAST_FLASH_BOOT)
SPIWP:0xee
mode:DIO, clock div:1
load:0x3ffe6100,len:0x4
load:0x3ffe6104,len:0x1a24
load:0x4004c000,len:0x1a6c
load:0x40050000,len:0x20fc
entry 0x4004c35c
I (46) boot: ESP-IDF v4.2 2nd stage bootloader
I (46) boot: compile time 11:24:34
I (46) boot: chip revision: 0
I (47) qio_mode: Enabling default flash chip QIO
I (53) boot.esp32s2: SPI Speed      : 80MHz
I (57) boot.esp32s2: SPI Mode       : QIO
I (62) boot.esp32s2: SPI Flash Size : 4MB
I (67) boot: Enabling RNG early entropy source...
I (72) boot: Partition Table:
I (76) boot: ## Label            Usage          Type ST Offset   Length
I (83) boot:  0 phy_init         RF data          01 01 0000f000 00001000
I (91) boot:  1 otadata          OTA data         01 00 00010000 00002000
I (98) boot:  2 nvs              WiFi data        01 02 00012000 0000e000
I (106) boot:  3 at_customize     unknown          40 00 00020000 000e0000
I (113) boot:  4 ota_0            OTA app          00 10 00100000 00180000
I (121) boot:  5 ota_1            OTA app          00 11 00280000 00180000
I (128) boot: End of partition table
I (133) esp_image: segment 0: paddr=0x00100020 vaddr=0x3f000020 size=0x21bec (138220)␣
↪map
I (167) esp_image: segment 1: paddr=0x00121c14 vaddr=0x3ffc9330 size=0x02fe0 ( 12256)␣
↪load
I (169) esp_image: segment 2: paddr=0x00124bfc vaddr=0x40024000 size=0x00404 (  1028)␣
↪load
I (173) esp_image: segment 3: paddr=0x00125008 vaddr=0x40024404 size=0x0b010 ( 45072)␣
↪load
I (193) esp_image: segment 4: paddr=0x00130020 vaddr=0x40080020 size=0xb0784 (722820)␣
↪map
I (324) esp_image: segment 5: paddr=0x001e07ac vaddr=0x4002f414 size=0x09f18 ( 40728)␣
↪load
I (334) esp_image: segment 6: paddr=0x001ea6cc vaddr=0x40070000 size=0x0001c (    28)␣
↪load
I (346) boot: Loaded app from partition at offset 0x100000
I (346) boot: Disabling RNG early entropy source...
max tx power=78,ret=0
2.1.0
```

ESP32-C3 startup log:

```
ESP-ROM:esp32c3-20200918
Build:Sep 18 2020
rst:0x1 (POWERON),boot:0xc (SPI_FAST_FLASH_BOOT)
SPIWP:0xee
```

(continues on next page)

(continued from previous page)

```
mode:DIO, clock div:2
load:0x3fcd6100,len:0x14
load:0x3fcd6114,len:0x179c
load:0x403ce000,len:0x894
load:0x403d0000,len:0x2bf8
entry 0x403ce000
I (54) boot: ESP-IDF v4.3-beta1 2nd stage bootloader
I (55) boot: compile time 12:09:42
I (55) boot: chip revision: 1
I (57) boot_comm: chip revision: 1, min. bootloader chip revision: 0
I (64) boot.esp32c3: SPI Speed      : 40MHz
I (68) boot.esp32c3: SPI Mode       : DIO
I (73) boot.esp32c3: SPI Flash Size : 4MB
I (78) boot: Enabling RNG early entropy source...
I (83) boot: Partition Table:
I (87) boot: ## Label            Usage          Type ST Offset   Length
I (94) boot:  0 phy_init         RF data          01 01 0000f000 00001000
I (102) boot:  1 otadata          OTA data         01 00 00010000 00002000
I (109) boot:  2 nvs              WiFi data        01 02 00012000 0000e000
I (117) boot:  3 at_customize     unknown          40 00 00020000 000e0000
I (124) boot:  4 ota_0            OTA app          00 10 00100000 00180000
I (132) boot:  5 ota_1            OTA app          00 11 00280000 00180000
I (139) boot: End of partition table
I (144) boot: No factory image, trying OTA 0
I (149) boot_comm: chip revision: 1, min. application chip revision: 0
I (156) esp_image: segment 0: paddr=00100020 vaddr=3c140020 size=29cc8h (171208) map
I (201) esp_image: segment 1: paddr=00129cf0 vaddr=3fc8f000 size=03be8h ( 15336) load
I (205) esp_image: segment 2: paddr=0012d8e0 vaddr=40380000 size=02738h ( 10040) load
I (210) esp_image: segment 3: paddr=00130020 vaddr=42000020 size=135bf0h (1268720) map
I (489) esp_image: segment 4: paddr=00265c18 vaddr=40382738 size=0c778h ( 51064) load
I (502) esp_image: segment 5: paddr=00272398 vaddr=50000000 size=00004h (     4) load
I (508) boot: Loaded app from partition at offset 0x100000
I (544) boot: Set actual ota_seq=1 in otadata[0]
I (544) boot: Disabling RNG early entropy source...
max tx power=78,ret=0
2.1.0
```

ESP8266 startup log:

```
...
boot: ESP-IDF v3.4-rc 2nd stage bootloader
I (54) boot: compile time 11:18:21
I (54) boot: SPI Speed      : 80MHz
I (57) boot: SPI Mode       : DIO
I (61) boot: SPI Flash Size : 2MB
I (65) boot: Partition Table:
I (68) boot: ## Label            Usage          Type ST Offset   Length
I (75) boot:  0 otadata          OTA data         01 00 00009000 00002000
I (83) boot:  1 phy_init         RF data          01 01 0000f000 00001000
I (90) boot:  2 ota_0            OTA app          00 10 00010000 000e0000
I (98) boot:  3 at_customize     unknown          40 00 000f0000 00020000
I (105) boot:  4 ota_1            OTA app          00 11 00110000 000e0000
I (112) boot:  5 nvs              WiFi data        01 02 001f0000 00010000
I (120) boot: End of partition table
I (124) boot: No factory image, trying OTA 0
I (129) esp_image: segment 0: paddr=0x00010010 vaddr=0x40210010 size=0xac0d0 (704720)␣
↪map
```

(continues on next page)

(continued from previous page)

```
I (138) esp_image: segment 1: paddr=0x000bc0e8 vaddr=0x402bc0e0 size=0x1aba8 (109480)␣
↪map
I (146) esp_image: segment 2: paddr=0x000d6c98 vaddr=0x3ffe8000 size=0x00788 (  1928)␣
↪load
I (155) esp_image: segment 3: paddr=0x000d7428 vaddr=0x40100000 size=0x00080 (   128)␣
↪load
I (164) esp_image: segment 4: paddr=0x000d74b0 vaddr=0x40100080 size=0x059c4 ( 22980)␣
↪load
I (173) boot: Loaded app from partition at offset 0x10000
phy_version: 1163.0, 665d56c, Jun 24 2020, 10:00:08, RTOS new
max tx power=78,ret=0
2.0.0
```

To learn more about ESP-AT, please read *What is ESP-AT*.

To get started with ESP-AT, please read *Hardware Connection* first to learn what hardware to prepare and how to connect them. Then, you can download and flash AT firmware into your device according to *Downloading Guide*.

CHAPTER 2

AT Binary Lists

[]

2.1 Released Firmware

It is recommended to use the lastest version of firmware.

2.1.1 ESP32-WROOM-32 Series

- v2.1.0.0 ESP32-WROOM-32_AT_Bin_V2.1.0.0.zip (Recommended)
- v2.0.0.0 ESP32-WROOM-32_AT_Bin_V2.0.zip

2.1.2 ESP32-WROVER-32 Series

- v2.1.0.0 ESP32-WROVER_AT_Bin_V2.1.0.0.zip (Recommended)
- v2.0.0.0 ESP32-WROVER_AT_Bin_V2.0.zip

2.1.3 ESP32-PICO Series

- v2.1.0.0 ESP32-PICO-D4_AT_Bin_V2.1.0.0.zip (Recommended)
- v2.0.0.0 ESP32-PICO-D4_AT_Bin_V2.0.zip

2.1.4 ESP32-SOLO Series

- v2.1.0.0 ESP32-SOLO_AT_Bin_V2.1.0.0.zip (Recommended)
- v2.0.0.0 ESP32-SOLO_AT_Bin_V2.0.zip

2.2 Released Firmware

It is recommended to use the lastest version of firmware.

2.2.1 ESP32-S2-WROOM Series

- v2.1.0.0 ESP32-S2-WROOM_AT_Bin_V2.1.0.0.zip (Recommended)

2.2.2 ESP32-S2-WROVER Series

- v2.1.0.0 ESP32-S2-WROVER_AT_Bin_V2.1.0.0.zip (Recommended)

2.2.3 ESP32-S2-SOLO Series

- v2.1.0.0 ESP32-S2-SOLO_AT_Bin_V2.1.0.0.zip (Recommended)

2.2.4 ESP32-S2-MINI Series

- v2.1.0.0 ESP32-S2-MINI_AT_Bin_V2.1.0.0.zip (Recommended)

2.3 Released Firmware

It is recommended to use the lastest version of firmware.

2.3.1 ESP-WROOM-02 Series

- v2.1.0.0 ESP8266-IDF-AT_V2.1.0.0.zip (Recommended)
- v2.0.0.0 ESP8266-IDF-AT_V2.0_0.zip

Each of the linked above ESP-AT-Bin files contains several binaries for some specific functions, and the factory/factory/xxx.bin is the combination of all binaries. So user can only download the `factory/factory_xxx.bin` to address 0, or several binaries to different addresses according to `ESP-AT-Bin/download.config`.

- `at_customize.bin` is to provide a user partition table, which lists different partitions for the `ble_data.bin`, SSL certificates, and `factory_param_XXX.bin`. Furthermore, users can add their own users partitions, and read/write the user partitions with the command `AT+FS` and `AT+SYSFLASH`.
- `factory_param_XXX.bin` indicates the hardware configurations for different ESP modules (see the table below). Please make sure the correct bin is used for your specific module. If users design their own module, they can configure it with reference to the `esp-at/docs/en/Compile_and_Develop/How_to_create_factory_parameter_bin.md`, and the binaries will be automatically generated after compilation. When users flash the firmware into module according to the `download.config`, the `customized_partitions/factory_param.bin` should be replaced with the actual module-specific `customized_partitions/factory_param_XXX.bin`. UART CTS and RTS pins are optional.
  - **ESP32 Series**

Modules	UART Pins (TX, RX, CTS, RTS)	Factory Parameter Bin
ESP32-WROOM-32 Series (ESP32 Default Value)	GPIO17, GPIO16, GPIO15, GPIO14	`factory_param_WROOM-32.bin`
ESP32-WROVER Series (Supports Classic Bluetooth)	GPIO22, GPIO19, GPIO15, GPIO14	`factory_param_WROVER-32.bin`
ESP32-PICO Series	GPIO22, GPIO19, GPIO15, GPIO14	`factory_param_PICO-D4.bin`
ESP32-SOLO Series	GPIO17, GPIO16, GPIO15, GPIO14	`factory_param_SOLO-1.bin`

– **ESP32S2 Series**

Modules	UART Pins (TX, RX, CTS, RTS)	Factory Parameter Bin
ESP32S2-WROOM Series	GPIO17, GPIO21, GPIO20, GPIO19	`factory_param_WROOM.bin`
ESP32S2-WROVER Series	GPIO17, GPIO21, GPIO20, GPIO19	`factory_param_WROVER.bin`
ESP32S2-SOLO Series	GPIO17, GPIO21, GPIO20, GPIO19	`factory_param_SOLO.bin`
ESP32S2-MINI Series	GPIO17, GPIO21, GPIO20, GPIO19	`factory_param_MINI.bin`

– **ESP32-C3 Series**

Modules	UART Pins (TX, RX, CTS, RTS)	Factory Parameter Bin
ESP32-C3-MINI Series	GPIO7, GPIO6, GPIO5, GPIO4	`factory_param_MINI-1.bin`

– **ESP8266 Series**

Modules	UART Pins (TX, RX, CTS, RTS)	Factory Parameter Bin
ESP-WROOM-02 Series (ESP8266 Default Value)	GPIO15, GPIO13, GPIO3, GPIO1	`factory_param_WROOM-02.bin`

- `ble_data.bin` is to provide BLE services when the ESP32 works as a BLE server;
- `server_cert.bin`, `server_key.bin` and `server_ca.bin` are examples of SSL server's certificate;
- `client_cert.bin`, `client_key.bin` and `client_ca.bin` are examples of SSL client's certificate.

If some of the functions are not used, then the corresponding binaries need not to be downloaded into flash.

CHAPTER 3

AT Command Set

[]

Here is a list of AT commands. Some of them can only work on the ESP32 series, so they are marked as [ESP32 Only] at the beginning; those without any mark can work on all ESP series, including ESP32, ESP8266, ESP32-S2, and ESP32-C3.

3.1 Basic AT Commands

[]

- *AT*: Test AT startup.
- *AT+RST*: Restart a module.
- *AT+GMR*: Check version information.
- *AT+CMD*: List all AT commands and types supported in current firmware.
- *AT+GSLP*: Enter Deep-sleep mode.
- *ATE*: Configure AT commands echoing.
- *AT+RESTORE*: Restore factory default settings of the module.
- *AT+UART_CUR*: Current UART configuration, not saved in flash.
- *AT+UART_DEF*: Default UART configuration, saved in flash.
- *AT+SLEEP*: Set the sleep mode.
- *AT+SYSRAM*: Query current remaining heap size and minimum heap size.
- *AT+SYSMSG*: Query/Set System Prompt Information.
- *AT+USERRAM*: Operate user's free RAM.
- *AT+SYSFLASH*: Query/Set User Partitions in Flash.
- [ESP32 Only] *AT+FS*: Filesystem Operations.

- *AT+RFPOWER*: Query/Set RF TX Power.
- *AT+SYSROLLBACK*: Roll back to the previous firmware.
- *AT+SYSTIMESTAMP*: Query/Set local time stamp.
- *AT+SYSLOG*: Enable or disable the AT error code prompt.
- *AT+SLEEPWKCFG*: Query/Set the light-sleep wakeup source and awake GPIO.
- *AT+SYSSTORE*: Query/Set parameter store mode.
- *AT+SYSREG*: Read/write the register.
- [ESP32-S2 Only] *AT+SYSTEMP*: Read ESP32-S2 internal temperature.

3.1.1 AT: Test AT Startup

Execute Command

Command:

```
AT
```

Response:

```
OK
```

3.1.2 AT+RST: Restart a Module

Execute Command

Command:

```
AT+RST
```

Response:

```
OK
```

3.1.3 AT+GMR: Check Version Information

Execute Command

Command:

```
AT+GMR
```

Response:

```
<AT version info>
<SDK version info>
<compile time>
<Bin version>
```

(continues on next page)

(continued from previous page)

```
OK
```

Parameters

- **<AT version info>**: information about the esp-at core library version, which is under the directory: `esp-at/components/at/lib/`. Code is closed source, no plan to open.
- **<SDK version info>**: information about the esp-at platform sdk version, which is defined in file: `esp-at/module_config/module_{platform}_default/IDF_VERSION`
- **<compile time>**: the time to compile the firmware.
- **<Bin version>**: esp-at firmware version. Version information can be modified in menuconfig.

Note

- If you have any issues on esp-at firmware, please provide `AT+GMR` version information firstly.

Example

```
AT+GMR
AT version:2.2.0.0-dev(ca41ec4 - ESP32 - Sep 16 2020 11:28:17)
SDK version:v4.0.1-193-ge7ac221b4
compile time(98b95fc):Oct 29 2020 11:23:25
Bin version:2.1.0(MINI-1)

OK
```

3.1.4 AT+CMD: List all AT commands and types supported in current firmware

Query Command

Command:

```
AT+CMD?
```

Response:

```
+CMD:<index>,<AT command name>,<support test command>,<support query command>,
↪<support set command>,<support execute command>

OK
```

Parameters

- **<index>**: AT command sequence number.
- **<AT command name>**: AT command name.
- **<support test command>**: 0 means not supported, 1 means supported.

- **<support query command>**: 0 means not supported, 1 means supported.
- **<support set command>**: 0 means not supported, 1 means supported.
- **<support execute command>**: 0 means not supported, 1 means supported.

3.1.5 AT+GSLP: Enter Deep-sleep Mode

Set Command

Command:

```
AT+GSLP=<time>
```

Response:

```
<time>

OK
```

Parameter

- **<time>**: the duration when the device stays in Deep-sleep. Unit: millisecond. When the time is up, the device automatically wakes up, calls Deep-sleep wake stub, and then proceeds to load the application.
  - For ESP32 devices:
    - 0 means restarting right now
    - the maximum Deep-sleep time is about 28.8 days (2^{31}-1 milliseconds)
  - For ESP32-S2 devices:
    - 0 means staying in Deep-sleep mode forever
    - the maximum Deep-sleep time is about 28.8 days (2^{31}-1 milliseconds)
  - For ESP8266 devices:
    - 0 means staying in Deep-sleep mode forever
    - the maximum Deep-sleep time is about 3 hours (due to hardware limitation, more time will lead to setting failure or internal time overflow)

Notes

- For ESP8266 devices, you must connect GPIO16 to RST pin to wake them up automatically when time is up.
- For all devices, affected by external factors, the theoretical and actual time of Deep-sleep will be different.
- ESP8266 devices can be waken up from Deep-sleep by directly triggering the RST pin low-level pulse.

3.1.6 ATE: Configure AT Commands Echoing

Execute Command

Command:

```
ATE0
```

or

```
ATE1
```

Response:

```
OK
```

Parameters

- **ATE0**: Switch echo off.
- **ATE1**: Switch echo on.

3.1.7 AT+RESTORE: Restore Factory Default Settings

Execute Command

Command:

```
AT+RESTORE
```

Response:

```
OK
```

Notes

- The execution of this command will restore all parameters saved in flash to factory default settings of the module.
- The device will be restarted when this command is executed.

3.1.8 AT+UART_CUR: Current UART Configuration, Not Saved in Flash

Query Command

Command:

```
AT+UART_CUR?
```

Response:

```
+UART_CUR:<baudrate>,<databits>,<stopbits>,<parity>,<flow control>

OK
```

Set Command

Command:

```
AT+UART_CUR=<baudrate>,<databits>,<stopbits>,<parity>,<flow control>
```

Response:

```
OK
```

Parameters

- **<baudrate>**: UART baud rate
  - For ESP32 and ESP32-S2 devices, the supported range is 80 ~ 5000000.
  - For ESP8266 devices, the supported rang is 80 ~ 4500000.
- **<databits>**: data bits
  - 5: 5-bit data
  - 6: 6-bit data
  - 7: 7-bit data
  - 8: 8-bit data
- **<stopbits>**: stop bits
  - 1: 1-bit stop bit
  - 2: 1.5-bit stop bit
  - 3: 2-bit stop bit
- **<parity>**: parity bit
  - 0: None
  - 1: Odd
  - 2: Even
- **<flow control>**: flow control
  - 0: flow control is not enabled
  - 1: enable RTS
  - 2: enable CTS
  - 3: enable both RTS and CTS

Notes

- The Query Command will return actual values of UART configuration parameters, which may have minor differences from the set value because of the clock division.
- The configuration changes will NOT be saved in flash.
- To use hardware flow control, you need to connect CTS/RTS pins of your ESP device. For more details, please refer to *Hardware Connection* or `components/customized_partitions/raw_data/factory_param/factory_param_data.csv`.

Example

```
AT+UART_CUR=115200,8,1,0,3
```

3.1.9 AT+UART_DEF: Default UART Configuration, Saved in Flash

Query Command

Command:

```
AT+UART_DEF?
```

Response:

```
+UART_DEF:<baudrate>,<databits>,<stopbits>,<parity>,<flow control>

OK
```

Set Command

Command:

```
AT+UART_DEF=<baudrate>,<databits>,<stopbits>,<parity>,<flow control>
```

Response:

```
OK
```

Parameters

- **<baudrate>**: UART baud rate
  - For ESP32 and ESP32-S2 devices, the supported range is 80 ~ 5000000.
  - For ESP8266 devices, the supported rang is 80 ~ 4500000.
- **<databits>**: data bits
  - 5: 5-bit data
  - 6: 6-bit data
  - 7: 7-bit data
  - 8: 8-bit data
- **<stopbits>**: stop bits
  - 1: 1-bit stop bit
  - 2: 1.5-bit stop bit
  - 3: 2-bit stop bit
- **<parity>**: parity bit
  - 0: None

- 1: Odd
- 2: Even

- **<flow control>**: flow control
  - 0: flow control is not enabled
  - 1: enable RTS
  - 2: enable CTS
  - 3: enable both RTS and CTS

Notes

- The configuration changes will be saved in the NVS area, and will still be valid when the chip is powered on again.
- To use hardware flow control, you need to connect CTS/RTS pins of your ESP device. For more details, please refer to *Hardware Connection* or `components/customized_partitions/raw_data/factory_param/factory_param_data.csv`.

Example

```
AT+UART_DEF=115200,8,1,0,3
```

3.1.10 AT+SLEEP: Set the Sleep Mode

Set Command

Command:

```
AT+SLEEP=<sleep mode>
```

Response:

```
OK
```

Parameter

- **<sleep mode>**:
  - 0: Disable the sleep mode.
  - 1: Modem-sleep DTIM mode. RF will be periodically closed according to AP DTIM.
  - 2: Light-sleep mode. CPU will automatically sleep and RF will be periodically closed according to `listen interval` set by *AT+CWJAP*.
  - 3: Modem-sleep listen interval mode. RF will be periodically closed according to `listen interval` set by *AT+CWJAP*.

Note

- Modem-sleep mode and Light-sleep mode can be set only in station mode.
- Before setting the Light-sleep mode, it is recommended to set the wakeup source in advance through the command *AT+SLEEPWKCFG*, otherwise ESP devices can't wake up and will always be in sleep mode.
- After setting the Light-sleep mode, if the Light-sleep wakeup condition is not met, ESP devices will automatically enter the sleep mode. When the Light-sleep wakeup condition is met, ESP devices will automatically wake up from sleep mode.

Example

```
AT+SLEEP=0
```

3.1.11 AT+SYSRAM: Query Current Remaining Heap Size and Minimum Heap Size

Query Command

Command:

```
AT+SYSRAM?
```

Response:

```
+SYSRAM:<remaining RAM size>,<minimum heap size>
OK
```

Parameters

- **<remaining RAM size>**: current remaining heap size. Unit: byte.
- **<minimum heap size>**: minimum heap size that has ever been available. Unit: byte.

Example

```
AT+SYSRAM?
+SYSRAM:148408,84044
OK
```

3.1.12 AT+SYSMSG: Query/Set System Prompt Information

Query Command

Function:

Query the current system prompt information state.

Command:

```
AT+SYSMSG?
```

Response:

```
+SYSMSG:<state>
OK
```

Set Command

Function:

Configure system prompt information.

Command:

```
AT+SYSMSG=<state>
```

Response:

```
OK
```

Parameter

- **<state>**:
  - Bit0: Prompt information when quitting Wi-Fi *Passthrough Mode*.
    - 0: Print no prompt information when quitting Wi-Fi *Passthrough Mode*.
    - 1: Print `+QUITT` when quitting Wi-Fi *Passthrough Mode*.
  - Bit1: Connection prompt information type.
    - 0: Use simple prompt information, such as `XX,CONNECT`.
    - 1: Use detailed prompt information, such as `+LINK_CONN:status_type,link_id,ip_type,terminal_type,remote_ip,remote_port,local_port`.
  - Bit2: Connection status prompt information for Wi-Fi *Passthrough Mode*, Bluetooth LE SPP and Bluetooth SPP.
    - 0: Print no prompt information.
    - 1: Print one of the following prompt information when Wi-Fi, socket, Bluetooth LE or Bluetooth status is changed:

```
- "CONNECT\r\n" or the message prefixed with "+LINK_CONN:"
- "CLOSED\r\n"
- "WIFI CONNECTED\r\n"
- "WIFI GOT IP\r\n"
- "WIFI GOT IPv6 LL\r\n"
- "WIFI GOT IPv6 GL\r\n"
- "WIFI DISCONNECT\r\n"
- "+ETH_CONNECTED\r\n"
- "+ETH_DISCONNECTED\r\n"
- the message prefixed with "+ETH_GOT_IP:"
- the message prefixed with "+STA_CONNECTED:"
- the message prefixed with "+STA_DISCONNECTED:"
```

(continues on next page)

(continued from previous page)

```
- the message prefixed with "+DIST_STA_IP:"
- the message prefixed with "+BLECONN:"
- the message prefixed with "+BLEDISCONN:"
```

Notes

- The configuration changes will be saved in the NVS area if `AT+SYSSTORE=1`.
- If you set Bit0 to 1, it will prompt "+QUITT" when you quit Wi-Fi *Passthrough Mode*.
- If you set Bit1 to 1, it will impact the information of command *AT+CIPSTART* and *AT+CIPSERVER*. It will supply "+LINK_CONN:status_type,link_id,ip_type,terminal_type,remote_ip,remote_port,local_port" instead of "XX,CONNECT".

Example

```
// print no promt info when quitting Wi-Fi passthrough mode
// print detailed connection prompt info
// print no prompt info when the connection status is changed
AT+SYSMSG=2
```

3.1.13 AT+USERRAM: Operate user's free RAM

Query Command

Function:

Query the current available user's RAM size.

Command:

```
AT+USERRAM?
```

Response:

```
+USERRAM:<size>
OK
```

Set Command

Function:

Operate user's free RAM

Command:

```
AT+USERRAM=<operation>,<size>[,<offset>]
```

Response:

:: +USERRAM:<length>,<data> // esp-at returns this response only when the operator is `read`

OK

Parameters

- **<operation>**:
  - 0: release user's RAM
  - 1: malloc user's RAM
  - 2: write user's RAM
  - 3: read user's RAM
  - 4: clear user's RAM
- **<size>**: the size to malloc/read/write
- **<offset>**: the offset to read/write. Default: 0

Notes

- Please malloc the RAM size before you perform any other operations.
- If the operator is `write`, wrap return > after the write command, then you can send the data that you want to write. The length should be parameter `<length>`.
- If the operator is `read` and the length is bigger than 1024, ESP-AT will reply multiple times in the same format, each reply can carry up to 1024 bytes of data, and eventually end up with `\r\nOK\r\n`.

Example

```
// malloc 1 KB user's RAM
AT+USERRAM=1,1024

// write 500 bytes to RAM (offset: 0)
AT+USERRAM=2,500

// read 64 bytes from RAM offset 100
AT+USERRAM=3,64,100

// free the user's RAM
AT+USERRAM=0
```

3.1.14 AT+SYSFLASH: Query/Set User Partitions in Flash

Query Command

Function:

Query user partitions in flash.

Command:

```
AT+SYSFLASH?
```

Response:

```
+SYSFLASH:<partition>,<type>,<subtype>,<addr>,<size>
OK
```

Set Command

Function:

Read/write the user partitions in flash.

Command:

```
AT+SYSFLASH=<operation>,<partition>,<offset>,<length>
```

Response:

```
+SYSFLASH:<length>,<data>
OK
```

Parameters

- **<operation>**:
  - 0: erase sector
  - 1: write data into the user partition
  - 2: read data from the user partition
- **<partition>**: name of user partition
- **<offset>**: offset of user partition
- **<length>**: data length
- **<type>**: type of user partition
- **<subtype>**: subtype of user partition
- **<addr>**: address of user partition
- **<size>**: size of user partition

Notes

- Please make sure that you have downloaded at_customize.bin before using this command. For more details, please refer to *The Secondary Partitions Table*.
- When erasing the targeted user partition in its entirety, you can omit the parameters `<offset>` and `<length>`. For example, command `AT+SYSFLASH=0,"ble_data"` can erase the entire "ble_data" user partition. But if you want to keep the two parameters, they have to be 4KB-aligned.
- The introduction to partitions is in ESP-IDF Partition Tables.
- If the operator is `write`, wrap return `>` after the write command, then you can send the data that you want to write. The length should be parameter `<length>`.
- If the operator is `write`, please make sure that you have already erased this partition.
- If the operator is `write` on a PKI bin, the `<length>` should be 4 bytes aligned.

Example

```
// read 100 bytes from the "ble_data" partition offset 0.
AT+SYSFLASH=2,"ble_data",0,100

// write 10 bytes to the "ble_data" partition offset 100.
AT+SYSFLASH=1,"ble_data",100,10

// erase 8192 bytes from the "ble_data" partition offset 4096.
AT+SYSFLASH=0,"ble_data",4096,8192
```

3.1.15 [ESP32 Only] AT+FS: Filesystem Operations

Set Command

Command:

```
AT+FS=<type>,<operation>,<filename>,<offset>,<length>
```

Response:

```
OK
```

Parameters

- **<type>**: only FATFS is currently supported.
  - 0: FATFS
- **<operation>**:
  - 0: delete file.
  - 1: write file.
  - 2: read file.
  - 3: query the size of the file.
  - 4: list files in a specific directory. Only root directory is currently supported.
- **<offset>**: apply to writing and reading operations only.
- **<length>**: data length, applying to writing and reading operations only.

Notes

- Please make sure that you have downloaded at_customize.bin before using this command. For more details, refer to ESP-IDF Partition Tables and *The Secondary Partitions Table*.
- If the length of the read data is greater than the actual file length, only the actual data length of the file will be returned.
- If the operator is `write`, wrap return `>` after the write command, then you can send the data that you want to write. The length should be parameter `<length>`.

Example

```
// delete a file.
AT+FS=0,0,"filename"

// write 10 bytes to offset 100 of a file.
AT+FS=0,1,"filename",100,10

// read 100 bytes from offset 0 of a file.
AT+FS=0,2,"filename",0,100

// list all files in the root directory.
AT+FS=0,4,"."
```

3.1.16 AT+RFPOWER: Query/Set RF TX Power

Query Command

Function:

Query the RF TX Power.

Command:

```
AT+RFPOWER?
```

Response:

```
+RFPOWER:<wifi_power>,<ble_adv_power>,<ble_scan_power>,<ble_conn_power>
OK
```

Set Command

Command:

```
AT+RFPOWER=<wifi_power>[,<ble_adv_power>,<ble_scan_power>,<ble_conn_power>]
```

Response:

```
OK
```

Parameters

- **<wifi_power>**: the unit is 0.25 dBm. For example, if you set the value to 78, the actual maximum RF Power value is 78 * 0.25 dBm = 19.5 dBm. After you configure it, please confirm the actual value by entering the command `AT+RFPOWER?`.
  - For ESP32 and ESP32-S2 devices, the range is [40,78]:

set value	actual value	actual dBm
[34,43]	34	8.5
[44,51]	44	11
[52,55]	52	13
[56,59]	56	14
[60,65]	60	15
[66,71]	66	16.5
[72,77]	72	18
78	78	19.5

– For ESP32-C3 devices, the range is [40,84]:

set value	actual value	actual dBm
[40,80]	<set value>	<set value> * 0.25
[81,84]	80	20

– For ESP8266 devices, the range is [40,82]:

set value	actual value	actual dBm
[40,47]	32	8
[48,55]	48	12
[56,63]	56	14
[64,67]	64	16
[68,73]	68	17
[74,77]	74	18.5
[78,81]	78	19.5
82	82	20.5

- **<ble_adv_power>**: RF TX Power of Bluetooth LE advertising. Range: [0,7].
  - 0: 7 dBm
  - 1: 4 dBm
  - 2: 1 dBm
  - 3: -2 dBm
  - 4: -5 dBm
  - 5: -8 dBm
  - 6: -11 dBm
  - 7: -14 dBm
- **<ble_scan_power>**: RF TX Power of Bluetooth LE scanning. Range: [0,7]: the parameters are the same as `<ble_adv_power>`.
- **<ble_conn_power>**: RF TX Power of Bluetooth LE connecting. Range: [0,7]: the same as `<ble_adv_power>`.

3.1.17 Note

- Since the RF TX Power is actually divided into several levels, and each level has its own value range, the `wifi_power` value queried by the `esp_wifi_get_max_tx_power` may differ from the value set by `esp_wifi_set_max_tx_power` and is no larger than the set value.

3.1.18 AT+SYSROLLBACK: Roll Back to the Previous Firmware

Execute Command

Command:

```
AT+SYSROLLBACK
```

Response:

```
OK
```

Note

- This command will not upgrade via OTA. It only rolls back to the firmware which is in the other OTA partition.

3.1.19 AT+SYSTIMESTAMP: Query/Set Local Time Stamp

Query Command

Function:

Query the time stamp.

Command:

```
AT+SYSTIMESTAMP?
```

Response:

```
+SYSTIMESTAMP:<Unix_timestamp>
OK
```

Set Command

Function:

Set local time stamp. It will be the same as SNTP time when the SNTP time is updated.

Command:

```
AT+SYSTIMESTAMP=<Unix_timestamp>
```

Response:

```
OK
```

Parameter

- **<Unix-timestamp>**: Unix timestamp. Unit: second.

Example

```
AT+SYSTIMESTAMP=1565853509    //2019-08-15 15:18:29
```

3.1.20 AT+SYSLOG: Enable or Disable the AT Error Code Prompt

Query Command

Function:

Query whether the AT error code prompt is enabled or not.

Command:

```
AT+SYSLOG?
```

Response:

```
+SYSLOG:<status>

OK
```

Set Command

Function:

Enable or disable the AT error code prompt.

Command:

```
AT+SYSLOG=<status>
```

Response:

```
OK
```

Parameter

- **<status>**: enable or disable
  - 0: disable
  - 1: enable

Example

```
// enable AT error code prompt
AT+SYSLOG=1

OK
AT+FAKE
ERR CODE:0x01090000

ERROR
```

```
// disable AT error code prompt
AT+SYSLOG=0

OK
AT+FAKE
// No `ERR CODE:0x01090000`

ERROR
```

The error code is a 32-bit hexadecimal value and defined as follows:

category	subcategory	extension
bit32 ~ bit24	bit23 ~ bit16	bit15 ~ bit0

- **category:** stationary value 0x01.
- **subcategory:** error type.

Table 1: Subcategory of Error Code

Error Type	Error Code	Description
ESP_AT_SUB_OK	0x00	OK
ESP_AT_SUB_COMMON_ERROR	0x01	reserved
ESP_AT_SUB_NO_TERMINATOR	0x02	terminator character not found ("rn" expected)
ESP_AT_SUB_NO_AT	0x03	Starting AT not found (or at, At or aT entered)
ESP_AT_SUB_PARA_LENGTH_MISMATCH	0x04	parameter length mismatch
ESP_AT_SUB_PARA_TYPE_MISMATCH	0x05	parameter type mismatch
ESP_AT_SUB_PARA_NUM_MISMATCH	0x06	parameter number mismatch
ESP_AT_SUB_PARA_INVALID	0x07	the parameter is invalid
ESP_AT_SUB_PARA_PARSE_FAIL	0x08	parse parameter fail
ESP_AT_SUB_UNSUPPORT_CMD	0x09	the command is not supported
ESP_AT_SUB_CMD_EXEC_FAIL	0x0A	the command execution failed
ESP_AT_SUB_CMD_PROCESSING	0x0B	processing of previous command is in progress
ESP_AT_SUB_CMD_OP_ERROR	0x0C	the command operation type is error

- **extension:** error extension information. There are different extensions for different subcategory. For more information, please see the `components/at/include/esp_at.h`.

For example, the error code `ERR CODE:0x01090000` means the command is not supported.

3.1.21 AT+SLEEPWKCFG: Set the Light-sleep Wakeup Source and Awake GPIO

Set Command

Command:

```
AT+SLEEPWKCFG=<wakeup source>,<param1>[,<param2>]
```

Response:

```
OK
```

Parameters

- **<wakeup source>**:
  - 0: wakeup by a timer.
  - 1: reserved.
  - 2: wakeup by GPIO.
- **<param1>**:
  - If the wakeup source is a timer, it means the time before wakeup. Unit: millisecond.
  - If the wakeup source is GPIO, it means the GPIO number.
- **<param2>**:
  - If the wakeup source is GPIO, it means the wakeup level:
  - 0: low level.
  - 1: high level.

Note

- GPIO16 as the RTC IO can not be set as GPIO wakeup source on ESP8266 platform for light sleep.

Example

```
// Timer wakeup
AT+SLEEPWKCFG=0,1000

// GPIO12 wakeup, low level
AT+SLEEPWKCFG=2,12,0
```

3.1.22 AT+SYSSTORE: Query/Set Parameter Store Mode

Query Command

Function:

Query the AT parameter store mode.

Command:

```
AT+SYSSTORE?
```

Response:

```
+SYSSTORE:<store_mode>

OK
```

Set Command

Command:

```
AT+SYSSTORE=<store_mode>
```

Response:

```
OK
```

Parameter

- **<store_mode>**:
  - 0: command configuration is not stored into flash.
  - 1: command configuration is stored into flash. (Default)

Note

- This command affects set commands only. Query commands are always fetched from RAM.
- Affected commands:
  - *AT+SYSMSG*
  - *AT+CWMODE*
  - *AT+CIPV6*
  - *AT+CWJAP*
  - *AT+CWSAP*
  - *AT+CWRECONNCFG*
  - *AT+CIPAP*
  - *AT+CIPSTA*
  - *AT+CIPAPMAC*
  - *AT+CIPSTAMAC*
  - *AT+CIPDNS*
  - *AT+CIPSSLCCONF*
  - *AT+CIPRECONNINTV*
  - *AT+CIPTCPOPT*
  - *AT+CWDHCPS*
  - *AT+CWDHCP*
  - *AT+CWSTAPROTO*
  - *AT+CWAPPROTO*
  - *AT+CWJEAP*
  - *AT+CIPETH*
  - *AT+CIPETHMAC*

- *AT+BLENAME*
- *AT+BTNAME*
- *AT+BLEADVPARAM*
- *AT+BLEADVDATA*
- *AT+BLEADVDATAEX*
- *AT+BLESCANRSPDATA*
- *AT+BLESCANPARAM*
- *AT+BTSCANMODE*
- *AT+BLECONNPARAM*

Examples

```
AT+SYSSTORE=0
AT+CWMODE=1  // Not stored into flash
AT+CWJAP="test","1234567890" // Not stored into flash

AT+SYSSTORE=1
AT+CWMODE=3  // Stored into flash
AT+CWJAP="test","1234567890" // Stored into flash
```

3.1.23 AT+SYSREG: Read/Write the Register

Set Command

Command:

```
AT+SYSREG=<direct>,<address>[,<write value>]
```

Response:

```
+SYSREG:<read value>    // Only in read mode
OK
```

Parameters

- **<direct>**: read or write register.
  - 0: read register.
  - 1: write register.
- **<address>**: (uint32) register address. You can refer to Technical Reference Manuals.
- **<write value>**: (uint32) write value (only in write mode).

Note

- AT does not check address. Make sure that the registers you are operating on are valid.

Example

```
// Enable ESP32-S2 IO33 output, 0x3F40402C means base address 0x3F404000 add relative␣
↪address 0x2C (GPIO_ENABLE1_REG)
AT+SYSREG=1,0x3F40402C,0x2

// ESP32-S2 IO33 output high
AT+SYSREG=1,0x3F404010,0x2

// ESP32-S2 IO33 output low
AT+SYSREG=1,0x3F404010,0x0
```

3.1.24 [ESP32-S2 Only] AT+SYSTEMP: Read ESP32-S2 Internal Temperature

Query Command

Command:

```
AT+SYSTEMP?
```

Response:

```
+SYSTEMP:<temperature>
OK
```

Parameter

- **<temperature>**: the measured output value. Unit: Celsius.

Note

- Measure range: -10°C ~ 80°C. Error < 1°C.

Example

```
AT+SYSTEMP?
+SYSTEMP:21.59
OK
```

3.2 Wi-Fi AT Commands

[]

- *AT+CWMODE*: Set the Wi-Fi mode (Station/SoftAP/Station+SoftAP).
- *AT+CWSTATE*: Query the Wi-Fi state and Wi-Fi information.
- *AT+CWJAP*: Connect to an AP.
- *AT+CWRECONNCFG*: Query/Set the Wi-Fi reconnecting configuration.

- *AT+CWLAPOPT*: Set the configuration for the command *AT+CWLAP*.
- *AT+CWLAP*: List available APs.
- *AT+CWQAP*: Disconnect from an AP.
- *AT+CWSAP*: Query/Set the configuration of an ESP SoftAP.
- *AT+CWLIF*: Obtain IP address of the station that connects to an ESP SoftAP.
- *AT+CWQIF*: Disconnect stations from an ESP SoftAP.
- *AT+CWDHCP*: Enable/disable DHCP.
- *AT+CWDHCPS*: Query/Set the IP addresses allocated by an ESP SoftAP DHCP server.
- *AT+CWAUTOCONN*: Connect to an AP automatically when powered on.
- *AT+CWAPPROTO*: Query/Set the 802.11 b/g/n protocol standard of SoftAP mode.
- *AT+CWSTAPROTO*: Query/Set the 802.11 b/g/n protocol standard of station mode.
- *AT+CIPSTAMAC*: Query/Set the MAC address of an ESP station.
- *AT+CIPAPMAC*: Query/Set the MAC address of an ESP SoftAP.
- *AT+CIPSTA*: Query/Set the IP address of an ESP station.
- *AT+CIPAP*: Query/Set the IP address of an ESP SoftAP.
- *AT+CWSTARTSMART*: Start SmartConfig.
- *AT+CWSTOPSMART*: Stop SmartConfig.
- *AT+WPS*: Enable the WPS function.
- *AT+MDNS*: Configure the mDNS function.
- [ESP32 Only] *AT+CWJEAP*: Connect to a WPA2 Enterprise AP.
- *AT+CWHOSTNAME*: Query/Set the host name of an ESP station.
- *AT+CWCOUNTRY*: Query/Set the Wi-Fi Country Code.

3.2.1 AT+CWMODE: Query/Set the Wi-Fi Mode (Station/SoftAP/Station+SoftAP)

Query Command

Function:

Query the Wi-Fi mode of ESP devices.

Command:

```
AT+CWMODE?
```

Response:

```
+CWMODE:<mode>
OK
```

Set Command

Function:

Set the Wi-Fi mode of ESP devices.

Command:

```
AT+CWMODE=<mode>[,<auto_connect>]
```

Response:

```
OK
```

Parameters

- **<mode>**:
  - 0: Null mode. Wi-Fi RF will be disabled.
  - 1: Station mode.
  - 2: SoftAP mode.
  - 3: SoftAP+Station mode.
- **<auto_connect>**: Enable or disable automatic connection to an AP when you change the mode of the ESP device from the SoftAP mode or null mode to the station mode or the SoftAP+Station mode. Default: 1. If you omit the parameter, the default value will be used, i.e. automatically connecting to an AP.
  - 0: The ESP device will not automatically connect to an AP.
  - 1: The ESP device will automatically connect to an AP if the configuration to connect to the AP has already been saved in flash before.

Note

- The configuration changes will be saved in the NVS area if *AT+SYSSTORE=1*.

Example

```
AT+CWMODE=3
```

3.2.2 AT+CWSTATE: Query the Wi-Fi state and Wi-Fi information

Query Command

Function:

Query the Wi-Fi state and Wi-Fi information of ESP devices.

Command:

```
AT+CWSTATE?
```

Response:

```
+CWSTATE:<state>,<"ssid">

OK
```

Parameters

- **<state>**: current Wi-Fi state.
  - 0: ESP station has not started any Wi-Fi connection.
  - 1: ESP station has connected to an AP, but does not get an IPv4 address yet.
  - 2: ESP station has connected to an AP, and got an IPv4 address.
  - 3: ESP station is in Wi-Fi connecting or reconnecting state.
  - 4: ESP station is in Wi-Fi disconnected state.
- **<"ssid">**: the SSID of the target AP.

Note

- When ESP station is not connected to an AP, it is recommended to use this command to query Wi-Fi information; after ESP station is connected to an AP, it is recommended to use *AT+CWJAP* to query Wi-Fi information.

3.2.3 AT+CWJAP: Connect to an AP

Query Command

Function:

Query the AP to which the ESP Station is already connected.

Command:

```
AT+CWJAP?
```

Response:

```
+CWJAP:<ssid>,<bssid>,<channel>,<rssi>,<pci_en>,<reconn_interval>,<listen_interval>,
↪<scan_mode>,<pmf>
OK
```

Set Command

Function:

Connect an ESP station to a targeted AP.

Command:

```
AT+CWJAP=[<ssid>],[<pwd>][,<bssid>][,<pci_en>][,<reconn_interval>][,<listen_interval>
↪][,<scan_mode>][,<jap_timeout>][,<pmf>]
```

Response:

```
WIFI CONNECTED
WIFI GOT IP

OK
[WIFI GOT IPv6 LL]
[WIFI GOT IPv6 GL]
```

or

```
+CWJAP:<error code>
ERROR
```

Execute Command

Function:

Connect an ESP station to a targeted AP with last Wi-Fi configuration.

Command:

```
AT+CWJAP
```

Response:

```
WIFI CONNECTED
WIFI GOT IP

OK
[WIFI GOT IPv6 LL]
[WIFI GOT IPv6 GL]
```

or

```
+CWJAP:<error code>
ERROR
```

Parameters

- **<ssid>**: the SSID of the target AP.
  - Escape character syntax is needed if SSID or password contains special characters, such `,`, `"`, or `\\`.
- **<pwd>**: password, MAX: 64-byte ASCII.
- **<bssid>**: the MAC address of the target AP. It cannot be omitted when multiple APs have the same SSID.
- **<channel>**: channel.
- **<rssi>**: signal strength.
- **<pci_en>**: PCI Authentication.
  - 0: The ESP station will connect APs with all encryption methods, including OPEN and WEP.
  - 1: The ESP station will connect APs with all encryption methods, except OPEN and WEP.
- **<reconn_interval>**: the interval between Wi-Fi reconnections. Unit: second. Default: 1. Maximum: 7200.
  - 0: The ESP station will not reconnect to the AP when disconnected.

  - [1,7200]: The ESP station will reconnect to the AP at the specified interval when disconnected.
- **<listen_interval>**: the interval of listening to the AP's beacon. Unit: AP beacon intervals. Default: 3. Range: [1,100].
- **<scan_mode>**:
  - 0: fast scan. It will end after finding the targeted AP. The ESP station will connect to the first scanned AP.
  - 1: all-channel scan. It will end after all the channels are scanned. The device will connect to the scanned AP with the strongest signal.
- **<jap_timeout>**: maximum timeout for *AT+CWJAP* command. Unit: second. Default: 15. Range: [3,600].
- **<pmf>**: Protected Management Frames. Default: 0.
  - 0 means disable PMF.
  - bit 0: PMF capable, advertizes support for protected management frame. Device will prefer to connect in PMF mode if other device also advertizes PMF capability.
  - bit 1: PMF required, advertizes that protected management frame is required. Device will not associate to non-PMF capable devices.
- **<error code>**: (for reference only)
  - 1: connection timeout.
  - 2: wrong password.
  - 3: cannot find the target AP.
  - 4: connection failed.
  - others: unknown error occurred.

Notes

- The configuration changes will be saved in the NVS area if *AT+SYSSTORE=1*.
- This command requires Station mode to be enabled.
- The parameter `<reconn_interval>` of this command is the same as `<interval_second>` of the command *AT+CWRECONNCFG*. Therefore, if you omit `<reconn_interval>` when running this command, the interval between Wi-Fi reconnections will use the default value 1.
- If the `<ssid>` and `<password>` parameter are omitted, AT will use the last configuration.
- Execute command has the same maximum timeout to setup command. The default value is 15 seconds, but you can change it by setting the parameter `<jap_timeout>`.
- To get an IPv6 address, you need to set *AT+CIPV6=1*.
- Response `OK` means that the IPv4 network is ready, but not the IPv6 network. At present, ESP-AT is mainly based on IPv4 network, supplemented by IPv6 network.
- `WIFI GOT IPv6 LL` represents that the linklocal IPv6 address has been obtained. This address is calculated locally through EUI-64 and does not require the participation of the AP. Because of the parallel timing, this print may be before or after `OK`.
- `WIFI GOT IPv6 GL` represents that the global IPv6 address has been obtained. This address is combined by the prefix issued by AP and the suffix calculated internally, which requires the participation of the AP. Because of the parallel timing, this print may be before or after `OK`, or it may not be printed because the AP does not support IPv6.

Example

```
// If the target AP's SSID is "abc" and the password is "0123456789", the command␣
↪should be:
AT+CWJAP="abc","0123456789"

// If the target AP's SSID is "ab\,c" and the password is "0123456789"\", the command␣
↪should be:
AT+CWJAP="ab\\\,c","0123456789\"\\"

// If multiple APs all have the SSID of "abc", the target AP can be found by BSSID:
AT+CWJAP="abc","0123456789","ca:d7:19:d8:a6:44"

// If esp-at is required that connect to a AP by protected management frame, the␣
↪command should be:
AT+CWJAP="abc","0123456789",,,,,,,3
```

3.2.4 AT+CWRECONNCFG: Query/Set the Wi-Fi Reconnecting Configuration

Query Command

Function:

Query the configuration of Wi-Fi reconnect.

Command:

```
AT+CWRECONNCFG?
```

Response:

```
+CWRECONNCFG:<interval_second>,<repeat_count>
OK
```

Set Command

Function:

Set the configuration of Wi-Fi reconnect.

Command:

```
AT+CWRECONNCFG=<interval_second>,<repeat_count>
```

Response:

```
OK
```

Parameters

- **<interval_second>**: the interval between Wi-Fi reconnections. Unit: second. Default: 0. Maximum: 7200.
  - 0: The ESP station will not reconnect to the AP when disconnected.
  - [1,7200]: The ESP station will reconnect to the AP at the specified interval when disconnected.

- **<repeat_count>**: the number of attempts the ESP device makes to reconnect to the AP. This parameter only works when the parameter `<interval_second>` is not 0. Default: 0. Maximum: 1000.
  - 0: The ESP station will always try to reconnect to AP.
  - [1,1000]: The ESP station will attempt to reconnect to AP for the specified times.

Example

```
// The ESP station tries to reconnect to AP at the interval of one second for 100␣
↪times.
AT+CWRECONNCFG=1,100

// The ESP station will not reconnect to AP when disconnected.
AT+CWRECONNCFG=0,0
```

Notes

- The parameter `<interval_second>` of this command is the same as the parameter `[<reconn_interval>]` of the command *AT+CWJAP*.
- This command works for passive disconnection from APs, Wi-Fi mode switch, and Wi-Fi auto connect after power on.

3.2.5 AT+CWLAPOPT: Set the Configuration for the Command AT+CWLAP

Set Command

Command:

```
AT+CWLAPOPT=<reserved>,<print mask>[,<rssi filter>][,<authmode mask>]
```

Response:

```
OK
```

or

```
ERROR
```

Parameters

- **<reserved>**: reserved item.
- **<print mask>**: determine whether the following parameters are shown in the result of *AT+CWLAP*. Default: 0x7FF. If you set them to 1, it means showing the corresponding parameters; if you set them as 0, it means NOT showing the corresponding parameters.
  - bit 0: determine whether <ecn> will be shown.
  - bit 1: determine whether <ssid> will be shown.
  - bit 2: determine whether <rssi> will be shown.
  - bit 3: determine whether <mac> will be shown.

  - bit 4: determine whether <channel> will be shown.
  - bit 5: determine whether <freq_offset> will be shown.
  - bit 6: determine whether <freqcal_val> will be shown.
  - bit 7: determine whether <pairwise_cipher> will be shown.
  - bit 8: determine whether <group_cipher> will be shown.
  - bit 9: determine whether <bgn> will be shown.
  - bit 10: determine whether <wps> will be shown.
- **[<rssi filter>]**: determine whether the result of the command *AT+CWLAP* will be filtered according to `rssi filter`. In other words, the result of the command will **NOT** show the APs whose signal strength is below `rssi filter`. Unit: dBm. Default: –100. Range: [–100,40].
- **[<authmode mask>]**: determine whether APs with the following authmodes are shown in the result of *AT+CWLAP*. Default: 0xFFFF. If you set `bit x` to 1, the APs with the corresponding authmode will be shown. If you set `bit x` to 0, the APs with the corresponding authmode will NOT be shown;
  - bit 0: determine whether APs with `OPEN` authmode will be shown.
  - bit 1: determine whether APs with `WEP` authmode will be shown.
  - bit 2: determine whether APs with `WPA_PSK` authmode will be shown.
  - bit 3: determine whether APs with `WPA2_PSK` authmode will be shown.
  - bit 4: determine whether APs with `WPA_WPA2_PSK` authmode will be shown.
  - bit 5: determine whether APs with `WPA2_ENTERPRISE` authmode will be shown.
  - bit 6: determine whether APs with `WPA3_PSK` authmode will be shown.
  - bit 7: determine whether AP with `WPA2_WPA3_PSK` authmode will be shown.
  - [ESP32-C3 Only] bit 8: determine whether AP with `WAPI_PSK` authmode will be shown.

Example

```
// The first parameter is 1, meaning that the result of the command AT+CWLAP will be␣
↪ordered according to RSSI;
// The second parameter is 31, namely 0x1F, meaning that the corresponding bits of
↪<print mask> are set to 1. All parameters will be shown in the result of AT+CWLAP.
AT+CWLAPOPT=1,31
AT+CWLAP

// Just show the AP which authmode is OPEN
AT+CWLAPOPT=1,31,-100,1
AT+CWLAP
```

3.2.6 AT+CWLAP: List Available APs

Set Command

Function:

Query the APs with specified parameters, such as the SSID, MAC address, or channel.

Command:

```
AT+CWLAP=[<ssid>,<mac>,<channel>,<scan_type>,<scan_time_min>,<scan_time_max>]
```

Execute Command

Function:

List all available APs.

Command:

```
AT+CWLAP
```

Response:

```
+CWLAP:<ecn>,<ssid>,<rssi>,<mac>,<channel>,<freq_offset>,<freqcal_val>,<pairwise_
↪cipher>,<group_cipher>,<bgn>,<wps>
OK
```

Parameters

- **<ecn>**: encryption method.
  - 0: OPEN
  - 1: WEP
  - 2: WPA_PSK
  - 3: WPA2_PSK
  - 4: WPA_WPA2_PSK
  - 5: WPA2_ENTERPRISE
  - 6: WPA3_PSK
  - 7: WPA2_WPA3_PSK
  - [ESP32-C3 Only] 8: WAPI_PSK
- **<ssid>**: string parameter showing SSID of the AP.
- **<rssi>**: signal strength.
- **<mac>**: string parameter showing MAC address of the AP.
- **<channel>**: channel.
- **<scan_type>**: Wi-Fi scan type:
  - 0: active scan
  - 1: passive scan
- **<scan_time_min>**: the minimum active scan time per channel. Unit: millisecond. Range [0,1500]. If the scan type is passive, this parameter is invalid.
- **<scan_time_max>**: the maximum active scan time per channel. Unit: millisecond. Range [0,1500]. If this parameter is 0, the firmware will use the default time: 120 ms for active scan; 360 ms for passive scan.
- **<freq_offset>**: frequency offset (reserved item).
- **<freqcal_val>**: frequency calibration value (reserved item).

- **<pairwise_cipher>**: pairwise cipher type.
  - 0: None
  - 1: WEP40
  - 2: WEP104
  - 3: TKIP
  - 4: CCMP
  - 5: TKIP and CCMP
  - 6: AES-CMAC-128
  - 7: Unknown
- **<group_cipher>**: group cipher type, same enumerated value to `<pairwise_cipher>`.
- **<bgn>**: 802.11 b/g/n. If the corresponding bit is 1, the corresponding mode is enabled; if the corresponding bit is 0, the corresponding mode is disabled.
  - bit 0: bit to identify if 802.11b mode is enabled or not
  - bit 1: bit to identify if 802.11g mode is enabled or not
  - bit 2: bit to identify if 802.11n mode is enabled or not
- **<wps>**: wps flag.
  - 0: WPS disabled
  - 1: WPS enabled

Example

```
AT+CWLAP="Wi-Fi","ca:d7:19:d8:a6:44",6,0,400,1000

// Search for APs with a designated SSID:
AT+CWLAP="Wi-Fi"
```

3.2.7 AT+CWQAP: Disconnect from an AP

Execute Command

Command:

```
AT+CWQAP
```

Response:

```
OK
```

3.2.8 AT+CWSAP: Query/Set the configuration of an ESP SoftAP

Query Command

Function:

Query the configuration parameters of an ESP SoftAP.

Command:

```
AT+CWSAP?
```

Response:

```
+CWSAP:<ssid>,<pwd>,<channel>,<ecn>,<max conn>,<ssid hidden>
OK
```

Set Command

Function:

Set the configuration of an ESP SoftAP.

Command:

```
AT+CWSAP=<ssid>,<pwd>,<chl>,<ecn>[,<max conn>][,<ssid hidden>]
```

Response:

```
OK
```

Parameters

- **<ssid>**: string parameter showing SSID of the AP.
- **<pwd>**: string parameter showing the password. Length: 8 ~ 64 bytes ASCII.
- **<channel>**: channel ID.
- **<ecn>**: encryption method; WEP is not supported.
  - 0: OPEN
  - 2: WPA_PSK
  - 3: WPA2_PSK
  - 4: WPA_WPA2_PSK
- **[<max conn>]**: maximum number of stations that ESP SoftAP can connect. Range: [1,10].
- **[<ssid hidden>]**:
  - 0: broadcasting SSID (default).
  - 1: not broadcasting SSID.

Notes

- This command works only when *AT+CWMODE=2* or *AT+CWMODE=3*.
- The configuration changes will be saved in the NVS area if *AT+SYSSTORE=1*.

Example

```
AT+CWSAP="ESP","1234567890",5,3
```

3.2.9 AT+CWLIF: Obtain IP Address of the Station That Connects to an ESP SoftAP

Execute Command

Command:

```
AT+CWLIF
```

Response:

```
+CWLIF:<ip addr>,<mac>

OK
```

Parameters

- **<ip addr>**: IP address of the station that connects to the ESP SoftAP.
- **<mac>**: MAC address of the station that connects to the ESP SoftAP.

Note

- This command cannot get a static IP. It works only when DHCP of both the ESP SoftAP and the connected station are enabled.

3.2.10 AT+CWQIF: Disconnect Stations from an ESP SoftAP

Execute Command

Function:

Disconnect all stations that are connected to the ESP SoftAP.

Command:

```
AT+CWQIF
```

Response:

```
OK
```

Set Command

Function:

Disconnect a specific station from the ESP SoftAP.

Command:

```
AT+CWQIF=<mac>
```

Response:

```
OK
```

Parameter

- **<mac>**: MAC address of the station to disconnect.

3.2.11 AT+CWDHCP: Enable/Disable DHCP

Query Command

Command:

```
AT+CWDHCP?
```

Response:

```
<state>
```

Set Command

Function:

Enable/disable DHCP.

Command:

```
AT+CWDHCP=<operate>,<mode>
```

Response:

```
OK
```

Parameters

- **<operate>**:
  - 0: disable
  - 1: enable
- **<mode>**:
  - Bit0: Station DHCP
  - Bit1: SoftAP DHCP
- **<state>**: the status of DHCP
  - Bit0:
    - 0: Station DHCP is disabled.

    * 1: Station DHCP is enabled.

  - Bit1:
    * 0: SoftAP DHCP is disabled.
    * 1: SoftAP DHCP is enabled.
  - Bit2 (ESP32 only):
    * 0: Ethernet DHCP is disabled.
    * 1: Ethernet DHCP is enabled.

Notes

- The configuration changes will be saved in the NVS area if *AT+SYSSTORE=1*.
- This Set Command correlates with the commands that set static IP, such as *AT+CIPSTA* and *AT+CIPAP*:
  - If DHCP is enabled, static IP address will be disabled;
  - If static IP address is enabled, DHCP will be disabled;
  - The last configuration overwrites the previous configuration.

Example

```
// Enable Station DHCP. If the last DHCP mode is 2, the current DHCP mode will be 3.
AT+CWDHCP=1,1

// Disable SoftAP DHCP. If the last DHCP mode is 3, the current DHCP mode will be 1.
AT+CWDHCP=0,2
```

3.2.12 AT+CWDHCPS: Query/Set the IP Addresses Allocated by an ESP SoftAP DHCP Server

Query Command

Command:

```
AT+CWDHCPS?
```

Response:

```
+CWDHCPS=<lease time>,<start IP>,<end IP>
OK
```

Set Command

Function:

Set the IP address range of the ESP SoftAP DHCP server.

Command:

```
AT+CWDHCPS=<enable>,<lease time>,<start IP>,<end IP>
```

Response:

```
OK
```

Parameters

- **<enable>**:
  - 1: Enable DHCP server settings. The parameters below have to be set.
  - 0: Disable DHCP server settings and use the default IP address range.
- **<lease time>**: lease time. Unit: minute. Range [1,2880].
- **<start IP>**: start IP address of the IP address range that can be obtained from ESP SoftAP DHCP server.
- **<end IP>**: end IP address of the IP address range that can be obtained from ESP SoftAP DHCP server.

Notes

- The configuration changes will be saved in the NVS area if *AT+SYSSTORE=1*.
- This AT command works only when both SoftAP and DHCP server are enabled for ESP devices.
- The IP address should be in the same network segment as the IP address of ESP SoftAP.

Example

```
AT+CWDHCPS=1,3,"192.168.4.10","192.168.4.15"

AT+CWDHCPS=0 // Disable the settings and use the default IP address range.
```

3.2.13 AT+CWAUTOCONN: Automatically Connect to an AP When Powered on

Set Command

Command:

```
AT+CWAUTOCONN=<enable>
```

Response:

```
OK
```

Parameters

- **<enable>**:
  - 1: Enable automatic connection to an AP when powered on. (Default)
  - 0: Disable automatic connection to an AP when powered on.

Note

- The configuration changes will be saved in the NVS area.

Example

```
AT+CWAUTOCONN=1
```

3.2.14 AT+CWAPPROTO: Query/Set the 802.11 b/g/n Protocol Standard of SoftAP Mode

Query Command

Command:

```
AT+CWAPPROTO?
```

Response:

```
+CWAPPROTO=<protocol>
OK
```

Set Command

Command:

```
AT+CWAPPROTO=<protocol>
```

Response:

```
OK
```

Parameters

- **<protocol>**:
  - bit0: 802.11b protocol standard.
  - bit1: 802.11g protocol standard.
  - bit2: 802.11n protocol standard.

Note

- Currently ESP devices only support 802.11b or 802.11bg or 802.11bgn mode.
- By default, PHY mode of ESP8266 device is 802.11bg mode, and PHY mode of non ESP8266 device is 802.11bgn mode.

3.2.15 AT+CWSTAPROTO: Query/Set the 802.11 b/g/n Protocol Standard of Station Mode

Query Command

Command:

```
AT+CWSTAPROTO?
```

Response:

```
+CWSTAPROTO=<protocol>
OK
```

Set Command

Command:

```
AT+CWSTAPROTO=<protocol>
```

Response:

```
OK
```

Parameters

- **<protocol>**:
  - bit0: 802.11b protocol standard.
  - bit1: 802.11g protocol standard.
  - bit2: 802.11n protocol standard.

Note

- Currently ESP devices only support 802.11b or 802.11bg or 802.11bgn mode.
- By default, PHY mode of ESP8266 device is 802.11bg mode, and PHY mode of non ESP8266 device is 802.11bgn mode.

3.2.16 AT+CIPSTAMAC: Query/Set the MAC Address of an ESP Station

Query Command

Function:

Query the MAC address of the ESP Station.

Command:

```
AT+CIPSTAMAC?
```

Response:

```
+CIPSTAMAC:<mac>
OK
```

Set Command

Function:

Set the MAC address of an ESP station.

Command:

```
AT+CIPSTAMAC=<mac>
```

Response:

```
OK
```

Parameters

- **<mac>**: string parameter showing MAC address of an ESP station.

Notes

- The configuration changes will be saved in the NVS area if *AT+SYSSTORE=1*.
- The MAC address of ESP SoftAP is different from that of the ESP Station. Please make sure that you do not set the same MAC address for both of them.
- Bit 0 of the ESP MAC address CANNOT be 1. For example, a MAC address can be "1a:. . . " but not "15:. . . ".
- FF:FF:FF:FF:FF:FF and 00:00:00:00:00:00 are invalid MAC address and cannot be set.

Example

```
AT+CIPSTAMAC="1a:fe:35:98:d3:7b"
```

3.2.17 AT+CIPAPMAC: Query/Set the MAC Address of an ESP SoftAP

Query Command

Function:

Query the MAC address of the ESP SoftAP.

Command:

```
AT+CIPAPMAC?
```

Response:

```
+CIPAPMAC:<mac>
OK
```

Set Command

Function:

Set the MAC address of the ESP SoftAP.

Command:

```
AT+CIPAPMAC=<mac>
```

Response:

```
OK
```

Parameters

- **<mac>**: string parameter showing MAC address of the ESP SoftAP.

Notes

- The configuration changes will be saved in the NVS area if *AT+SYSSTORE=1*.
- The MAC address of ESP SoftAP is different from that of the ESP station. Please make sure that you do not set the same MAC address for both of them.
- Bit 0 of the ESP MAC address CANNOT be 1. For example, a MAC address can be "18:. . . " but not "15:. . . ".
- FF:FF:FF:FF:FF:FF and 00:00:00:00:00:00 are invalid MAC and cannot be set.

Example

```
AT+CIPAPMAC="18:fe:35:98:d3:7b"
```

3.2.18 AT+CIPSTA: Query/Set the IP Address of an ESP Station

Query Command

Function:

Query the IP address of the ESP Station.

Command:

```
AT+CIPSTA?
```

Response:

```
+CIPSTA:ip:<"ip">
+CIPSTA:gateway:<"gateway">
+CIPSTA:netmask:<"netmask">
+CIPSTA:ip6ll:<"ipv6 addr">
+CIPSTA:ip6gl:<"ipv6 addr">

OK
```

Set Command

Function:

Set the IPv4 address of the ESP station.

Command:

```
AT+CIPSTA=<"ip">[,<"gateway">,<"netmask">]
```

Response:

```
OK
```

Parameters

- **<"ip">**: string parameter showing the IPv4 address of the ESP station.
- **<"gateway">**: gateway.
- **<"netmask">**: netmask.
- **<"ipv6 addr">**: string parameter showing the IPv6 address of the ESP station.

Notes

- For the query command, only when the ESP station is connected to an AP or the static IP address is configured can its IP address be queried.
- The configuration changes will be saved in the NVS area if *AT+SYSSTORE=1*.
- The Set Command correlates with the commands that set DHCP, such as *AT+CWDHCP*.
  - If static IP address is enabled, DHCP will be disabled;
  - If DHCP is enabled, static IP address will be disabled;
  - The last configuration overwrites the previous configuration.

Example

```
AT+CIPSTA="192.168.6.100","192.168.6.1","255.255.255.0"
```

3.2.19 AT+CIPAP: Query/Set the IP Address of an ESP SoftAP

Query Command

Function:

Query the IP address of the ESP SoftAP.

Command:

```
AT+CIPAP?
```

Response:

```
+CIPAP:ip:<"ip">
+CIPAP:gateway:<"gateway">
+CIPAP:netmask:<"netmask">
+CIPAP:ip6ll:<"ipv6 addr">
+CIPAP:ip6gl:<"ipv6 addr">

OK
```

Set Command

Function:

Set the IPv4 address of the ESP SoftAP.

Command:

```
AT+CIPAP=<"ip">[,<"gateway">,<"netmask">]
```

Response:

```
OK
```

Parameters

- **<"ip">**: string parameter showing the IPv4 address of the ESP SoftAP.
- **<"gateway">**: gateway.
- **<"netmask">**: netmask.
- **<"ipv6 addr">**: string parameter showing the IPv6 address of the ESP SoftAP.

Notes

- The configuration changes will be saved in the NVS area if *AT+SYSSTORE=1*.
- The set command correlates with the commands that set DHCP, such as *AT+CWDHCP*.
  - If static IP address is enabled, DHCP will be disabled;
  - If DHCP is enabled, static IP address will be disabled;
  - The last configuration overwrites the previous configuration.

Example

```
AT+CIPAP="192.168.5.1","192.168.5.1","255.255.255.0"
```

3.2.20 AT+CWSTARTSMART: Start SmartConfig

Execute Command

Function:

Start SmartConfig of the type ESP-TOUCH+AirKiss.

Command:

```
AT+CWSTARTSMART
```

Set Command

Function:

Start SmartConfig of a designated type.

Command:

```
AT+CWSTARTSMART=<type>[,<auth floor>]
```

Response:

```
OK
```

Parameters

- **<type>**:
  - 1: ESP-TOUCH
  - 2: AirKiss
  - 3: ESP-TOUCH+AirKiss
- **<auth floor>**: Wi-Fi authentication mode floor. ESP-AT will not connect to the AP whose authmode is lower than this floor.
  - 0: OPEN (Default)
  - 1: WEP
  - 2: WPA_PSK
  - 3: WPA2_PSK
  - 4: WPA_WPA2_PSK
  - 5: WPA2_ENTERPRISE
  - 6: WPA3_PSK
  - 7: WPA2_WPA3_PSK

Notes

- For more details on SmartConfig, please see ESP-TOUCH User Guide.
- SmartConfig is only available in the ESP station mode.
- The message `Smart get Wi-Fi info` means that SmartConfig has successfully acquired the AP information. ESP device will try to connect to the target AP.
- Message `Smartconfig connected Wi-Fi` is printed if the connection is successful.

- When AT returns `Smartconfig connected Wi-Fi`, it is recommended to delay more than `6` seconds before executing *AT+CWSTOPSMART* because the ESP device needs to synchronize the SmartConfig results to the mobile phone.
- Use command *AT+CWSTOPSMART* to stop SmartConfig before running other commands. Please make sure that you do not execute other commands during SmartConfig.

Example

```
AT+CWMODE=1
AT+CWSTARTSMART
```

3.2.21 AT+CWSTOPSMART: Stop SmartConfig

Execute Command

Command:

```
AT+CWSTOPSMART
```

Response:

```
OK
```

Note

- Irrespective of whether SmartConfig succeeds or not, please always call *AT+CWSTOPSMART* before executing any other AT commands to release the internal memory taken up by SmartConfig.

Example

```
AT+CWMODE=1
AT+CWSTARTSMART
AT+CWSTOPSMART
```

3.2.22 AT+WPS: Enable the WPS Function

Set Command

Command:

```
AT+WPS=<enable>[,<auth floor>]
```

Response:

```
OK
```

Parameters

- **<enable>**:
  - 1: Enable WPS (Wi-Fi Protected Setup) that uses PBC (Push Button Configuration) mode.
  - 0: Disable WPS that uses PBC mode.
- **<auth floor>**: Wi-Fi authentication mode floor. ESP-AT will not connect to the AP whose authmode is lower than this floor.
  - 0: OPEN (Default)
  - 1: WEP
  - 2: WPA_PSK
  - 3: WPA2_PSK
  - 4: WPA_WPA2_PSK
  - 5: WPA2_ENTERPRISE
  - 6: WPA3_PSK
  - 7: WPA2_WPA3_PSK

Notes

- WPS can only be used when the ESP station is enabled.
- WPS does not support WEP (Wired-Equivalent Privacy) encryption.

Example

```
AT+CWMODE=1
AT+WPS=1
```

3.2.23 AT+MDNS: Configure the mDNS Function

Set Command

Command:

```
AT+MDNS=<enable>[,<hostname>,<service_name>,<port>]
```

Response:

```
OK
```

Parameters

- **<enable>**:
  - 1: Enable the mDNS function. The following three parameters need to be set.
  - 0: Disable the mDNS function. The following three parameters does not need to be set.

- **<hostname>**: mDNS host name.
- **<service_name>**: mDNS service name.
- **<port>**: mDNS port.

Example

```
AT+MDNS=1,"espressif","_iot",8080
AT+MDNS=0
```

3.2.24 [ESP32 Only] AT+CWJEAP: Connect to a WPA2 Enterprise AP

Query Command

Function:

Query the configuration information of the Enterprise AP to which the ESP station is already connected.

Command:

```
AT+CWJEAP?
```

Response:

```
+CWJEAP:<ssid>,<method>,<identity>,<username>,<password>,<security>
OK
```

Set Command

Function:

Connect to the targeted Enterprise AP.

Command:

```
AT+CWJEAP=<ssid>,<method>,<identity>,<username>,<password>,<security>[,<jeap_timeout>]
```

Response:

```
OK
```

or

```
+CWJEAP:Timeout
ERROR
```

Parameters

- **<ssid>**: the SSID of the Enterprise AP.
  - Escape character syntax is needed if SSID or password contains any special characters, such as `,`, `"`, or `\\`.
- **<method>**: WPA2 Enterprise authentication method.

  - 0: EAP-TLS.
  - 1: EAP-PEAP.
  - 2: EAP-TTLS.
- **<identity>**: identity for phase 1. String limited to 1 ~ 32.
- **<username>**: username for phase 2. Range: 1 ~ 32 bytes. For the EAP-PEAP and EAP-TTLS method, you must set this parameter. For the EAP-TLS method, you don't need to.
- **<password>**: password for phase 2. Range: 1 ~ 32 bytes. For the EAP-PEAP and EAP-TTLS method, you must set this parameter. For the EAP-TLS method, you don't need to.
- **<security>**:
  - Bit0: Client certificate.
  - Bit1: Server certificate.
- **[<jeap_timeout>]**: maximum timeout for *AT+CWJEAP* command. Unit: second. Default: 15. Range: [3,600].

Example

```
// Connect to EAP-TLS mode Enterprise AP, set identity, verify server certificate and␣
↪load client certificate
AT+CWJEAP="dlink11111",0,"example@espressif.com",,,3

// Connect to EAP-PEAP mode Enterprise AP, set identity, username and password, not␣
↪verify server certificate and not load client certificate
AT+CWJEAP="dlink11111",1,"example@espressif.com","espressif","test11",0
```

Error Code:

The WPA2 Enterprise error code will be prompt as `ERR CODE:0x<%08x>`.

AT_EAP_MALLOC_FAILED	0x8001
AT_EAP_GET_NVS_CONFIG_FAILED	0x8002
AT_EAP_CONN_FAILED	0x8003
AT_EAP_SET_WIFI_CONFIG_FAILED	0x8004
AT_EAP_SET_IDENTITY_FAILED	0x8005
AT_EAP_SET_USERNAME_FAILED	0x8006
AT_EAP_SET_PASSWORD_FAILED	0x8007
AT_EAP_GET_CA_LEN_FAILED	0x8008
AT_EAP_READ_CA_FAILED	0x8009
AT_EAP_SET_CA_FAILED	0x800A
AT_EAP_GET_CERT_LEN_FAILED	0x800B
AT_EAP_READ_CERT_FAILED	0x800C
AT_EAP_GET_KEY_LEN_FAILED	0x800D
AT_EAP_READ_KEY_FAILED	0x800E
AT_EAP_SET_CERT_KEY_FAILED	0x800F
AT_EAP_ENABLE_FAILED	0x8010
AT_EAP_ALREADY_CONNECTED	0x8011
AT_EAP_GET_SSID_FAILED	0x8012
AT_EAP_SSID_NULL	0x8013
AT_EAP_SSID_LEN_ERROR	0x8014
AT_EAP_GET_METHOD_FAILED	0x8015

Continued on next page

Table 2 – continued from previous page

AT_EAP_MALLOC_FAILED	0x8001
AT_EAP_CONN_TIMEOUT	0x8016
AT_EAP_GET_IDENTITY_FAILED	0x8017
AT_EAP_IDENTITY_LEN_ERROR	0x8018
AT_EAP_GET_USERNAME_FAILED	0x8019
AT_EAP_USERNAME_LEN_ERROR	0x801A
AT_EAP_GET_PASSWORD_FAILED	0x801B
AT_EAP_PASSWORD_LEN_ERROR	0x801C
AT_EAP_GET_SECURITY_FAILED	0x801D
AT_EAP_SECURITY_ERROR	0x801E
AT_EAP_METHOD_SECURITY_UNMATCHED	0x801F
AT_EAP_PARAMETER_COUNTS_ERROR	0x8020
AT_EAP_GET_WIFI_MODE_ERROR	0x8021
AT_EAP_WIFI_MODE_NOT_STA	0x8022
AT_EAP_SET_CONFIG_FAILED	0x8023
AT_EAP_METHOD_ERROR	0x8024

Note

- The configuration changes will be saved in the NVS area if *AT+SYSSTORE=1*.
- This command requires Station mode to be active.
- TLS mode will use client certificate. Please make sure it is enabled.

3.2.25 AT+CWHOSTNAME: Query/Set the Host Name of an ESP Station

Query Command

Function:

Query the host name of ESP Station.

Command:

```
AT+CWHOSTNAME?
```

Response:

```
+CWHOSTNAME:<hostname>

OK
```

Set Command

Function:

Set the host name of ESP Station.

Command:

```
AT+CWHOSTNAME=<hostname>
```

Response:

```
OK
```

If the Station mode is not enabled, the command will return:

```
ERROR
```

Parameters

- **<hostname>**: the host name of the ESP Station. Maximum length: 32 bytes.

Note

- The configuration changes are not saved in the flash.

Example

```
AT+CWMODE=3
AT+CWHOSTNAME="my_test"
```

3.2.26 AT+CWCOUNTRY: Query/Set the Wi-Fi Country Code

Query Command

Function:

Query Wi-Fi country code information.

Command:

```
AT+CWCOUNTRY?
```

Response:

```
+CWCOUNTRY:<country_policy>,<country_code>,<start_channel>,<total_channel_count>

OK
```

Set Command

Function:

Set the Wi-Fi country code information.

Command:

```
AT+ CWCOUNTRY=<country_policy>,<country_code>,<start_channel>,<total_channel_count>
```

Response:

```
OK
```

Parameters

- **<country_policy>**:
  - 0: will change the county code to be the same as the AP that the ESP device is connected to.
  - 1: the country code will not change, always be the one set by command.
- **<country_code>**: country code. Maximum length: 3 characters.
- **<start_channel>**: the channel number to start. Range: [1,14].
- **<total_channel_count>**: total number of channels.

Note

- The configuration changes are not saved in the flash.

Example

```
AT+CWMODE=3
AT+CWCOUNTRY=1,"CN",1,13
```

3.3 TCP/IP AT Commands

[]

- *AT+CIPV6*: Enable/disable the network of Internet Protocol Version 6 (IPv6).
- *AT+CIPSTATUS*: Obtain the TCP/UDP/SSL connection status and information.
- *AT+CIPDOMAIN*: Resolve a Domain Name.
- *AT+CIPSTART*: Establish TCP connection, UDP transmission, or SSL connection.
- *AT+CIPSTARTEX*: Establish TCP connection, UDP transmission, or SSL connection with an automatically assigned ID.
- *[Passthrough Mode Only] +++*: Exit the *passthrough mode*.
- *AT+CIPSEND*: Send data in the *normal transmission mode* or Wi-Fi *passthrough mode*.
- *AT+CIPSENDEX*: Send data in the *normal transmission mode* in expanded ways.
- *AT+CIPCLOSE*: Close TCP/UDP/SSL connection.
- *AT+CIFSR*: Obtain the local IP address and MAC address.
- *AT+CIPMUX*: Enable/disable the multiple connections mode.
- *AT+CIPSERVER*: Delete/create a TCP/SSL server.
- *AT+CIPSERVERMAXCONN*: Query/Set the maximum connections allowed by a server.
- *AT+CIPMODE*: Query/Set the transmission mode.
- *AT+SAVETRANSLINK*: Set whether to enter Wi-Fi *passthrough mode* on power-up.

- *AT+CIPSTO*: Query/Set the local TCP Server Timeout.
- *AT+CIPSNTPCFG*: Query/Set the time zone and SNTP server.
- *AT+CIPSNTPTIME*: Query the SNTP time.
- *AT+CIUPDATE*: Upgrade the firmware through Wi-Fi.
- *AT+CIPDINFO*: Set "+IPD" message mode.
- *AT+CIPSSLCCONF*: Query/Set SSL clients.
- *AT+CIPSSLCCN*: Query/Set the Common Name of the SSL client.
- *AT+CIPSSLCSNI*: Query/Set SSL client Server Name Indication (SNI).
- *AT+CIPSSLCALPN*: Query/Set SSL client Application Layer Protocol Negotiation (ALPN).
- *AT+CIPSSLCPSK*: Query/Set SSL client Pre-shared Key (PSK).
- *AT+CIPRECONNINTV*: Query/Set the TCP/UDP/SSL reconnection interval for the Wi-Fi *passthrough mode*.
- *AT+CIPRECVMODE*: Query/Set socket receiving mode.
- *AT+CIPRECVDATA*: Obtain socket data in passive receiving mode.
- *AT+CIPRECVLEN*: Obtain socket data length in passive receiving mode.
- *AT+PING*: Ping the remote host.
- *AT+CIPDNS*: Query/Set DNS server information.
- *AT+CIPTCPOPT*: Query/Set the socket options.

3.3.1 AT+CIPV6: Enable/disable the network of Internet Protocol Version 6 (IPv6)

Query Command

Function:

Query whether IPv6 is enabled.

Command:

```
AT+CIPV6?
```

Response:

```
+CIPV6:<enable>

OK
```

Set Command

Function:

Enable/Disable IPv6 network.

Command:

```
AT+CIPV6=<enable>
```

Response:

```
OK
```

Parameters

- **<enable>**: status of IPv6 network. Default: 0.
  - 0: disable IPv6 network.
  - 1: enable IPv6 network.

Notes

- You should enable IPv6 network before using IPv6 related upper layer AT commands (TCP/UDP/SSL/PING/DNS based on IPv6 network, also known as TCP6/UDP6/SSL6/PING6/DNS6 or TCPv6/UDPv6/SSLv6/PINGv6/DNSv6).

3.3.2 AT+CIPSTATUS: Obtain the TCP/UDP/SSL Connection Status and Information

Execute Command

Command:

```
AT+CIPSTATUS
```

Response:

```
STATUS:<stat>
+CIPSTATUS:<link ID>,<"type">,<"remote IP">,<remote port>,<local port>,<tetype>
OK
```

Parameters

- **<stat>**: status of the ESP station interface.
  - 0: The ESP station is not initialized.
  - 1: The ESP station is initialized, but not started a Wi-Fi connection yet.
  - 2: The ESP station is connected to an AP and its IP address is obtained.
  - 3: The ESP station has created a TCP/SSL transmission.
  - 4: All of the TCP/UDP/SSL connections of the ESP device station are disconnected.
  - 5: The ESP station started a Wi-Fi connection, but was not connected to an AP or disconnected from an AP.
- **<link ID>**: ID of the connection (0~4), used for multiple connections.
- **<"type">**: string parameter showing the type of transmission: "TCP", "TCPv6", "UDP", "UDPv6", "SSL", or "SSLv6".
- **<"remote IP">**: string parameter showing the remote IPv4 address or IPv6 address.
- **<remote port>**: the remote port number.
- **<local port>**: the local port number.

- **<tetype>**:
  - 0: ESP device runs as a client.
  - 1: ESP device runs as a server.

3.3.3 AT+CIPDOMAIN: Resolve a Domain Name

Set Command

Command:

```
AT+CIPDOMAIN=<"domain name">[,<ip network>]
```

Response:

```
+CIPDOMAIN:<"IP address">

OK
```

Parameter

- **<"domain name">**: the domain name.
- **<ip network>**: preferred IP network. Default: 1.
  - 1: preferred resolution of IPv4 address
  - 2: resolve IPv4 address only
  - 3: resolve IPv6 address only
- **<"IP address">**: the resolved IPv4 address or IPv6 address.

Example

```
AT+CWMODE=1                         // set the station mode
AT+CWJAP="SSID","password"          // access to the internet
AT+CIPDOMAIN="iot.espressif.cn"     // Domain Name Resolution function

// Domain Name Resolution Function for IPv4 address only
AT+CIPDOMAIN="iot.espressif.cn",2

// Domain Name Resolution Function for IPv6 address only
AT+CIPDOMAIN="ipv6.test-ipv6.com",3

// Domain Name Resolution Function for compatible IP address
AT+CIPDOMAIN="ds.test-ipv6.com",1
```

3.3.4 AT+CIPSTART: Establish TCP Connection, UDP Transmission, or SSL Connection

Establish TCP Connection

Set Command

Command:

```
// Single connection (AT+CIPMUX=0):
AT+CIPSTART=<"type">,<"remote host">,<remote port>[,<keep alive>][,<"local IP">]

// Multiple Connections (AT+CIPMUX=1):
AT+CIPSTART=<link ID>,<"type">,<"remote host">,<remote port>[,<keep alive>][,<"local␣
↪IP">]
```

Response:

```
CONNECT

OK
```

Parameters

- **<link ID>**: ID of network connection (0~4), used for multiple connections.
- **<"type">**: string parameter showing the type of transmission: "TCP", or "TCPv6". Default: "TCP".
- **<"remote host">**: string parameter showing the IPv4 address or IPv6 address or domain name of remote host.
- **<remote port>**: the remote port number.
- **<keep alive>**: TCP keep-alive interval. Default: 0.
  - 0: disable TCP keep-alive function.
  - 1 ~ 7200: detection interval. Unit: second.
- **<"local IP">**: the local IPv4 address or IPv6 address that the connection binds. This parameter is useful when you are using multiple network interfaces or multiple IP addresses. By default, it is disabled. If you want to use it, you should specify it first. Null is also valid.

Notes

- If you want to establish TCP connection based on IPv6 network, set *AT+CIPV6=1* first, and ensure the connected AP by *AT+CWJAP* supports IPv6 and esp-at got the IPv6 address which you can check it by AT+CIPSTA.
- `<keep alive>` parameter will eventually be configured to the socket option `TCP_KEEPIDLE`. As for other socket options of keepalive, `TCP_KEEPINTVL` will use `1` by default, and `TCP_KEEPCNT` will use `3` by default.

Example

```
AT+CIPSTART="TCP","iot.espressif.cn",8000
AT+CIPSTART="TCP","192.168.101.110",1000
AT+CIPSTART="TCP","192.168.101.110",1000,,"192.168.101.100"
AT+CIPSTART="TCPv6","test-ipv6.com",80
AT+CIPSTART="TCPv6","fe80::860d:8eff:fe9d:cd90",1000,,"fe80::411c:1fdb:22a6:4d24"

// esp-at has obtained an IPv6 global address by AT+CWJAP before
AT+CIPSTART="TCPv6","2404:6800:4005:80b::2004",80,,
↪"240e:3a1:2070:11c0:32ae:a4ff:fe80:65ac"
```

(continues on next page)

(continued from previous page)

Establish UDP Transmission

Set Command

Command:

```
// Single connection (AT+CIPMUX=0):
AT+CIPSTART=<"type">,<"remote host">,<remote port>[,<local port>,<mode>,<"local IP">]

// Multiple connections (AT+CIPMUX=1):
AT+CIPSTART=<link ID>,<"type">,<"remote host">,<remote port>[,<local port>,<mode>,<
↪"local IP">]
```

Response:

```
CONNECT

OK
```

Parameters

- **<link ID>**: ID of network connection (0~4), used for multiple connections.
- **<"type">**: string parameter showing the type of transmission: "UDP", or "UDPv6". Default: "TCP".
- **<"remote host">**: string parameter showing the IPv4 address or IPv6 address or domain name of remote host.
- **<remote port>**: remote port number.
- **<local port>**: UDP port of ESP devices.
- **<mode>**: In the UDP Wi-Fi passthrough, the value of this parameter has to be 0.
  - 0: After UDP data is received, the parameters `<"remote host">` and `<remote port>` will stay unchanged (default).
  - 1: Only the first time that UDP data is received from an IP address and port that are different from the initially set value of parameters `<remote host>` and `<remote port>`, will they be changed to the IP address and port of the device that sends the data.
  - 2: Each time UDP data is received, the `<"remote host">` and `<remote port>` will be changed to the IP address and port of the device that sends the data.
- **<"local IP">**: the local IPv4 address or IPv6 address that the connection binds. This parameter is useful when you are using multiple network interfaces or multiple IP addresses. By default, it is disabled. If you want to use it, you should specify it first. Null is also valid.

Notes

- If the remote host over the UDP is an IPv4 multicast address (224.0.0.0 ~ 239.255.255.255), the ESP device will send and receive the UDPv4 multicast.

- If the remote host over the UDP is an IPv4 broadcast address (255.255.255.255), the ESP device will send and receive the UDPv4 broadcast.
- If the remote host over the UDP is an IPv6 multicast address (FF00:0:0:0:0:0:0:0 ~ FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF), the ESP device will send and receive the UDP multicast based on IPv6 network.
- To use the parameter `<mode>`, parameter `<local port>` must be set first.
- If you want to establish UDP connection based on IPv6 network, set *AT+CIPV6=1* first, and ensure the connected AP by *AT+CWJAP* supports IPv6 and esp-at got the IPv6 address which you can check it by AT+CIPSTA.

Example

```
// UDP unicast
AT+CIPSTART="UDP","192.168.101.110",1000,1002,2
AT+CIPSTART="UDP","192.168.101.110",1000,,,"192.168.101.100"

// UDP unicast based on IPv6 network
AT+CIPSTART="UDPv6","fe80::32ae:a4ff:fe80:65ac",1000,,,"fe80::5512:f37f:bb03:5d9b"

// UDP multicast based on IPv6 network
AT+CIPSTART="UDPv6","FF02::FC",1000,1002,0
```

Establish SSL Connection

Set Command

Command:

```
AT+CIPSTART=[<link ID>,]<"type">,<"remote host">,<remote port>[,<keep alive>,<"local␣
↪IP">]
```

Response:

```
OK
```

Parameters

- **<link ID>**: ID of network connection (0~4), used for multiple connections.
- **<"type">**: string parameter showing the type of transmission: "SSL", or "SSLv6". Default: "TCP".
- **<"remote host">**: string parameter showing the IPv4 address or IPv6 address or domain name of remote host.
- **<remote port>**: the remote port number.
- **<keep alive>**: reserved item for SSL. Default: 0.
- **<"local IP">**: the local IPv4 address or IPv6 address that the connection binds. This parameter is useful when you are using multiple network interfaces or multiple IP addresses. By default, it is disabled. If you want to use it, you should specify it first. Null is also valid.

Notes

- The number of SSL connections depends on available memory and the maximum number of connections. For ESP8266 devices, only one SSL connection can be established due to limited memory.
- SSL connection needs a large amount of memory. Insufficient memory may cause the system reboot.
- If the `AT+CIPSTART` is based on an SSL connection and the timeout of each packet is 10 s, the total timeout will be much longer depending on the number of handshake packets.
- If you want to establish SSL connection based on IPv6 network, set *AT+CIPV6=1* first, and ensure the connected AP by *AT+CWJAP* supports IPv6 and esp-at got the IPv6 address which you can check it by AT+CIPSTA.
- `<keep alive>` parameter will eventually be configured to the socket option `TCP_KEEPIDLE`. As for other socket options of keepalive, `TCP_KEEPINTVL` will use `1` by default, and `TCP_KEEPCNT` will use `3` by default.

Example

```
AT+CIPSTART="SSL","iot.espressif.cn",8443
AT+CIPSTART="SSL","192.168.101.110",1000,,"192.168.101.100"

// esp-at has obtained an IPv6 global address by AT+CWJAP before
AT+CIPSTART="SSLv6","240e:3a1:2070:11c0:6972:6f96:9147:d66d",1000,,
↪"240e:3a1:2070:11c0:55ce:4e19:9649:b75"
```

3.3.5 AT+CIPSTARTEX: Establish TCP connection, UDP transmission, or SSL connection with an Automatically Assigned ID

This command is similar to *AT+CIPSTART* except that you don't need to assign an ID by yourself in multiple connections mode (*AT+CIPMUX=1*). The system will assign an ID to the new connection automatically.

3.3.6 [Passthrough Mode Only] +++: Exit from Passthrough Mode

Special Execute Command

Function:

Exit from *Passthrough Mode* and enter the *Passthrough Receiving Mode*.

Command:

```
// Only for passthrough mode
+++
```

Notes

- This special execution command consists of three identical + characters (0x2b ASCII), and no CR-LF appends to the command tail.
- Make sure there is more than 20 ms interval before the first + character, more than 20 ms interval after the third + character, less than 20 ms interval among the three + characters. Otherwise, the + characters will be sent out as normal passthrough data.

- This command returns no reply.

3.3.7 AT+CIPSEND: Send Data in the Normal Transmission Mode or Wi-Fi Passthrough Mode

Set Command

Function:

Set the data length to be send in the *Normal Transmission Mode*.

Command:

```
// Single connection: (AT+CIPMUX=0)
AT+CIPSEND=<length>

// Multiple connections: (AT+CIPMUX=1)
AT+CIPSEND=<link ID>,<length>

// Remote host and port can be set for UDP transmission:
AT+CIPSEND=[<link ID>,]<length>[,<"remote host">,<remote port>]
```

Response:

```
OK

>
```

This response indicates that AT is ready for receiving serial data. You should enter the data, and when the data length reaches the `<length>` value, the transmission of data starts.

If the connection cannot be established or is disrupted during data transmission, the system returns:

```
ERROR
```

If data is transmitted successfully, the system returns:

```
SEND OK
```

Execute Command

Function:

Enter the Wi-Fi *Passthrough Mode*.

Command:

```
AT+CIPSEND
```

Response:

```
OK
>
```

or

```
ERROR
```

Enter the Wi-Fi *Passthrough Mode*. The ESP8266 devices can receive 2048 bytes and send 1460 bytes at most each time; the other ESP devices can receive 8192 bytes and send 2920 bytes at most each time. If the length of the currently received data is greater than the maximum number of bytes that can be sent, AT will send the received data immediately; Otherwise, the received data will be sent out within 20 ms. When a single packet containing `+++` is received, the ESP device will exit the data sending mode under the Wi-Fi *Passthrough Mode*. Please wait for at least one second before sending the next AT command.

This command can only be used for single connection in the Wi-Fi *Passthrough Mode*. For UDP Wi-Fi passthrough, the `<mode>` parameter has to be 0 when using *AT+CIPSTART*.

Parameters

- **<link ID>**: ID of the connection (0~4), for multiple connections.
- **<length>**: data length. Maximum: 2048 bytes.
- **<"remote host">**: IPv4 address or IPv6 address or domain name of remote host, can be set in UDP transmission.
- **<remote port>**: the remote port number.

3.3.8 AT+CIPSENDEX: Send Data in the Normal Transmission Mode in Expanded Ways

Set Command

Function:

Set the data length to be send in *Normal Transmission Mode*, or use `\0` (0x5c, 0x30 ASCII) to trigger data transmission.

Command:

```
// Single connection: (AT+CIPMUX=0)
AT+CIPSENDEX=<length>

// Multiple connections: (AT+CIPMUX=1)
AT+CIPSENDEX=<link ID>,<length>

// Remote host and port can be set for UDP transmission:
AT+CIPSENDEX=[<link ID>,]<length>[,<"remote host">,<remote port>]
```

Response:

```
OK

>
```

This response indicates that AT is ready for receiving data. You should enter the data of designated length. When the data length reaches the `<length>` value, or when the string `\0` appears in the data, the transmission starts.

If the connection cannot be established or gets disconnected during transmission, the system returns:

```
ERROR
```

If the data are successfully transmitted, the system returns:

```
SEND OK
```

Parameters

- **<link ID>**: ID of the connection (0~4), for multiple connections.
- **<length>**: data length. Maximum: 2048 bytes.
- **<"remote host">**: IPv4 address or IPv6 address or domain name of remote host, can be set in UDP transmission.
- **<remote port>**: remote port can be set in UDP transmission.

Notes

- When the requirement of data length is met, or when the string `\0` (0x5c, 0x30 in ASCII) appears, the transmission of data starts. Go back to the normal command mode and wait for the next AT command.
- If the data contains the `\<any>`, it means that drop backslash symbol and only use `<any>` character.
- When sending `\0`, please use a backslash to escape it as `\\0`.

3.3.9 AT+CIPCLOSE: Close TCP/UDP/SSL Connection

Set Command

Function:

Close TCP/UDP/SSL connection in the multiple connections mode.

Command:

```
AT+CIPCLOSE=<link ID>
```

Execute Command

Function:

Close TCP/UDP/SSL connection in the single connection mode.

```
AT+CIPCLOSE
```

Response:

```
OK
```

Parameter

- **<link ID>**: ID of the connection that you want to close. If you set it to 5, all connections will be closed.

3.3.10 AT+CIFSR: Obtain the Local IP Address and MAC Address

Execute Command

Command:

```
AT+CIFSR
```

Response:

```
+CIFSR:APIP,<"APIP">
+CIFSR:APIP6LL,<"APIP6LL">
+CIFSR:APIP6GL,<"APIP6GL">
+CIFSR:APMAC,<"APMAC">
+CIFSR:STAIP,<"STAIP">
+CIFSR:STAIP6LL,<"STAIP6LL">
+CIFSR:STAIP6GL,<"STAIP6GL">
+CIFSR:STAMAC,<"STAMAC">
+CIFSR:ETHIP,<"ETHIP">
+CIFSR:ETHIP6LL,<"ETHIP6LL">
+CIFSR:ETHIP6GL,<"ETHIP6GL">
+CIFSR:ETHMAC,<"ETHMAC">

OK
```

Parameters

- **<"APIP">**: IPv4 address of Wi-Fi softAP interface
- **<"APIP6LL">**: Linklocal IPv6 address of Wi-Fi softAP interface
- **<"APIP6GL">**: Global IPv6 address of Wi-Fi softAP interface
- **<"APMAC">**: MAC address of Wi-Fi softAP interface
- **<"STAIP">**: IPv4 address of Wi-Fi station interface
- **<"STAIP6LL">**: Linklocal IPv6 address of Wi-Fi station interface
- **<"STAIP6GL">**: Global IPv6 address of Wi-Fi station interface
- **<"STAMAC">**: MAC address of Wi-Fi station interface
- **<"ETHIP">**: IPv4 address of ethernet interface
- **<"ETHIP6LL">**: Linklocal IPv6 address of ethernet interface
- **<"ETHIP6GL">**: Global IPv6 address of ethernet interface
- **<"ETHMAC">**: MAC address of ethernet interface

Note

- Only when the ESP device has the valid interface information can you query its IP address and MAC address.

3.3.11 AT+CIPMUX: Enable/disable Multiple Connections

Query Command

Function:

Query the connection type.

Command:

```
AT+CIPMUX?
```

Response:

```
+CIPMUX:<mode>
OK
```

Set Command

Function:

Set the connection type.

Command:

```
AT+CIPMUX=<mode>
```

Response:

```
OK
```

Parameter

- **<mode>**: connection mode. Default: 0.
  - 0: single connection.
  - 1: multiple connections.

Notes

- This mode can only be changed after all connections are disconnected.
- If you want to set the multiple connections mode, ESP devices should be in the *Normal Transmission Mode* (*AT+CIPMODE=0*).
- If you want to set the single connection mode when the TCP/SSL server is running, you should delete the server first. (*AT+CIPSERVER=0*).

Example

```
AT+CIPMUX=1
```

3.3.12 AT+CIPSERVER: Delete/create a TCP/SSL Server

Query Command

Function:

Query the TCP/SSL server status.

Command:

```
AT+CIPSERVER?
```

Response:

```
+CIPSERVER:<mode>[,<port>,<"type">][,<CA enable>]

OK
```

Set Command

Command:

```
AT+CIPSERVER=<mode>[,<param2>][,<"type">][,<CA enable>]
```

Response:

```
OK
```

Parameters

- **<mode>**:
  - 0: delete a server.
  - 1: create a server.
- **<param2>**: It means differently depending on the parameter `<mode>`:
- If `<mode>` is 1, `<param2>` represents the port number. Default: 333.
- If `<mode>` is 0, `<param2>` represents whether the server closes all connections. Default: 0.
  - 0: shutdown the server and keep existing connections.
  - 1: shutdown the server and close all connections.
- **<"type">**: server type: "TCP", "TCPv6", "SSL", or "SSLv6". Default: "TCP". This parameter is **NOT** applicable to ESP8266 platform due to memory limitation.
- **<CA enable>**: not applicable to ESP8266 devices.
  - 0: disable CA.
  - 1: enable CA.

Notes

- A TCP/SSL server can only be created when multiple connections are activated (*AT+CIPMUX=1*).
- A server monitor will be created automatically when the server is created. Only one server can be created at most.
- When a client is connected to the server, it will take up one connection and be assigned an ID.
- If you want to create a TCP/SSL server based on IPv6 network, set *AT+CIPV6=1* first, and obtain an IPv6 address.

Example

```
// To create a TCP server
AT+CIPMUX=1
AT+CIPSERVER=1,80

// To create an SSL server
AT+CIPMUX=1
AT+CIPSERVER=1,443,"SSL",1

// To create an SSL server based on IPv6 network
AT+CIPMUX=1
AT+CIPSERVER=1,443,"SSLv6",0

// To delete an server and close all clients
AT+CIPSERVER=0,1
```

3.3.13 AT+CIPSERVERMAXCONN: Query/Set the Maximum Connections Allowed by a Server

Query Command

Function:

Obtain the maximum number of clients allowed to connect to the TCP/SSL server.

Command:

```
AT+CIPSERVERMAXCONN?
```

Response:

```
+CIPSERVERMAXCONN:<num>
OK
```

Set Command

Function:

Set the maximum number of clients allowed to connect to the TCP/SSL server.

Command:

```
AT+CIPSERVERMAXCONN=<num>
```

Response:

```
OK
```

Parameter

- **<num>**: the maximum number of clients allowed to connect to the TCP/SSL server.

Note

- You should call the command `AT+CIPSERVERMAXCONN=<num>` before creating a server.

Example

```
AT+CIPMUX=1
AT+CIPSERVERMAXCONN=2
AT+CIPSERVER=1,80
```

3.3.14 AT+CIPMODE: Query/Set the Transmission Mode

Query Command

Function:

Query the transmission mode.

Command:

```
AT+CIPMODE?
```

Response:

```
+CIPMODE:<mode>
OK
```

Set Command

Function:

Set the transmission mode.

Command:

```
AT+CIPMODE=<mode>
```

Response:

```
OK
```

Parameter

- **<mode>**:
  - 0: *Normal Transmission Mode*.
  - 1: Wi-Fi *Passthrough Receiving Mode*, or called transparent receiving transmission, which can only be enabled in TCP single connection mode, UDP mode when the remote host and port do not change, or SSL single connection mode.

Notes

- The configuration changes will NOT be saved in flash.

Example

```
AT+CIPMODE=1
```

3.3.15 AT+SAVETRANSLINK: Set Whether to Enter Wi-Fi Passthrough Mode on Power-up

For TCP/SSL Single Connection

Set Command

Command:

```
AT+SAVETRANSLINK=<mode>,<"remote host">,<remote port>[,<"type">,<keep alive>]
```

Response:

```
OK
```

Parameters

- **<mode>**:
  - 0: ESP will NOT enter Wi-Fi *Passthrough Mode* on power-up.
  - 1: ESP will enter Wi-Fi *Passthrough Mode* on power-up.
- **<"remote host">**: string parameter showing the IPv4 address or IPv6 address or domain name of remote host.
- **<remote port>**: the remote port number.
- **<"type">**: string parameter showing the type of transmission: "TCP", "TCPv6", "SSL", or "SSLv6". Default: "TCP".
- **<keep alive>**: TCP keep-alive interval. Default: 0.
  - 0: disable the keep-alive function.
  - 1 ~ 7200: detection interval. Unit: second.

Notes

- This command will save the Wi-Fi *Passthrough Mode* configuration in the NVS area. If `<mode>` is set to 1, ESP device will enter the Wi-Fi *Passthrough Mode* in any subsequent power cycles. The configuration will take effect after ESP reboots.
- As long as the remote host and port are valid, the configuration will be saved in flash.
- If you want to establish TCP/SSL connection based on IPv6 network, set *AT+CIPV6=1* first, and ensure the connected AP by *AT+CWJAP* supports IPv6 and esp-at got the IPv6 address which you can check it by AT+CIPSTA.

Example

```
AT+SAVETRANSLINK=1,"192.168.6.110",1002,"TCP"
AT+SAVETRANSLINK=1,"www.baidu.com",443,"SSL"
AT+SAVETRANSLINK=1,"240e:3a1:2070:11c0:55ce:4e19:9649:b75",8080,"TCPv6"
AT+SAVETRANSLINK=1,"240e:3a1:2070:11c0:55ce:4e19:9649:b75",8080,"SSLv6"
```

For UDP Transmission

Set Command

Command:

```
AT+SAVETRANSLINK=<mode>,<"remote host">,<remote port>,[<"type">,<local port>]
```

Response:

```
OK
```

Parameters

- **<mode>**:
  - 0: ESP will NOT enter Wi-Fi *Passthrough Mode* on power-up.
  - 1: ESP will enter Wi-Fi *Passthrough Mode* on power-up.
- **<"remote host">**: string parameter showing the IPv4 address or IPv6 address or domain name of remote host.
- **<remote port>**: the remote port number.
- **<"type">**: string parameter showing the type of transmission: "UDP" or "UDPv6". Default: "TCP".
- **<local port>**: local port when UDP Wi-Fi passthrough is enabled on power-up.

Notes

- This command will save the Wi-Fi *Passthrough Mode* configuration in the NVS area. If `<mode>` is set to 1, ESP device will enter the Wi-Fi *Passthrough Mode* in any subsequent power cycles. The configuration will take effect after ESP reboots.
- As long as the remote host and port are valid, the configuration will be saved in flash.

- If you want to establish UDP transmission based on IPv6 network, set *AT+CIPV6=1* first, and ensure the connected AP by *AT+CWJAP* supports IPv6 and esp-at got the IPv6 address which you can check it by AT+CIPSTA.

Example

```
AT+SAVETRANSLINK=1,"192.168.6.110",1002,"UDP",1005
AT+SAVETRANSLINK=1,"240e:3a1:2070:11c0:55ce:4e19:9649:b75",8081,"UDPv6",1005
```

3.3.16 AT+CIPSTO: Query/Set the local TCP/SSL Server Timeout

Query Command

Function:

Query the local TCP/SSL server timeout.

Command:

```
AT+CIPSTO?
```

Response:

```
+CIPSTO:<time>
OK
```

Set Command

Function:

Set the local TCP/SSL server timeout.

Command:

```
AT+CIPSTO=<time>
```

Response:

```
OK
```

Parameter

- **<time>**: local TCP/SSL server timeout. Unit: second. Range: [0,7200].

Notes

- When a TCP/SSL client does not communicate with the ESP server within the `<time>` value, the server will terminate this connection.
- If you set `<time>` to 0, the connection will never timeout. This configuration is not recommended.
- When the client initiates a communication with the server within the set time, the timer will restart. After the timeout expires, the client is closed. During the set time, if the server initiate a communication with the client, the timer will not restart. After the timeout expires, the client is closed.

Example

```
AT+CIPMUX=1
AT+CIPSERVER=1,1001
AT+CIPSTO=10
```

3.3.17 AT+CIPSNTPCFG: Query/Set the Time Zone and the SNTP Server

Query Command

Command:

```
AT+CIPSNTPCFG?
```

Response:

```
+CIPSNTPCFG:<enable>,<timezone>,<SNTP server1>[,<SNTP server2>,<SNTP server3>]
OK
```

Set Command

Command:

```
AT+CIPSNTPCFG=<enable>,<timezone>[,<SNTP server1>,<SNTP server2>,<SNTP server3>]
```

Response:

```
OK
```

Parameters

- **<enable>**: configure the SNTP server:
  - 1: the SNTP server is configured.
  - 0: the SNTP server is not configured.
- **<timezone>**: support the following two formats:
  - The first format range is [-12,14]. It marks most of the time zones by offset from Coordinated Universal Time (UTC) in **whole hours** (UTC12:00 to UTC+14:00).
  - The second format is `UTC offset`. The `UTC offset` specifies the time value you must add to the UTC time to get a local time value. It has syntax like `[+|-][hh]mm`. This is negative if the local time zone is on the west of the Prime Meridian and positive if it is on the east. The hour(hh) must be between -12 and 14, and the minute(mm) between 0 and 59. For example, if you want to set the timezone to New Zealand (Chatham Islands) which is in `UTC+12:45`, you should set the parameter `<timezone>` to `1245`. Please refer to UTC offset wiki for more information.
- **[<SNTP server1>]**: the first SNTP server.
- **[<SNTP server2>]**: the second SNTP server.
- **[<SNTP server3>]**: the third SNTP server.

Note

- If the three SNTP servers are not configured, one of the following default servers will be used: "cn.ntp.org.cn", "ntp.sjtu.edu.cn", and "us.pool.ntp.org".
- For the query command, `<timezone>` parameter in the response may be different from the `<timezone>` parameter in set command. Because the `<timezone>` parameter supports the second `UTC offset` format, for example, set `AT+CIPSNTPCFG=1,015`, for query command, ESP-AT ignores the leading zero of the `<timezone>` parameter, and the valid value is `15`. It does not belong to the first format, so it is parsed according to the second `UTC offset` format, that is, `UTC+00:15`, that is, `timezone` is 0 in the response.

Example

```
// Enable SNTP server, set timezone to China (UTC+08:00)
AT+CIPSNTPCFG=1,8,"cn.ntp.org.cn","ntp.sjtu.edu.cn"
or
AT+CIPSNTPCFG=1,800,"cn.ntp.org.cn","ntp.sjtu.edu.cn"

// Enable SNTP server, set timezone to New York of the United States (UTC05:00)
AT+CIPSNTPCFG=1,-5,"0.pool.ntp.org","time.google.com"
or
AT+CIPSNTPCFG=1,-500,"0.pool.ntp.org","time.google.com"

// Enable SNTP server, set timezone to New Zealand (Chatham Islands, UTC+12:45)
AT+CIPSNTPCFG=1,1245,"0.pool.ntp.org","time.google.com"
```

3.3.18 AT+CIPSNTPTIME: Query the SNTP Time

Query Command

Command:

```
AT+CIPSNTPTIME?
```

Response:

```
+CIPSNTPTIME:<asctime style time>
OK
```

Note

- The asctime style time is defined at asctime man page.

Example

```
AT+CIPSNTPCFG=1,8,"cn.ntp.org.cn","ntp.sjtu.edu.cn"

OK

AT+CIPSNTPTIME?
+CIPSNTPTIME:Mon Dec 12 02:33:32 2016
OK
```

3.3.19 AT+CIUPDATE: Upgrade Firmware Through Wi-Fi

ESP-AT upgrades firmware at runtime by downloading the new firmware from a specific server through Wi-Fi and then flash it into some partitions.

Query Command

Function:

Query ESP device upgrade status.

Command:

```
AT+CIUPDATE?
```

Response:

```
+CIPUPDATE:<state>
OK
```

Execute Command

Function:

Upgrade OTA the latest version of firmware via TCP/SSL from the server.

Command:

```
AT+CIUPDATE
```

Response:

```
+CIPUPDATE:<state>
OK
```

or

```
ERROR
```

Set Command

Function:

Upgrade the specified version of firmware from the server.

Command:

```
AT+CIUPDATE=<ota mode>[,<version>][,<firmware name>][,<nonblocking>]
```

Response:

```
+CIPUPDATE:<state>
OK
```

or

```
ERROR
```

Parameters

- **<ota mode>**:
  - 0: OTA via HTTP.
  - 1: OTA via HTTPS. If it does not work, please check whether `./build.py menuconfig` > `Component config` > `AT` > `OTA based upon ssl` is enabled. For more information, please refer to *Build Your Own ESP-AT Project*.
- **<version>**: AT version, such as, `v1.2.0.0`, `v1.1.3.0`, `v1.1.2.0`.
- **<firmware name>**: firmware to upgrade, such as, `ota`, `mqtt_ca`, `client_ca` or other custom partition in `at_customize.csv`.
- **<nonblocking>**:
  - 0: OTA by blocking mode (In this mode, user can not send AT command until OTA completes successfully or fails.)
  - 1: OTA by non-blocking mode (Users need to manually restart after upgrade done (+CIPUPDATE:4).)
- **<state>**:
  - 0: Idle.
  - 1: Server found.
  - 2: Connected to the server.
  - 3: Got the upgrade version.
  - 4: Upgrade done.
  - -1: Upgrade failed.

Notes

- The speed of the upgrade depends on the network status.
- If the upgrade fails due to unfavorable network conditions, AT will return `ERROR`. Please wait for some time before retrying.
- If you use Espressif's AT BIN, `AT+CIUPDATE` will download a new AT BIN from the Espressif Cloud.
- If you use a user-compiled AT BIN, you need to implement your own AT+CIUPDATE FOTA function. ESP-AT project provides an example of FOTA.
- After you upgrade the AT firmware, you are suggested to call the command *AT+RESTORE* to restore the factory default settings.
- Upgraded to an older version is not recommended.

Example

```
AT+CIUPDATE
AT+CIUPDATE=1
AT+CIUPDATE=1,"v1.2.0.0"
AT+CIUPDATE=1,"v2.2.0.0","mqtt_ca"
AT+CIUPDATE=1,"V2.2.0.0","ota",1
AT+CIUPDATE=1,,,1
AT+CIUPDATE=1,,"ota",1
AT+CIUPDATE=1,"V2.2.0.0",,1
```

3.3.20 AT+CIPDINFO: Set "+IPD" Message Mode

Set Command

Command:

```
AT+CIPDINFO=<mode>
```

Response:

```
OK
```

Parameters

- **<mode>**:
  - 0: does not show the remote host and port in "+IPD" and "+CIPRECVDATA" messages.
  - 1: show the remote host and port in "+IPD" and "+CIPRECVDATA" messages.

Example

```
AT+CIPDINFO=1
```

3.3.21 AT+CIPSSLCCONF: Query/Set SSL Clients

Query Command

Function:

Query the configuration of each connection where the ESP device runs as an SSL client.

Command:

```
AT+CIPSSLCCONF?
```

Response:

```
+CIPSSLCCONF:<link ID>,<auth_mode>,<pki_number>,<ca_number>
OK
```

Set Command

Command:

```
// Single connection: (AT+CIPMUX=0)
AT+CIPSSLCCONF=<auth_mode>[,<pki_number>][,<ca_number>]

// Multiple connections: (AT+CIPMUX=1)
AT+CIPSSLCCONF=<link ID>,<auth_mode>[,<pki_number>][,<ca_number>]
```

Response:

```
OK
```

Parameters

- **<link ID>**: ID of the connection (0 ~ max). For multiple connections, if the value is max, it means all connections. By default, max is 5.
- **<auth_mode>**:
  - 0: no authentication. In this case `<pki_number>` and `<ca_number>` are not required.
  - 1: the client provides the client certificate for the server to verify.
  - 2: the client loads CA certificate to verify the server's certificate.
  - 3: mutual authentication.
- **<pki_number>**: the index of certificate and private key. If there is only one certificate and private key, the value should be 0.
- **<ca_number>**: the index of CA. If there is only one CA, the value should be 0.

Notes

- If you want this configuration to take effect immediately, run this command before establishing an SSL connection.
- The configuration changes will be saved in the NVS area. If you set the command *AT+SAVETRANSLINK* to enter SSL Wi-Fi *Passthrough Mode* on power-up, the ESP device will establish an SSL connection based on this configuration when powered up next time.

3.3.22 AT+CIPSSLCCN: Query/Set the Common Name of the SSL Client

Query Command

Function:

Query the common name of the SSL client of each connection.

Command:

```
AT+CIPSSLCCN?
```

Response:

```
+CIPSSLCCN:<link ID>,<"common name">
OK
```

Set Command

Command:

```
// Single connection: (AT+CIPMUX=0)
AT+CIPSSLCCN=<"common name">

// Multiple connections: (AT+CIPMUX=1)
AT+CIPSSLCCN=<link ID>,<"common name">
```

Response:

```
OK
```

Parameters

- **<link ID>**: ID of the connection (0 ~ max). For the single connection, the link ID is 0. For multiple connections, if the value is max, it means all connections. Max is 5 by default.
- **<"common name">**: this parameter is used to verify the Common Name in the certificate sent by the server. The maximum length of `common name` is 64 bytes.

Note

- If you want this configuration to take effect immediately, run this command before establishing the SSL connection.

3.3.23 AT+CIPSSLCSNI: Query/Set SSL Client Server Name Indication (SNI)

Query Command

Function:

Query the SNI configuration of each connection.

Command:

```
AT+CIPSSLCSNI?
```

Response:

```
+CIPSSLCSNI:<link ID>,<"sni">
OK
```

Set Command

Command:

```
Single connection: (AT+CIPMUX=0)
AT+CIPSSLCSNI=<"sni">

Multiple connections: (AT+CIPMUX=1)
AT+CIPSSLCSNI=<link ID>,<"sni">
```

Response:

```
OK
```

Parameters

- **<link ID>**: ID of the connection (0 ~ max). For the single connection, the link ID is 0. For multiple connections, if the value is max, it means all connections. Max is 5 by default.
- **<"sni">**: the Server Name Indication in ClientHello. The maximum length of `sni` is 64 bytes.

Notes

- If you want this configuration to take effect immediately, run this command before establishing the SSL connection.

3.3.24 AT+CIPSSLCALPN: Query/Set SSL Client Application Layer Protocol Negotiation (ALPN)

Query Command

Function:

Query the ALPN configuration of each connection where the ESP device runs as an SSL client.

Command:

```
AT+CIPSSLCALPN?
```

Response:

```
+CIPSSLCALPN:<link ID>[,<"alpn">][,<"alpn">][,<"alpn">]

OK
```

Set Command

Command:

```
// Single connection: (AT+CIPMUX=0)
AT+CIPSSLCALPN=<counts>[,<"alpn">][,<"alpn">][,<"alpn">]

// Multiple connections: (AT+CIPMUX=1)
AT+CIPSSLCALPN=<link ID>,<counts>[,<"alpn">][,<"alpn">][,<"alpn">]
```

Response:

```
OK
```

Parameters

- **<link ID>**: ID of the connection (0 ~ max). For the single connection, the link ID is 0. For multiple connections, if the value is max, it means all connections. Max is 5 by default.
- **<counts>**: the number of ALPNs. Range: [0,5].
- 0: clean the ALPN configuration.
- [1,5]: set the ALPN configuration.
- **<"alpn">**: a string paramemter showing the ALPN in ClientHello. The maximum length of alpn is limited by the command length.

Note

- If you want this configuration to take effect immediately, run this command before establishing the SSL connection.

3.3.25 AT+CIPSSLCPSK: Query/Set SSL Client Pre-shared Key (PSK)

Query Command

Function:

Query the PSK configuration of each connection where the ESP device runs as an SSL client.

Command:

```
AT+CIPSSLCPSK?
```

Response:

```
+CIPSSLCPSK:<link ID>,<"psk">,<"hint">
OK
```

Set Command

Command:

```
// Single connection: (AT+CIPMUX=0)
AT+CIPSSLCPSK=<"psk">,<"hint">

// Multiple connections: (AT+CIPMUX=1)
AT+CIPSSLCPSK=<link ID>,<"psk">,<"hint">
```

Response:

```
OK
```

Parameters

- **<link ID>**: ID of the connection (0 ~ max). For single connection, <link ID> is 0. For multiple connections, if the value is max, it means all connections, max is 5 by default.
- **<"psk">**: PSK identity. Maximum length: 32.
- **<"hint">**: PSK hint. Maximum length: 32.

Notes

- If you want this configuration to take effect immediately, run this command before establishing the SSL connection.

3.3.26 AT+CIPRECONNINTV: Query/Set the TCP/UDP/SSL reconnection Interval for the Wi-Fi Passthrough Mode

Query Command

Function:

Query the automatic connect interval for the Wi-Fi *Passthrough Mode*.

Command:

```
AT+CIPRECONNINTV?
```

Response:

```
+CIPRECONNINTV:<interval>
OK
```

Set Command

Function:

Set the automatic reconnecting interval when TCP/UDP/SSL transmission breaks in the Wi-Fi *Passthrough Mode*.

Command:

```
AT+CIPRECONNINTV=<interval>
```

Response:

```
OK
```

Parameter

- **<interval>**: the duration between automatic reconnections. Unit: 100 milliseconds. Default: 1. Range: [1,36000].

Note

- The configuration changes will be saved in the NVS area if *AT+SYSSTORE=1*.

Example

```
AT+CIPRECONNINTV=10
```

3.3.27 AT+CIPRECVMODE: Query/Set Socket Receiving Mode

Query Command

Function:

Query the socket receiving mode.

Command:

```
AT+CIPRECVMODE?
```

Response:

```
+CIPRECVMODE:<mode>
OK
```

Set Command

Command:

```
AT+CIPRECVMODE=<mode>
```

Response:

```
OK
```

Parameter

- **<mode>**: the receive mode of socket data. Default: 0.
  - 0: active mode. ESP-AT will send all the received socket data instantly to the host MCU with header "+IPD".
  - 1: passive mode. ESP-AT will keep the received socket data in an internal buffer (socket receive window, 5760 bytes by default for ESP8266 devices, 5744 bytes by default for non ESP8266 devices), and wait for the host MCU to read. If the buffer is full, the socket transmission will be blocked for TCP/SSL connections, or data will be lost for UDP connections.

Notes

- The configuration can not be used in the Wi-Fi *Passthrough Mode*. If it is a UDP transmission in passive mode, data will be lost when the buffer is full.
- When ESP-AT receives socket data in passive mode, it will prompt the following messages in different scenarios:
  - For multiple connections mode (AT+CIPMUX=1), the message is `+IPD,<link ID>,<len>`.
  - For single connection mode (AT+CIPMUX=0), the message is `+IPD,<len>`.
- `<len>` is the total length of socket data in the buffer.
- You should read data by running *AT+CIPRECVDATA* once there is a `+IPD` reported. Otherwise, the next `+IPD` will not be reported to the host MCU until the previous `+IPD` has been read.
- In case of disconnection, the buffered socket data will still be there and can be read by the MCU until you send *AT+CIPCLOSE*. In other words, if `+IPD` has been reported, the message `CLOSED` of this connection will never come until you send *AT+CIPCLOSE* or read all data by command *AT+CIPRECVDATA*.

Example

```
AT+CIPRECVMODE=1
```

3.3.28 AT+CIPRECVDATA: Obtain Socket Data in Passive Receiving Mode

Set Command

Command:

```
// Single connection: (AT+CIPMUX=0)
AT+CIPRECVDATA=<len>

// Multiple connections: (AT+CIPMUX=1)
AT+CIPRECVDATA=<link_id>,<len>
```

Response:

```
+CIPRECVDATA:<actual_len>,<data>
OK
```

or

```
+CIPRECVDATA:<actual_len>,<"remote IP">,<remote port>,<data>
OK
```

Parameters

- **<link_id>**: connection ID in multiple connections mode.
- **<len>**: the max value is 0x7fffffff. If the actual length of the received data is less than `len`, the actual length will be returned.
- **<actual_len>**: length of the data you actually obtain.
- **<data>**: the data you want to obtain.

- **<"remote IP">**: string parameter showing the remote IPv4 address or IPv6 address, enabled by the command *AT+CIPDINFO=1*.
- **<remote port>**: the remote port number, enabled by the command *AT+CIPDINFO=1*.

Example

```
AT+CIPRECVMODE=1

// For example, if host MCU gets a message of receiving 100-byte data in connection␣
↪with No.0,
// the message will be "+IPD,0,100".
// Then you can read those 100-byte data by using the command below.
AT+CIPRECVDATA=0,100
```

3.3.29 AT+CIPRECVLEN: Obtain Socket Data Length in Passive Receiving Mode

Query Command

Function:

Query the length of the entire data buffered for the connection.

Command:

```
AT+CIPRECVLEN?
```

Response:

```
+CIPRECVLEN:<data length of link0>,<data length of link1>,<data length of link2>,
↪<data length of link3>,<data length of link4>
OK
```

Parameters

- **<data length of link>**: length of the entire data buffered for the connection.

Note

- For SSL connections, ESP-AT will return the length of the encrypted data, so the returned length will be larger than the real data length.

Example

```
AT+CIPRECVLEN?
+CIPRECVLEN:100,,,,,
OK
```

3.3.30 AT+PING: Ping the Remote Host

Set Command

Function:

Ping the remote host.

Command:

```
AT+PING=<"host">
```

Response:

```
+PING:<time>

OK
```

or

```
+PING:TIMEOUT   // esp-at returns this response only when the domain name resolution␣
↪failure or ping timeout

ERROR
```

Parameters

- **<"host">**: string parameter showing the host IPv4 address or IPv6 address or domain name.
- **<time>**: the response time of ping. Unit: millisecond.

Notes

- If you want to ping a remote host based on IPv6 network, set *AT+CIPV6=1* first, and ensure the connected AP by *AT+CWJAP* supports IPv6 and esp-at got the IPv6 address which you can check it by AT+CIPSTA.
- If the remote host is a domain name string, ping will first resolve the domain name (IPv4 address preferred) from DNS (domain name server), and then ping the remote IP address.

Example

```
AT+PING="192.168.1.1"
AT+PING="www.baidu.com"

// China Future Internet Engineering Center
AT+PING="240c::6666"
```

3.3.31 AT+CIPDNS: Query/Set DNS Server Information

Query Command

Function:

Query the current DNS server information.

Command:

```
AT+CIPDNS?
```

Response:

```
+CIPDNS:<enable>[,<"DNS IP1">,<"DNS IP2">,<"DNS IP3">]
OK
```

Set Command

Function:

Set DNS server information.

Command:

```
AT+CIPDNS=<enable>[,<"DNS IP1">,<"DNS IP2">,<"DNS IP3">]
```

Response:

```
OK
```

or

```
ERROR
```

Parameters

- **<enable>**: configure DNS settings
  - 0: Enable automatic DNS settings from DHCP. The DNS will be restored to `208.67.222.222`. Only when DHCP is updated will it take effect.
  - 1: Enable manual DNS settings. If you do not set a value for `<DNS IPx>`, it will use `208.67.222.222` by default.
- **<"DNS IP1">**: the first DNS IP. For the set command, this parameter only works when you set <enable> to 1, i.e. enable manual DNS settings. If you set <enable> to 1 and a value for this parameter, the ESP-AT will return this parameter as the current DNS setting when you run the query command.
- **<"DNS IP2">**: the second DNS IP. For the set command, this parameter only works when you set <enable> to 1, i.e. enable manual DNS settings. If you set <enable> to 1 and a value for this parameter, the ESP-AT will return this parameter as the current DNS setting when you run the query command.
- **<"DNS IP3">**: the third DNS IP. For the set command, this parameter only works when you set <enable> to 1, i.e. enable manual DNS settings. If you set <enable> to 1 and a value for this parameter, the ESP-AT will return this parameter as the current DNS setting when you run the query command.

Notes

- The configuration changes will be saved in the NVS area if *AT+SYSSTORE=1*.
- The three parameters cannot be set to the same server.
- When `<enable>` is set to 1, the DNS server may change according to the configuration of the router which the ESP device is connected to.

Example

```
AT+CIPDNS=0
AT+CIPDNS=1,"208.67.222.222","114.114.114.114","8.8.8.8"

// first DNS Server based on IPv6: China Future Internet Engineering Center
// second DNS Server based on IPv6: google-public-dns-a.google.com
// third DNS Server based on IPv6: main DNS Server based on IPv6 in JiangSu Province,␣
↪China
AT+CIPDNS=1,"240c::6666","2001:4860:4860::8888","240e:5a::6666"
```

3.3.32 AT+CIPTCPOPT: Query/Set the Socket Options

Query Command

Function:

Query current socket options.

Command:

```
AT+CIPTCPOPT?
```

Response:

```
+CIPTCPOPT:<link_id>,<so_linger>,<tcp_nodelay>,<so_sndtimeo>
OK
```

Set Command

Command:

```
// Single TCP connection (AT+CIPMUX=0):
AT+CIPTCPOPT=[<so_linger>],[<tcp_nodelay>],[<so_sndtimeo>]

// Multiple TCP Connections (AT+CIPMUX=1):
AT+CIPTCPOPT=<link ID>,[<so_linger>],[<tcp_nodelay>],[<so_sndtimeo>]
```

Response:

```
OK
```

or

```
ERROR
```

Parameters

- **<link_id>**: ID of the connection (0 ~ max). For multiple connections, if the value is max, it means all connections. By default, max is 5.
- **<so_linger>**: configure the `SO_LINGER` options for the socket. Unit: second. Default: -1.
  - = -1: off

- = 0: on, linger time = 0
- > 0: on, linger time = <so_linger>

- **<tcp_nodelay>**: configure the `TCP_NODELAY` option for the socket. Default: 0.
  - 0: disable TCP_NODELAY
  - 1: enable TCP_NODELAY
- **<so_sndtimeo>**: configure the `SO_SNDTIMEO` option for socket. Unit: millisecond. Default: 0.

3.4 [ESP32 Only] Bluetooth® Low Energy AT Commands

[]

ESP32 AT firmware supports Bluetooth® Core Specification Version 5.0.

- [ESP32 Only] *AT+BLEINIT*: Bluetooth LE initialization.
- [ESP32 Only] *AT+BLEADDR*: Query/Set Bluetooth LE device address.
- [ESP32 Only] *AT+BLENAME*: Query/Set Bluetooth LE device name.
- [ESP32 Only] *AT+BLESCANPARAM*: Query/Set parameters of Bluetooth LE scanning.
- [ESP32 Only] *AT+BLESCAN*: Enable Bluetooth LE scanning.
- [ESP32 Only] *AT+BLESCANRSPDATA*: Set Bluetooth LE scan response.
- [ESP32 Only] *AT+BLEADVPARAM*: Query/Set parameters of Bluetooth LE advertising.
- [ESP32 Only] *AT+BLEADVDATA*: Set Bluetooth LE advertising data.
- [ESP32 Only] *AT+BLEADVDATAEX*: Automatically set Bluetooth LE advertising data.
- [ESP32 Only] *AT+BLEADVSTART*: Start Bluetooth LE advertising.
- [ESP32 Only] *AT+BLEADVSTOP*: Stop Bluetooth LE advertising.
- [ESP32 Only] *AT+BLECONN*: Establish Bluetooth LE connection.
- [ESP32 Only] *AT+BLECONNPARAM*: Query/Update parameters of Bluetooth LE connection.
- [ESP32 Only] *AT+BLEDISCONN*: End Bluetooth LE connection.
- [ESP32 Only] *AT+BLEDATALEN*: Set Bluetooth LE data packet length.
- [ESP32 Only] *AT+BLECFGMTU*: Set Bluetooth LE MTU length.
- [ESP32 Only] *AT+BLEGATTSSRVCRE*: Generic Attributes Server (GATTS) creates services.
- [ESP32 Only] *AT+BLEGATTSSRVSTART*: GATTS starts services.
- [ESP32 Only] *AT+BLEGATTSSRVSTOP*: GATTS Stops Services.
- [ESP32 Only] *AT+BLEGATTSSRV*: GATTS discovers services.
- [ESP32 Only] *AT+BLEGATTSCHAR*: GATTS discovers characteristics.
- [ESP32 Only] *AT+BLEGATTSNTFY*: Notify a client of the value of a characteristic value from the server.
- [ESP32 Only] *AT+BLEGATTSIND*: Indicate the characteristic value from the server to a client.
- [ESP32 Only] *AT+BLEGATTSSETATTR*: GATTS sets characteristics.
- [ESP32 Only] *AT+BLEGATTCPRIMSRV*: Generic Attributes Client (GATTC) discovers primary services.

- [ESP32 Only] *AT+BLEGATTCINCLSRV*: GATTC discovers included services.
- [ESP32 Only] *AT+BLEGATTCCHAR*: GATTC discovers characteristics.
- [ESP32 Only] *AT+BLEGATTCRD*: GATTC reads characteristics.
- [ESP32 Only] *AT+BLEGATTCWR*: GATTC writes characteristics.
- [ESP32 Only] *AT+BLESPPCFG*: Query/Set Bluetooth LE SPP parameters.
- [ESP32 Only] *AT+BLESPP*: Enter Bluetooth LE SPP mode.
- [ESP32 Only] *AT+BLESECPARAM*: Query/Set Bluetooth LE encryption parameters.
- [ESP32 Only] *AT+BLEENC*: Initiate Bluetooth LE encryption request.
- [ESP32 Only] *AT+BLEENCRSP*: Respond to the pairing request from the peer device.
- [ESP32 Only] *AT+BLEKEYREPLY*: Reply the key value to the peer device in the legacy connection stage.
- [ESP32 Only] *AT+BLECONFREPLY*: Reply the confirm value to the peer device in the legacy connection stage.
- [ESP32 Only] *AT+BLEENCDEV*: Query bonded Bluetooth LE encryption device list.
- [ESP32 Only] *AT+BLEENCCLEAR*: Clear Bluetooth LE encryption device list.
- [ESP32 Only] *AT+BLESETKEY*: Set Bluetooth LE static pair key.
- [ESP32 Only] *AT+BLEHIDINIT*: Bluetooth LE Human Interface Device (HID) profile initialization.
- [ESP32 Only] *AT+BLEHIDKB*: Send Bluetooth LE HID keyboard information.
- [ESP32 Only] *AT+BLEHIDMUS*: Send Bluetooth LE HID mouse information.
- [ESP32 Only] *AT+BLEHIDCONSUMER*: Send Bluetooth LE HID consumer information.
- [ESP32 Only] *AT+BLUFI*: Start or Stop BluFi.
- [ESP32 Only] *AT+BLUFINAME*: Query/Set BluFi device name.

3.4.1 [ESP32 Only] AT+BLEINIT: Bluetooth LE Initialization

Query Command

Function:

Check the initialization status of Bluetooth LE.

Command:

```
AT+BLEINIT?
```

Response:

If Bluetooth LE is initialized, AT will return:

```
+BLEINIT:<role>
OK
```

If Bluetooth LE is not initialized, AT will return:

```
+BLEINIT:0
OK
```

Set Command

Function:

Initialize the role of Bluetooth LE.

Command:

```
AT+BLEINIT=<init>
```

Response:

```
OK
```

Parameter

- **<init>**:
  - 0: deinit Bluetooth LE
  - 1: client role
  - 2: server role

Notes

- The file "at_customize.bin" has to be downloaded, so that the relevant commands can be used. Please refer to *How To Customize ble services* for more details.
- Before using other Bluetooth LE AT commands, you should run this command first to trigger the initialization process.
- After the initialization, the Bluetooth LE role cannot be changed unless you run *AT+RST* to restart the system first and then re-initialize the Bluetooth LE role.
- If you use an ESP device as a Bluetooth LE server, a service bin should be downloaded into flash.
  - To learn how to generate a service bin, please refer to esp-at/tools/readme.md.
  - The download address of the service bin is the "ble_data" address in esp-at/module_config/module_${platform}_default/at_customize.csv.

Example

```
AT+BLEINIT=1
```

3.4.2 [ESP32 Only] AT+BLEADDR: Query/Set Bluetooth LE Device Address

Query Command

Function:

Query the Bluetooth LE Public Address.

Command:

```
AT+BLEADDR?
```

Response:

```
+BLEADDR:<BLE_public_addr>
OK
```

Set Command

Function:

Set the Bluetooth LE address type.

Command:

```
AT+BLEADDR=<addr_type>[,<random_addr>]
```

Response:

```
OK
```

Parameter

- **<addr_type>**:
  - 0: Public Address
  - 1: Random Address

Note

- A Static Address should meet the following requirements:
  - The two most significant bits of the address should be equal to 1.
  - At least one bit of the random part of the address should be 0.
  - At least one bit of the random part of the address should be 1.

Example

```
AT+BLEADDR=1,"f8:7f:24:87:1c:7b"    // Set Random Device Address, Static Address
AT+BLEADDR=1                        // Set Random Device Address, Private Address
AT+BLEADDR=0                        // Set Public Device Address
```

3.4.3 [ESP32 Only] AT+BLENAME: Query/Set Bluetooth LE Device Name

Query Command

Function:

Query the Bluetooth LE device name.

Command:

```
AT+BLENAME?
```

Response:

```
+BLENAME:<device_name>
OK
```

Set Command

Function:

Set the Bluetooth LE device name.

Command:

```
AT+BLENAME=<device_name>
```

Response:

```
OK
```

Parameter

- **<device_name>**: the Bluetooth LE device name. The maximum length is 32. Default: "BLE_AT".

Note

- The configuration changes will be saved in the NVS area if *AT+SYSSTORE=1*.

Example

```
AT+BLENAME="esp_demo"
```

3.4.4 [ESP32 Only] AT+BLESCANPARAM: Query/Set Parameters of Bluetooth LE Scanning

Query Command

Function:

Query the parameters of Bluetooth LE scanning.

Command:

```
AT+BLESCANPARAM?
```

Response:

```
+BLESCANPARAM:<scan_type>,<own_addr_type>,<filter_policy>,<scan_interval>,<scan_
↪window>
OK
```

Set Command

Function:

Set the parameters of Bluetooth LE scanning.

Command:

```
AT+BLESCANPARAM=<scan_type>,<own_addr_type>,<filter_policy>,<scan_interval>,<scan_
↪window>
```

Response:

```
OK
```

Parameters

- **<scan_type>**:
  - 0: passive scan
  - 1: active scan
- **<own_addr_type>**:
  - 0: Public Address
  - 1: Random Address
  - 2: RPA Public Address
  - 3: RPA Random Address
- **<filter_policy>**:
  - 0: BLE_SCAN_FILTER_ALLOW_ALL
  - 1: BLE_SCAN_FILTER_ALLOW_ONLY_WLST
  - 2: BLE_SCAN_FILTER_ALLOW_UND_RPA_DIR
  - 3: BLE_SCAN_FILTER_ALLOW_WLIST_PRA_DIR
- **<scan_interval>**: scan interval
- **<scan_window>**: scan window

Note

- The parameter `<scan_window>` CANNOT be larger than `<scan_interval>`.

Example

```
AT+BLEINIT=1   // role: client
AT+BLESCANPARAM=0,0,0,100,50
```

3.4.5 [ESP32 Only] AT+BLESCAN: Enable Bluetooth LE Scanning

Set Command

Function:

Enable/disable scanning.

Command:

```
AT+BLESCAN=<enable>[,<interval>][,<filter_type>,<filter_param>]
```

Response:

```
+BLESCAN:<addr>,<rssi>,<adv_data>,<scan_rsp_data>,<addr_type>
OK
```

Parameters

- **<enable>**:
  - 1: enable continuous scanning.
  - 0: disable continuous scanning.
- **[<interval>]**: optional parameter. Unit: second.
  - If you want to disable the scanning, this parameter should be omitted.
  - If you want to enable the scanning, set a value for this parameter:
  - When you set it to 0, it means that scanning is continuous.
  - When set it to a value other than 0, for example, `AT+BLESCAN=1,3`, it means that scanning will last for 3 seconds and then stop automatically. The scanning results will be returned.
- **[<filter_type>]**: filtering option.
  - 1: "MAC".
  - 2: "NAME".
- **<filter_param>**: filtering parameter showing the remote device MAC address or remote device name.
- **<addr>**: Bluetooth LE address.
- **<rssi>**: signal strength.
- **<adv_data>**: advertising data.
- **<scan_rsp_data>**: scan response data.
- **<addr_type>**: the address type of broadcasters.

Notes

- The response `OK` does not necessarily come before the response `+BLESCAN:<addr>,<rssi>,<adv_data>,<scan_rsp_data>,<addr_type>`. It may be output before `+BLESCAN:<addr>,<rssi>,<adv_data>,<scan_rsp_data>,<addr_type>` or after it.

Example

```
AT+BLEINIT=1    // role: client
AT+BLESCAN=1    // start scanning
AT+BLESCAN=0    // stop scanning
AT+BLESCAN=1,3,1,"24:0A:C4:96:E6:88"  // start scanning, filter type is MAC address
AT+BLESCAN=1,3,2,"ESP-AT"  // start scanning, filter type is device name
```

3.4.6 [ESP32 Only] AT+BLESCANRSPDATA: Set Bluetooth LE Scan Response

Set Command

Function:

Set scan response.

Command:

```
AT+BLESCANRSPDATA=<scan_rsp_data>
```

Response:

```
OK
```

Parameter

- **<scan_rsp_data>**: scan response data is a HEX string. For example, if you want to set the response data to "0x11 0x22 0x33 0x44 0x55", the command should be `AT+BLESCANRSPDATA="1122334455"`.

Example

```
AT+BLEINIT=2   // role: server
AT+BLESCANRSPDATA="1122334455"
```

3.4.7 [ESP32 Only] AT+BLEADVPARAM: Query/Set Parameters of Bluetooth LE Advertising

Query Command

Function:

Query the parameters of advertising.

Command:

```
AT+BLEADVPARAM?
```

Response:

```
+BLEADVPARAM:<adv_int_min>,<adv_int_max>,<adv_type>,<own_addr_type>,<channel_map>,
↪<adv_filter_policy>,<peer_addr_type>,<peer_addr>
OK
```

Set Command

Function:

Set the parameters of advertising.

Command:

```
AT+BLEADVPARAM=<adv_int_min>,<adv_int_max>, <adv_type>,<own_addr_type>,<channel_map>[,
↪<adv_filter_policy>][,<peer_addr_type>] [,<peer_addr>]
```

Response:

```
OK
```

Parameters

- **<adv_int_min>**: minimum advertising interval. It should be less than the value of `<adv_int_max>`. Range: 0x0020 ~ 0x4000.
- **<adv_int_max>**: maximum advertising interval. It should be more than the value of `<adv_int_min>`. Range: 0x0020 ~ 0x4000.
- **<adv_type>**:
  - 0: ADV_TYPE_IND
  - 2: ADV_TYPE_SCAN_IND
  - 3: ADV_TYPE_NONCONN_IND
- **<own_addr_type>**: own Bluetooth LE address type.
  - 0: BLE_ADDR_TYPE_PUBLIC
  - 1: BLE_ADDR_TYPE_RANDOM
- **<channel_map>**: channel of advertising.
  - 1: ADV_CHNL_37
  - 2: ADV_CHNL_38
  - 4: ADV_CHNL_39
  - 7: ADV_CHNL_ALL
- **[<adv_filter_policy>]**: filter policy of advertising.
  - 0: ADV_FILTER_ALLOW_SCAN_ANY_CON_ANY
  - 1: ADV_FILTER_ALLOW_SCAN_WLST_CON_ANY
  - 2: ADV_FILTER_ALLOW_SCAN_ANY_CON_WLST
  - 3: ADV_FILTER_ALLOW_SCAN_WLST_CON_WLST
- **[<peer_addr_type>]**: remote Bluetooth LE address type.
  - 0: PUBLIC
  - 1: RANDOM
- **[<peer_addr>]**: remote Bluetooth LE address.

Example

```
AT+BLEINIT=2   // role: server
AT+BLEADVPARAM=50,50,0,0,4,0,0,"12:34:45:78:66:88"
```

3.4.8 [ESP32 Only] AT+BLEADVDATA: Set Bluetooth LE Advertising Data

Set Command

Function:

Set advertising data.

Command:

```
AT+BLEADVDATA=<adv_data>
```

Response:

```
OK
```

Parameter

- **<adv_data>**: advertising data in HEX string. For example, to set the advertising data to "0x11 0x22 0x33 0x44 0x55", the command should be `AT+BLEADVDATA="1122334455"`.

Note

- If advertising data is preset by command *AT+BLEADVDATAEX*=<dev_name>,<uuid>,<manufacturer_data>,<include_power>, it will be overwritten by this command.

Example

```
AT+BLEINIT=2   // role: server
AT+BLEADVDATA="1122334455"
```

3.4.9 [ESP32 Only] AT+BLEADVDATAEX: Automatically Set Bluetooth LE Advertising Data

Query Command

Function:

Query the parameters of advertising data.

Command:

```
AT+BLEADVDATAEX?
```

Response:

```
+BLEADVDATAEX:<dev_name>,<uuid>,<manufacturer_data>,<include_power>

OK
```

Set Command

Function:

Set the advertising data and start advertising.

Command:

```
AT+BLEADVDATAEX=<dev_name>,<uuid>,<manufacturer_data>,<include_power>
```

Response:

```
OK
```

Parameters

- **<dev_name>**: string parameter showing a device name. For example, if you want to set the device name to "just-test", the command should be `AT+BLEADVSTARTEX="just-test",<uuid>,<manufacturer_data>,<include_power>`.
- **<uuid>**: string parameter. For example, if you want to set the UUID to "0xA002", the command should be `AT+BLEADVSTARTEX=<dev_name>,"A002",<manufacturer_data>,<include_power>`.
- **<manufacturer_data>**: manufacturer data in HEX string. For example, if you set the manufacturer data to "0x11 0x22 0x33 0x44 0x55", the command should be `AT+BLEADVSTARTEX=<dev_name>,<uuid>,"1122334455",<include_power>`.
- **<include_power>**: If you need to include the TX power in the advertising data, you should set the parameter to `1`. Otherwise, set it to `0`.

Note

- If advertising data is preset by command *AT+BLEADVDATA*=<adv_data>, it will be overwritten by this command.

Example

```
AT+BLEINIT=2   // role: server
AT+BLEADVDATAEX="ESP-AT","A002","0102030405",1
```

3.4.10 [ESP32 Only] AT+BLEADVSTART: Start Bluetooth LE Advertising

Execute Command

Function:

Start advertising.

Command:

```
AT+BLEADVSTART
```

Response:

```
OK
```

Notes

- If advertising parameters are NOT set by command *AT+BLEADVPARAM*=<adv_parameter>, the default parameters will be used.
- If advertising data is NOT set by command *AT+BLEADVDATA*=<adv_data>, the advertising playload will be empty.
- If advertising data is preset by command *AT+BLEADVDATA*=<adv_data>, it will be overwritten by *AT+BLEADVDATAEX*=<dev_name>,<uuid>,<manufacturer_data>,<include_power> and vice versa.

Example

```
AT+BLEINIT=2    // role: server
AT+BLEADVSTART
```

3.4.11 [ESP32 Only] AT+BLEADVSTOP: Stop Bluetooth LE Advertising

Execute Command

Function:

Stop advertising.

Command:

```
AT+BLEADVSTOP
```

Response:

```
OK
```

Note

- After the start of advertising, if the Bluetooth LE connection is established successfully, it will stop advertising automatically. In such a case, this command does NOT need to be called.

Example

```
AT+BLEINIT=2    // role: server
AT+BLEADVSTART
AT+BLEADVSTOP
```

3.4.12 [ESP32 Only] AT+BLECONN: Establish Bluetooth LE Connection

Query Command

Function:

Query the Bluetooth LE connection.

Command:

```
AT+BLECONN?
```

Response:

```
+BLECONN:<conn_index>,<remote_address>
OK
```

If the connection has not been established, there will be no <conn_index> and <remote_address> in the response.

Set Command

Function:

Establish the Bluetooth LE connection.

Command:

```
AT+BLECONN=<conn_index>,<remote_address>[,<addr_type>,<timeout>]
```

Response:

```
OK
```

If the connection is established successfully, it will prompt:

```
+BLECONN:<conn_index>,<remote_address>
```

Otherwise, it will prompt:

```
+BLECONN:<conn_index>,-1
```

Parameters

- **<conn_index>**: index of Bluetooth LE connection. Range: [0,2].
- **<remote_address>**: remote Bluetooth LE address.
- **[<addr_type>]**: the address type of broadcasters.
- **[<timeout>]**: the timeout for the connection command. Unit: second. Range: [3,30].

Notes

- It is recommended to scan devices by running *AT+BLESCAN* before initiating a new connection to ensure that the target device is in the broadcast state.
- The maximum timeout for connection is 30 seconds.

- If the Bluetooth LE server is initialized and the connection is established successfully, you can use this command to discover the services in the peer device (GATTC). The following GATTC commands can also be used:
  - *AT+BLEGATTCPRIMSRV*
  - *AT+BLEGATTCINCLSRV*
  - *AT+BLEGATTCCHAR*
  - *AT+BLEGATTCRD*
  - *AT+BLEGATTCWR*
  - *AT+BLEGATTSIND*

Example

```
AT+BLEINIT=1   // role: client
AT+BLECONN=0,"24:0a:c4:09:34:23",0,10
```

3.4.13 [ESP32 Only] AT+BLECONNPARAM: Query/Update Parameters of Bluetooth LE Connection

Query Command

Function:

Query the parameters of Bluetooth LE connection.

Command:

```
AT+BLECONNPARAM?
```

Response:

```
+BLECONNPARAM:<conn_index>,<min_interval>,<max_interval>,<cur_interval>,<latency>,
↪<timeout>
OK
```

Set Command

Function:

Update the parameters of Bluetooth LE connection.

Command:

```
AT+BLECONNPARAM=<conn_index>,<min_interval>,<max_interval>,<latency>,<timeout>
```

Response:

```
OK
```

If the setting fails, it will prompt the message below:

```
+BLECONNPARAM: <conn_index>,-1
```

Parameters

- **<conn_index>**: index of Bluetooth LE connection. Range: [0,2].
- **<min_interval>**: minimum connecting interval. Range: 0x0006 ~ 0x0C80.
- **<max_interval>**: maximum connecting interval. Range: 0x0006 ~ 0x0C80.
- **<cur_interval>**: current connecting interval.
- **<latency>**: latency. Range: 0x0000 ~ 0x01F3.
- **<timeout>**: timeout. Range: 0x000A ~ 0x0C80.

Note

- This command only supports the client role when updating its connection parameters. Of course, the connection has to be established first.

Example

```
AT+BLEINIT=1   // role: client
AT+BLECONN=0,"24:0a:c4:09:34:23"
AT+BLECONNPARAM=0,12,14,1,500
```

3.4.14 [ESP32 Only] AT+BLEDISCONN: End Bluetooth LE Connection

Execute Command

Function:

End the Bluetooth LE connection.

Command:

```
AT+BLEDISCONN=<conn_index>
```

Response:

```
OK  // The AT+BLEDISCONN command is received.
+BLEDISCONN:<conn_index>,<remote_address>  // The command is successful.
```

Parameters

- **<conn_index>**: index of Bluetooth LE connection. Range: [0,2].
- **<remote_address>**: remote Bluetooth LE address.

Note

- Only clients can call this command to terminate the connection.

Example

```
AT+BLEINIT=1   // role: client
AT+BLECONN=0,"24:0a:c4:09:34:23"
AT+BLEDISCONN=0
```

3.4.15 [ESP32 Only] AT+BLEDATALEN: Set Bluetooth LE Data Packet Length

Set Command

Function:

Set the length of Bluetooth LE data packet.

Command:

```
AT+BLEDATALEN=<conn_index>,<pkt_data_len>
```

Response:

```
OK
```

Parameters

- **<conn_index>**: index of Bluetooth LE connection. Range: [0,2].
- **<pkt_data_len>**: data packet's length. Range: 0x001b ~ 0x00fb.

Note

- The Bluetooth LE connection has to be established first.

Example

```
AT+BLEINIT=1   // role: client
AT+BLECONN=0,"24:0a:c4:09:34:23"
AT+BLEDATALEN=0,30
```

3.4.16 [ESP32 Only] AT+BLECFGMTU: Set Bluetooth LE MTU Length

Query Command

Function:

Query the length of the maximum transmission unit (MTU).

Command:

```
AT+BLECFGMTU?
```

Response:

```
+BLECFGMTU:<conn_index>,<mtu_size>
OK
```

Set Command

Function:

Set the length of the maximum transmission unit (MTU).

Command:

```
AT+BLECFGMTU=<conn_index>,<mtu_size>
```

Response:

```
OK  // The command is received.
```

Parameters

- **<conn_index>**: index of Bluetooth LE connection. Range: [0,2].
- **<mtu_size>**: MTU length.

Notes

- Bluetooth LE connection has to be established first.
- Only the client can call this command to set the length of MTU.
- The actual length of MTU needs to be negotiated. The `OK` response only indicates an attempt to negotiate the length. The actual length may not be the value you set. Therefore, it is recommended to run command *AT+BLECFGMTU?* to query the actual length.

Example

```
AT+BLEINIT=1   // role: client
AT+BLECONN=0,"24:0a:c4:09:34:23"
AT+BLECFGMTU=0,300
```

3.4.17 [ESP32 Only] AT+BLEGATTSSRVCRE: GATTS Creates Services

Execute Command

Function:

The Generic Attributes Server (GATTS) creates Bluetooth LE services.

Command:

```
AT+BLEGATTSSRVCRE
```

Response:

```
OK
```

Notes

- If you are using an ESP device as a Bluetooth LE server, a service bin should be downloaded into flash in order to provide services.
  - To learn how to generate a service bin, please refer to esp-at/tools/readme.md.
  - The download address of the service bin is the "ble_data" address in esp-at/module_config/module_${platform}_default/at_customize.csv.
- This command should be called immediately to create services, right after the Bluetooth LE server is initialized; If a Bluetooth LE connection is established first, the service creation will fail.
- If the Bluetooth LE client is initialized, you can use this command to create local services. Some GATTS commands can also be used, such as those to start and stop services, set attribute values, and send notifications/indications. See the list below for the specific commands.
  - *AT+BLEGATTSSRVCRE* (It is recommended to execute this command before the connection is established)
  - *AT+BLEGATTSSRVSTART* (It is recommended to execute this command before the connection is established)
  - *AT+BLEGATTSSRV*
  - *AT+BLEGATTSCHAR*
  - *AT+BLEGATTSNTFY*
  - *AT+BLEGATTSIND*
  - *AT+BLEGATTSSETATTR*

Example

```
AT+BLEINIT=2   // role: server
AT+BLEGATTSSRVCRE
```

3.4.18 [ESP32 Only] AT+BLEGATTSSRVSTART: GATTS Starts Services

Execute Command

Function:

GATTS starts all services.

Command:

```
AT+BLEGATTSSRVSTART
```

Set Command

Function:

GATTS starts a specific service.

Command:

```
AT+BLEGATTSSRVSTART=<srv_index>
```

Response:

```
OK
```

Parameter

- **<srv_index>**: service's index starting from 1.

Example

```
AT+BLEINIT=2   // role: server
AT+BLEGATTSSRVCRE
AT+BLEGATTSSRVSTART
```

3.4.19 [ESP32 Only] AT+BLEGATTSSRVSTOP: GATTS Stops Services

Execute Command

Function:

GATTS stops all services.

Command:

```
AT+BLEGATTSSRVSTOP
```

Set Command

Function:

GATTS stops a specific service.

Command:

```
AT+BLEGATTSSRVSTOP=<srv_index>
```

Response:

```
OK
```

Parameter

- **<srv_index>**: service's index starting from 1.

Example

```
AT+BLEINIT=2   // role: server
AT+BLEGATTSSRVCRE
AT+BLEGATTSSRVSTART
AT+BLEGATTSSRVSTOP
```

3.4.20 [ESP32 Only] AT+BLEGATTSSRV: GATTS Discovers Services

Query Command

Function:

GATTS discovers services.

Command:

```
AT+BLEGATTSSRV?
```

Response:

```
+BLEGATTSSRV:<srv_index>,<start>,<srv_uuid>,<srv_type>
OK
```

Parameters

- **<srv_index>**: service's index starting from 1.
- **<start>**:
  - 0: the service has not started.
  - 1: the service has already started.
- **<srv_uuid>**: service's UUID.
- **<srv_type>**: service's type.
  - 0: not primary service.
  - 1: primary service.

Example

```
AT+BLEINIT=2   // role: server
AT+BLEGATTSSRVCRE
AT+BLEGATTSSRV?
```

3.4.21 [ESP32 Only] AT+BLEGATTSCHAR: GATTS Discovers Characteristics

Query Command

Function:

GATTS discovers characteristics.

Command:

```
AT+BLEGATTSCHAR?
```

Response:

The response for a characteristic:

```
+BLEGATTSCHAR:"char",<srv_index>,<char_index>,<char_uuid>,<char_prop>
```

The response for a descriptor:

```
+BLEGATTSCHAR:"desc",<srv_index>,<char_index>,<desc_index>
OK
```

Parameters

- **<srv_index>**: service's index starting from 1.
- **<char_index>**: characteristic's index starting from 1.
- **<char_uuid>**: characteristic's UUID.
- **<char_prop>**: characteristic's properties.
- **<desc_index>**: descriptor's index.
- **<desc_uuid>**: descriptor's UUID.

Example

```
AT+BLEINIT=2   // role: server
AT+BLEGATTSSRVCRE
AT+BLEGATTSSRVSTART
AT+BLEGATTSCHAR?
```

3.4.22 [ESP32 Only] AT+BLEGATTSNTFY: Notify a Client of the Value of a Characteristic Value from the Server

Set Command

Function:

Notify a client of the value of a characteristic value from the server.

Command:

```
AT+BLEGATTSNTFY=<conn_index>,<srv_index>,<char_index>,<length>
```

Response:

```
>
```

The symbol > indicates that AT is ready for receiving serial data, and you can enter data now. When the requirement of data length determined by the parameter <length> is met, the notification starts.

If the data transmission is successful, AT returns:

```
OK
```

Parameters

- **<conn_index>**: index of Bluetooth LE connection. Range: [0,2].
- **<srv_index>**: service's index. It can be fetched with command *AT+BLEGATTSCHAR?*.
- **<char_index>**: characteristic's index. It can be fetched with command *AT+BLEGATTSCHAR?*.
- **<length>**: data length.

Example

```
AT+BLEINIT=2       // Role: server.
AT+BLEGATTSSRVCRE
AT+BLEGATTSSRVSTART
AT+BLEADVSTART     // Start advertising. After the client is connected, it must be␣
↪configured to receive notifications.
AT+BLEGATTSCHAR?   // Query the characteristics which the client will be notified of.
// For example, to notify of 4-byte data using the 6th characteristic in the 3rd␣
↪service, use the following command:
AT+BLEGATTSNTFY=0,3,6,4
// After the symbol ">" shows, enter the 4-byte data, such as "1234". Then the data␣
↪will be transmitted automatically.
```

3.4.23 [ESP32 Only] AT+BLEGATTSIND: Indicate the Characteristic Value from the Server to a Client

Set Command

Function:

Indicate the characteristic value from the server to a client.

Command:

```
AT+BLEGATTSIND=<conn_index>,<srv_index>,<char_index>,<length>
```

Response:

```
>
```

The symbol > indicates that AT is ready for receiving serial data and you can enter data now. When the requirement of data length determined by the parameter <length> is met, the indication starts.

If the data transmission is successful, AT returns:

```
OK
```

Parameters

- **<conn_index>**: index of Bluetooth LE connection. Range: [0,2].
- **<srv_index>**: service's index. It can be fetched with command *AT+BLEGATTSCHAR?*.
- **<char_index>**: characteristic's index; it can be fetched with command *AT+BLEGATTSCHAR?*.
- **<length>**: data length.

Example

```
AT+BLEINIT=2      // Role: server
AT+BLEGATTSSRVCRE
AT+BLEGATTSSRVSTART
AT+BLEADVSTART    // Start advertising. After the client is connected, it must be␣
↪configured to receive indications.
AT+BLEGATTSCHAR?  // Query the characteristics which the client can receive␣
↪indications.
// For example, to indicate 4 bytes of data using the 7th characteristic in the 3rd␣
↪service, use the following command:
AT+BLEGATTSIND=0,3,7,4
// After the symbol ">" shows, input 4 bytes of data, such as "1234". Then the data␣
↪will be transmitted automatically.
```

3.4.24 [ESP32 Only] AT+BLEGATTSSETATTR: GATTS Sets Characteristics

Set Command

Function:

GATTS sets its characteristic (descriptor).

Command:

```
AT+BLEGATTSSETATTR=<srv_index>,<char_index>,[<desc_index>],<length>
```

Response:

```
>
```

The symbol > indicates that AT is ready for receiving serial data and you can enter data now. When the requirement of data length determined by the parameter <length> is met, the setting starts.

If the setting is successful, AT returns:

```
OK
```

Parameters

- **<srv_index>**: service's index. It can be fetched with command *AT+BLEGATTSCHAR?*.
- **<char_index>**: characteristic's index; it can be fetched with command *AT+BLEGATTSCHAR?*.
- **[<desc_index>]**: descriptor's index.
  - If it is set, this command is used to set the value of the descriptor.

  - Otherwise, this command is used to set the value of the characteristic.

- **<length>**: data length.

Note

- If the value of `<length>` is larger than the maximum length allowed, the setting will fail. The service table is defined in *components/customized_partitions/raw_data/ble_data*.

Example

```
AT+BLEINIT=2   // Role: server.
AT+BLEGATTSSRVCRE
AT+BLEGATTSSRVSTART
AT+BLEGATTSCHAR?
// For example, to set 1 byte of data of the 1st characteristic in the 1st service,␣
↪use the following command:
AT+BLEGATTSSETATTR=1,1,,1
// After the symbol ">" shows, input 1 byte of data, such as "8". Then the setting␣
↪starts.
```

3.4.25 [ESP32 Only] AT+BLEGATTCPRIMSRV: GATTC Discovers Primary Services

Query Command

Function:

Generic Attributes Client (GATTC) discovers primary services.

Command:

```
AT+BLEGATTCPRIMSRV=<conn_index>
```

Response:

```
+BLEGATTCPRIMSRV:<conn_index>,<srv_index>,<srv_uuid>,<srv_type>
OK
```

Parameters

- **<conn_index>**: index of Bluetooth LE connection. Range: [0,2].
- **<srv_index>**: service's index starting from 1.
- **<srv_uuid>**: service's UUID.
- **<srv_type>**: service's type.
  - 0: not primary service.
  - 1: primary service.

Note

- The Bluetooth LE connection has to be established first.

Example

```
AT+BLEINIT=1   // role: client
AT+BLECONN=0,"24:12:5f:9d:91:98"
AT+BLEGATTCPRIMSRV=0
```

3.4.26 [ESP32 Only] AT+BLEGATTCINCLSRV: GATTC Discovers Included Services

Set Command

Function:

GATTC discovers included services.

Command:

```
AT+BLEGATTCINCLSRV=<conn_index>,<srv_index>
```

Response:

```
+BLEGATTCINCLSRV:<conn_index>,<srv_index>,<srv_uuid>,<srv_type>,<included_srv_uuid>,
↪<included_srv_type>
OK
```

Parameters

- **<conn_index>**: index of Bluetooth LE connection. Range: [0,2].
- **<srv_index>**: service's index. It can be fetched with command *AT+BLEGATTCPRIMSRV*=<conn_index>.
- **<srv_uuid>**: service's UUID.
- **<srv_type>**: service's type.
  - 0: not primary service.
  - 1: primary service.
- **<included_srv_uuid>**: included service's UUID.
- **<included_srv_type>**: included service's type.
  - 0: not primary service.
  - 1: primary service.

Note

- The Bluetooth LE connection has to be established first.

Example

```
AT+BLEINIT=1   // role: client
AT+BLECONN=0,"24:12:5f:9d:91:98"
AT+BLEGATTCPRIMSRV=0
AT+BLEGATTCINCLSRV=0,1  // set a specific index according to the result of the␣
↪previous command
```

3.4.27 [ESP32 Only] AT+BLEGATTCCHAR: GATTC Discovers Characteristics

Set Command

Function:

GATTC discovers characteristics.

Command:

```
AT+BLEGATTCCHAR=<conn_index>,<srv_index>
```

Response:

The response for a characteristic:

```
+BLEGATTCCHAR:"char",<conn_index>,<srv_index>,<char_index>,<char_uuid>,<char_prop>
```

The response for a descriptor:

```
+BLEGATTCCHAR:"desc",<conn_index>,<srv_index>,<char_index>,<desc_index>,<desc_uuid>
OK
```

Parameters

- **<conn_index>**: index of Bluetooth LE connection. Range: [0,2].
- **<srv_index>**: service's index. It can be fetched with command *AT+BLEGATTCPRIMSRV*=<conn_index>.
- **<char_index>**: characteristic's index starting from 1.
- **<char_uuid>**: characteristic's UUID.
- **<char_prop>**: characteristic's properties.
- **<desc_index>**: descriptor's index.
- **<desc_uuid>**: descriptor's UUID.

Note

- The Bluetooth LE connection has to be established first.

Example

```
AT+BLEINIT=1   // role: client
AT+BLECONN=0,"24:12:5f:9d:91:98"
AT+BLEGATTCPRIMSRV=0
AT+BLEGATTCCHAR=0,1 // set a specific index according to the result of the previous␣
↪command
```

(continues on next page)

(continued from previous page)

3.4.28 [ESP32 Only] AT+BLEGATTCRD: GATTC Reads Characteristics

Set Command

Function:

GATTC reads a characteristic or descriptor.

Command:

```
AT+BLEGATTCRD=<conn_index>,<srv_index>,<char_index>[,<desc_index>]
```

Response:

```
+BLEGATTCRD:<conn_index>,<len>,<value>
OK
```

Parameters

- **<conn_index>**: index of Bluetooth LE connection. Range: [0,2].
- **<srv_index>**: service's index. It can be fetched with command *AT+BLEGATTCPRIMSRV*=<conn_index>.
- **<char_index>**: characteristic's index; it can be fetched with command *AT+BLEGATTCCHAR*=<conn_index>,<srv_index>.
- **[<desc_index>]**: descriptor's index.
  - If it is set, the value of the target descriptor will be read.
  - if it is not set, the value of the target characteristic will be read.
- **<len>**: data length.
- **<char_value>**: characteristic's value. HEX string is read by command *AT+BLEGATTCRD*=<conn_index>,<srv_index>,<char_index>. For example, if the response is `+BLEGATTCRD:1,30`, it means that the value length is 1, and the content is "0x30".
- **[<desc_value>]**: descriptor's value. HEX string is read by command *AT+BLEGATTCRD*=<conn_index>,<srv_index>,<char_index>,<desc_index>. For example, if the response is `+BLEGATTCRD:4,30313233`, it means that the value length is 4, and the content is "0x30 0x31 0x32 0x33".

Notes

- The Bluetooth LE connection has to be established first.
- If the target characteristic cannot be read, it will return "ERROR".

Example

```
AT+BLEINIT=1   // Role: client.
AT+BLECONN=0,"24:12:5f:9d:91:98"
AT+BLEGATTCPRIMSRV=0
AT+BLEGATTCCHAR=0,3 // Set a specific index according to the result of the previous␣
↪command.
// For example, to read 1st descriptor of the 2nd characteristic in the 3rd service,␣
↪use the following command:
AT+BLEGATTCRD=0,3,2,1
```

3.4.29 [ESP32 Only] AT+BLEGATTCWR: GATTC Writes Characteristics

Set Command

Function:

GATTC writes characteristics or descriptors.

Command:

```
AT+BLEGATTCWR=<conn_index>,<srv_index>,<char_index>[,<desc_index>],<length>
```

Response:

```
>
```

The symbol > indicates that AT is ready for receiving serial data and you can enter data now. When the requirement of data length determined by the parameter <length> is met, the writing starts.

If the setting is successful, AT returns:

```
OK
```

Parameters

- **<conn_index>**: index of Bluetooth LE connection. Range: [0,2].
- **<srv_index>**: service's index. It can be fetched with command *AT+BLEGATTCPRIMSRV*=<conn_index>.
- **<char_index>**: characteristic's index; it can be fetched with command *AT+BLEGATTCCHAR*=<conn_index>,<srv_index>.
- **[<desc_index>]**: descriptor's index.
  - If it is set, the value of the target descriptor will be written.
  - If it is not set, the value of the target characteristic will be written.
- **<length>**: data length.

Notes

- The Bluetooth LE connection has to be established first.
- If the target characteristic cannot be written, it will return "ERROR".

Example

```
AT+BLEINIT=1   // Role: client.
AT+BLECONN=0,"24:12:5f:9d:91:98"
AT+BLEGATTCPRIMSRV=0
AT+BLEGATTCCHAR=0,3 // Set a specific index according to the result of the previous␣
↪command.
// For example, to write 6 bytes of data to the 4th characteristic in the 3rd service,
↪ use the following command:
AT+BLEGATTCWR=0,3,4,,6
// After the symbol ">" shows, input 6 bytes of data, such as "123456". Then the␣
↪writing starts.
```

3.4.30 [ESP32 Only] AT+BLESPPCFG: Query/Set Bluetooth LE SPP Parameters

Query Command

Function:

Query the parameters of Bluetooth LE Serial Port Profile (SPP).

Command:

```
AT+BLESPPCFG?
```

Response:

```
+BLESPPCFG:<tx_service_index>,<tx_char_index>,<rx_service_index>,<rx_char_index>
OK
```

Set Command

Function:

Set or reset the parameters of Bluetooth LE SPP.

Command:

```
AT+BLESPPCFG=<cfg_enable>[,<tx_service_index>,<tx_char_index>,<rx_service_index>,<rx_
↪char_index>]
```

Response:

```
OK
```

Parameters

- **<cfg_enable>**:
  - 0: all the SPP parameters will be reset, and the following four parameters don't need input.
  - 1: you should input the following four parameters.
- **<tx_service_index>**: tx service's index. It can be fetched with command *AT+BLEGATTCPRIMSRV*=<conn_index> and *AT+BLEGATTSSRV?*.

- **<tx_char_index>**: tx characteristic's index. It can be fetched with command *AT+BLEGATTCCHAR*=<conn_index>,<srv_index> and *AT+BLEGATTSCHAR?*.
- **<rx_service_index>**: rx service's index. It can be fetched with command *AT+BLEGATTCPRIMSRV*=<conn_index> and *AT+BLEGATTSSRV?*.
- **<rx_char_index>**: rx characteristic's index. It can be fetched with command *AT+BLEGATTCCHAR*=<conn_index>,<srv_index> and *AT+BLEGATTSCHAR?*.

Notes

- In Bluetooth LE client, the property of tx characteristic must be `write with response` or `write without response`, and the property of rx characteristic must be `indicate` or `notify`.
- In Bluetooth LE server, the property of tx characteristic must be `indicate` or `notify`, and the property of rx characteristic must be `write with response` or `write without response`.

Example

```
AT+BLESPPCFG=0           // reset Bluetooth LE SPP parameters
AT+BLESPPCFG=1,3,5,3,7  // set Bluetooth LE SPP parameters
AT+BLESPPCFG?            // query Bluetooth LE SPP parameters
```

3.4.31 [ESP32 Only] AT+BLESPP: Enter Bluetooth LE SPP Mode

Execute Command

Function:

Enter Bluetooth LE SPP mode.

Command:

```
AT+BLESPP
```

Response:

```
>
```

Notes

- If the Bluetooth LE SPP parameters are illegal, this command will return `ERROR`.
- During the SPP transmission, AT will not prompt any connection status changes unless bit2 of *AT+SYSMSG* is 1.

Example

```
AT+BLESPP   // enter Bluetooth LE SPP mode
```

3.4.32 [ESP32 Only] AT+BLESECPARAM: Query/Set Bluetooth LE Encryption Parameters

Query Command

Function:

Query the parameters of Bluetooth LE SMP.

Command:

```
AT+BLESECPARAM?
```

Response:

```
+BLESECPARAM:<auth_req>,<iocap>,<key_size>,<init_key>,<rsp_key>,<auth_option>
OK
```

Set Command

Function:

Set the parameters of Bluetooth LE SMP.

Command:

```
AT+BLESECPARAM=<auth_req>,<iocap>,<key_size>,<init_key>,<rsp_key>[,<auth_option>]
```

Response:

```
OK
```

Parameters

- **<auth_req>**: authentication request.
  - 0: NO_BOND
  - 1: BOND
  - 4: MITM
  - 8: SC_ONLY
  - 9: SC_BOND
  - 12: SC_MITM
  - 13: SC_MITM_BOND
- **<iocap>**: input and output capability.
  - 0: DisplayOnly
  - 1: DisplayYesNo
  - 2: KeyboardOnly
  - 3: NoInputNoOutput
  - 4: Keyboard display

- **<key_size>**: key length. Range: 7 ~ 16 bytes.
- **<init_key>**: initial key represented in bit combinations.
- **<rsp_key>**: response key represented in bit combinations.
- **<auth_option>**: authentication option of security.
  - 0: Select the security level automatically.
  - 1: If it cannot follow the preset security level, the connection will disconnect.

Note

- The bit pattern for parameters `<init_key>` and `<rsp_key>` is:
  - Bit0: Used to exchange the encryption key in the init key & response key.
  - Bit1: Used to exchange the IRK key in the init key & response key.
  - Bit2: Used to exchange the CSRK key in the init key & response key.
  - Bit3: Used to exchange the link key (only used in the Bluetooth LE & BR/EDR coexist mode) in the init key & response key.

Example

```
AT+BLESECPARAM=1,4,16,3,3,0
```

3.4.33 [ESP32 Only] AT+BLEENC: Initiate Bluetooth LE Encryption Request

Set Command

Function:

Start a pairing request

Command:

```
AT+BLEENC=<conn_index>,<sec_act>
```

Response:

```
OK
```

Parameters

- **<conn_index>**: index of Bluetooth LE connection. Range: [0,2].
- **<sec_act>**:
  - 0: SEC_NONE
  - 1: SEC_ENCRYPT
  - 2: SEC_ENCRYPT_NO_MITM
  - 3: SEC_ENCRYPT_MITM

Note

- Before running this command, please set the security parameters and connection with remote devices.

Example

```
AT+BLESECPARAM=1,4,16,3,3
AT+BLEENC=0,3
```

3.4.34 [ESP32 Only] AT+BLEENCRSP: Respond to the Pairing Request from the Peer Device

Set Command

Function:

Respond to the pairing request from the peer device.

Command:

```
AT+BLEENCRSP=<conn_index>,<accept>
```

Response:

```
OK
```

Parameters

- **<conn_index>**: index of Bluetooth LE connection. Range: [0,2].
- **<accept>**:
  - 0: reject
  - 1: accept

Note

- After running this command, AT will output the pairing result at the end of the pairing process.

```
+BLEAUTHCMPL:<conn_index>,<enc_result>
```

- **<conn_index>**: index of Bluetooth LE connection. Range: [0,2].
- **<enc_result>**:
  - 0: encryption succeeded
  - 1: encryption failed

Example

```
AT+BLEENCRSP=0,1
```

3.4.35 [ESP32 Only] AT+BLEKEYREPLY: Reply the Key Value to the Peer Device in the Legacy Connection Stage

Set Command

Function:

Reply a pairing key.

Command:

```
AT+BLEKEYREPLY=<conn_index>,<key>
```

Response:

```
OK
```

Parameters

- **<conn_index>**: index of Bluetooth LE connection. Range: [0,2].
- **<key>**: pairing key.

Example

```
AT+BLEKEYREPLY=0,649784
```

3.4.36 [ESP32 Only] AT+BLECONFREPLY: Reply the Confirm Value to the Peer Device in the Legacy Connection Stage

Set Command

Function:

Reply a pairing result.

Command:

```
AT+BLECONFREPLY=<conn_index>,<confirm>
```

Response:

```
OK
```

Parameters

- **<conn_index>**: index of Bluetooth LE connection. Range: [0,2].
- **<confirm>**:
  - 0: No
  - 1: Yes

Example

```
AT+BLECONFREPLY=0,1
```

3.4.37 [ESP32 Only] AT+BLEENCDEV: Query Bonded Bluetooth LE Encryption Device List

Query Command

Function:

Query bonded Bluetooth LE encryption device list.

Command:

```
AT+BLEENCDEV?
```

Response:

```
+BLEENCDEV:<enc_dev_index>,<mac_address>
OK
```

Parameters

- **<enc_dev_index>**: index of the bonded devices.
- **<mac_address>**: MAC address.

Example

```
AT+BLEENCDEV?
```

3.4.38 [ESP32 Only] AT+BLEENCCLEAR: Clear Bluetooth LE Encryption Device List

Set Command

Function:

Remove a device from the security database list with a specific index.

Command:

```
AT+BLEENCCLEAR=<enc_dev_index>
```

Response:

```
OK
```

Execute Command

Function:

Remove all devices from the security database.

Command:

```
AT+BLEENCCLEAR
```

Response:

```
OK
```

Parameter

- **<enc_dev_index>**: index of the bonded devices.

Example

```
AT+BLEENCCLEAR
```

3.4.39 [ESP32 Only] AT+BLESETKEY: Set Bluetooth LE Static Pair Key

Query Command

Function:

Query the Bluetooth LE static pair key. If it is not set, AT will return -1.

Command:

```
AT+BLESETKEY?
```

Response:

```
+BLESETKEY:<static_key>
OK
```

Set Command

Function:

Set a Bluetooth LE static pair key for all Bluetooth LE connections.

Command:

```
AT+BLESETKEY=<static_key>
```

Response:

```
OK
```

Parameter

- **<static_key>**: static Bluetooth LE pair key.

Example

```
AT+BLESETKEY=123456
```

3.4.40 [ESP32 Only] AT+BLEHIDINIT: Bluetooth LE HID Profile Initialization

Query Command

Function:

Query the initialization status of Bluetooth LE HID profile.

Command:

```
AT+BLEHIDINIT?
```

Response:

If Bluetooth LE HID device profile is not initialized, AT will return:

```
+BLEHIDINIT:0
OK
```

If Bluetooth LE HID device profile is initialized, AT will return:

```
+BLEHIDINIT:1
OK
```

Set Command

Function:

Initialize the Bluetooth LE HID profile.

Command:

```
AT+BLEHIDINIT=<init>
```

Response:

```
OK
```

Parameter

- **<init>**:
  - 0: deinit Bluetooth LE HID profile
  - 1: init Bluetooth LE HID profile

Note

- The Bluetooth LE HID command cannot be used at the same time with general GATT/GAP commands.

Example

```
AT+BLEHIDINIT=1
```

3.4.41 [ESP32 Only] AT+BLEHIDKB: Send Bluetooth LE HID Keyboard Information

Set Command

Function:

Send keyboard information.

Command:

```
AT+BLEHIDKB=<Modifier_keys>,<key_1>,<key_2>,<key_3>,<key_4>,<key_5>,<key_6>
```

Response:

```
OK
```

Parameters

- **<Modifier_keys>**: modifier keys mask
- **<key_1>**: key code 1
- **<key_2>**: key code 2
- **<key_3>**: key code 3
- **<key_4>**: key code 4
- **<key_5>**: key code 5
- **<key_6>**: key code 6

Note

- For more information about key codes, please refer to the chapter Keyboard/Keypad Page of Universal Serial Bus HID Usage Tables.

Example

```
AT+BLEHIDKB=0,4,0,0,0,0,0    // input the string "a"
```

3.4.42 [ESP32 Only] AT+BLEHIDMUS: Send Bluetooth LE HID Mouse Information

Set Command

Function:

Send mouse information.

Command:

```
AT+BLEHIDMUS=<buttons>,<X_displacement>,<Y_displacement>,<wheel>
```

Response:

```
OK
```

Parameters

- **<buttons>**: mouse button
- **<X_displacement>**: X displacement
- **<Y_displacement>**: Y displacement
- **<wheel>**: wheel

Example

```
AT+BLEHIDMUS=0,10,10,0
```

3.4.43 [ESP32 Only] AT+BLEHIDCONSUMER: Send Bluetooth LE HID Consumer Information

Set Command

Function:

Send consumer information.

Command:

```
AT+BLEHIDCONSUMER=<consumer_usage_id>
```

Response:

```
OK
```

Parameter

- **<consumer_usage_id>**: consumer ID, such as power, reset, help, volume and so on. See chapter Consumer Page (0x0C) of HID Usage Tables for Universal Serial Bus (USB) for more information.

Example

```
AT+BLEHIDCONSUMER=233   // volume up
```

3.4.44 [ESP32 Only] AT+BLUFI: Start or Stop BluFi

Query Command

Function:

Query the status of BluFi.

Command:

```
AT+BLUFI?
```

Response:

If BluFi is not started, it will return:

```
+BLUFI:0
OK
```

If BluFi is started, it will return:

```
+BLUFI:1
OK
```

Set Command

Function:

Start or stop BluFi.

Command:

```
AT+BLUFI=<option>[,<auth floor>]
```

Response:

```
OK
```

Parameter

- **<option>**:
  - 0: stop BluFi
  - 1: start BluFi

- **<auth floor>**: Wi-Fi authentication mode floor. ESP-AT will not connect to the AP whose authmode is lower than this floor.
  - 0: OPEN (Default)
  - 1: WEP
  - 2: WPA_PSK
  - 3: WPA2_PSK
  - 4: WPA_WPA2_PSK
  - 5: WPA2_ENTERPRISE
  - 6: WPA3_PSK
  - 7: WPA2_WPA3_PSK

Example

```
AT+BLUFI=1
```

3.4.45 [ESP32 Only] AT+BLUFINAME: Query/Set BluFi Device Name

Query Command

Function:

Query the BluFi name.

Command:

```
AT+BLUFINAME?
```

Response:

```
+BLUFINAME:<device_name>
OK
```

Set Command

Function:

Set the BluFi device name.

Command:

```
AT+BLUFINAME=<device_name>
```

Response:

```
OK
```

Parameter

- **<device_name>**: the name of BluFi device.

Notes

- If you need to set BluFi name, please set it before command *AT+BLUFI=1*. Otherwise, it will use the default name `BLUFI_DEVICE`.
- The maximum length of BluFi name is 29 bytes.

Example

```
AT+BLUFINAME="BLUFI_DEV"
AT+BLUFINAME?
```

3.5 [ESP32 Only] Classic Bluetooth® AT Commands

[]

ESP32 AT firmware supports Bluetooth® Core Specification Version 5.0.

- [ESP32 Only] *AT+BTINIT*: Classic Bluetooth initialization.
- [ESP32 Only] *AT+BTNAME*: Query/Set Classic Bluetooth device name.
- [ESP32 Only] *AT+BTSCANMODE*: Set Classic Bluetooth scan mode.
- [ESP32 Only] *AT+BTSTARTDISC*: Start Classic Bluetooth discovery.
- [ESP32 Only] *AT+BTSPPINIT*: Classic Bluetooth SPP profile initialization.
- [ESP32 Only] *AT+BTSPPCONN*: Query/Establish SPP connection.
- [ESP32 Only] *AT+BTSPPDISCONN*: End SPP connection.
- [ESP32 Only] *AT+BTSPPSTART*: Start Classic Bluetooth SPP profile.
- [ESP32 Only] *AT+BTSPPSEND*: Send data to remote Classic Bluetooth SPP device.
- [ESP32 Only] *AT+BTA2DPINIT*: Classic Bluetooth A2DP profile initialization.
- [ESP32 Only] *AT+BTA2DPCONN*: Establish A2DP connection.
- [ESP32 Only] *AT+BTA2DPDISCONN*: End A2DP connection.
- [ESP32 Only] *AT+BTA2DPSRC*: Query/Set the audio file URL.
- [ESP32 Only] *AT+BTA2DPCTRL*: Control the audio play.
- [ESP32 Only] *AT+BTSECPARAM*: Query/Set the Classic Bluetooth security parameters.
- [ESP32 Only] *AT+BTKEYREPLY*: Input the Simple Pair Key.
- [ESP32 Only] *AT+BTPINREPLY*: Input the Legacy Pair PIN Code.
- [ESP32 Only] *AT+BTSECCFM*: Reply the confirm value to the peer device in the legacy connection stage.
- [ESP32 Only] *AT+BTENCDEV*: Query Classic Bluetooth encryption device list.
- [ESP32 Only] *AT+BTENCCLEAR*: Clear Classic Bluetooth encryption device list.
- [ESP32 Only] *AT+BTCOD*: Set class of devices.
- [ESP32 Only] *AT+BTPOWER*: Query/Set power of Classic Bluetooth.

3.5.1 [ESP32 Only] AT+BTINIT: Classic Bluetooth Initialization

Query Command

Function:

Query the initialization status of Classic Bluetooth.

Command:

```
AT+BTINIT?
```

Response:

If Classic Bluetooth is initialized, AT will return:

```
+BTINIT:1
OK
```

If Classic Bluetooth is not initialized, AT will return:

```
+BTINIT:0
OK
```

Set Command

Function:

Initialize or deinitialize Classic Bluetooth.

Command:

```
AT+BTINIT=<init>
```

Response:

```
OK
```

Parameter

- **<init>**:
  - 0: deinitialize Classic Bluetooth.
  - 1: initialize Classic Bluetooth.

Example

```
AT+BTINIT=1
```

3.5.2 [ESP32 Only] AT+BTNAME: Query/Set Classic Bluetooth Device Name

Query Command

Function:

Query the Classic Bluetooth device name.

Command:

```
AT+BTNAME?
```

Response:

```
+BTNAME:<device_name>
OK
```

Set Command

Function:

Set the Classic Bluetooth device name.

Command:

```
AT+BTNAME=<device_name>
```

Response:

```
OK
```

Parameter

- **<device_name>**: the Classic Bluetooth device name. The maximum length is 32.

Notes

- The configuration changes will be saved in the NVS area if *AT+SYSSTORE=1*.
- The default Classic Bluetooth device name is "ESP32_AT".

Example

```
AT+BTNAME="esp_demo"
```

3.5.3 [ESP32 Only] AT+BTSCANMODE: Set Classic Bluetooth Scan Mode

Set Command

Function:

Set the scan mode of Classic Bluetooth.

Command:

```
AT+BTSCANMODE=<scan_mode>
```

Response:

```
OK
```

Parameter

- **<scan_mode>**:
  - 0: Neither discoverable nor connectable.
  - 1: Connectable but not discoverable.
  - 2: Both discoverable and connectable.
  - 3: Discoverable but not connectable.

Example

```
AT+BTSCANMODE=2    // both discoverable and connectable
```

3.5.4 [ESP32 Only] AT+BTSTARTDISC: Start Classic Bluetooth Discovery

Set Command

Function:

Start Classic Bluetooth discovery.

Command:

```
AT+BTSTARTDISC=<inq_mode>,<inq_len>,<inq_num_rsps>
```

Response:

```
+BTSTARTDISC:<bt_addr>,<dev_name>,<major_dev_class>,<minor_dev_class>,<major_srv_
↪class>,<rssi>

OK
```

Parameters

- **<inq_mode>**:
  - 0: general inquiry mode.
  - 1: limited inquiry mode.
- **<inq_len>**: inquiry duration. Range: 0x01 ~ 0x30.
- **<inq_num_rsps>**: number of inquiry responses that can be received. If you set it to 0, AT will receive an unlimited number of responses.

- **<bt_addr>**: Classic Bluetooth address.
- **<dev_name>**: device name.
- **<major_dev_class>**:
  - 0x0: miscellaneous.
  - 0x1: computer.
  - 0x2: phone (cellular, cordless, pay phone, modem).
  - 0x3: LAN, Network Access Point.
  - 0x4: audio/video (headset, speaker, stereo, video display, VCR).
  - 0x5: peripheral (mouse, joystick, keyboard).
  - 0x6: imaging (printer, scanner, camera, display).
  - 0x7: wearable.
  - 0x8: toy.
  - 0x9: health.
  - 0x1F: uncategorized device.
- **<minor_dev_class>**: please refer to Minor Device Class field.
- **<major_srv_class>**:
  - 0x0: an invalid value.
  - 0x1: Limited Discoverable Mode.
  - 0x8: positioning (location identification).
  - 0x10: networking, such as LAN, Ad hoc.
  - 0x20: rendering, such as printing, speakers.
  - 0x40: capturing, such as scanner, microphone.
  - 0x80: object transfer, such as v-Inbox, v-Folder.
  - 0x100: audio, such as speaker, microphone, headerset service.
  - 0x200: telephony, such as cordless telephony, modem, headset service.
  - 0x400: information, such as WEB-server, WAP-server.
- **<rssi>**: signal strength.

Example

```
AT+BTINIT=1
AT+BTSCANMODE=2
AT+BTSTARTDISC=0,10,10
```

3.5.5 [ESP32 Only] AT+BTSPPINIT: Classic Bluetooth SPP Profile Initialization

Query Command

Function:

Query the initialization status of Classic Bluetooth SPP profile.

Command:

```
AT+BTSPPINIT?
```

Response:

If Classic Bluetooth SPP profile is initialized, it will return:

```
+BTSPPINIT:1
OK
```

If Classic Bluetooth SPP profile is not initialized, it will return:

```
+BTSPPINIT:0
OK
```

Set Command

Function:

Initialize or deinitialize Classic Bluetooth SPP profile.

Command:

```
AT+BTSPPINIT=<init>
```

Response:

```
OK
```

Parameter

- **<init>**:
  - 0: deinitialize Classic Bluetooth SPP profile.
  - 1: initialize Classic Bluetooth SPP profile, the role is master.
  - 2: initialize Classic Bluetooth SPP profile, the role is slave.

Example

```
AT+BTSPPINIT=1    // master
AT+BTSPPINIT=2    // slave
```

3.5.6 [ESP32 Only] AT+BTSPPCONN: Query/Establish SPP Connection

Query Command

Function:

Query Classic Bluetooth SPP connection.

Command:

```
AT+BTSPPCONN?
```

Response:

```
+BTSPPCONN:<conn_index>,<remote_address>
OK
```

If the connection has not been established, AT will return:

```
+BTSPPCONN:-1
```

Set Command

Function:

Establish the Classic Bluetooth SPP connection.

Command:

```
AT+BTSPPCONN=<conn_index>,<sec_mode>,<remote_address>
```

Response:

```
OK
```

If the connection is established successfully, AT will return:

```
+BTSPPCONN:<conn_index>,<remote_address>
```

Otherwise, AT will return:

```
+BTSPPCONN:<conn_index>,-1
```

Parameters

- **<conn_index>**: index of Classic Bluetooth SPP connection. Only 0 is supported for the single connection right now.
- **<sec_mode>**:
  - 0x0000: no security.
  - 0x0001: authorization required (only needed for out going connection).
  - 0x0036: encryption required.
  - 0x3000: Man-In-The-Middle protection.

  - 0x4000: Min 16 digit for pin code.
- **<remote_address>**: remote Classic Bluetooth SPP device address.

Example

```
AT+BTSPPCONN=0,0,"24:0a:c4:09:34:23"
```

3.5.7 [ESP32 Only] AT+BTSPPDISCONN: End SPP Connection

Execute Command

Function:

End the Classic Bluetooth SPP connection.

Command:

```
AT+BTSPPDISCONN=<conn_index>
```

Response:

```
OK
```

If the command is successful, it will prompt:

```
+BTSPPDISCONN:<conn_index>,<remote_address>
```

If the command is fail, it will prompt:

```
+BTSPPDISCONN:-1
```

Parameters

- **<conn_index>**: index of Classic Bluetooth SPP connection. Only 0 is supported for the single connection right now.
- **<remote_address>**: remote Classic Bluetooth A2DP device address.

Example

```
AT+BTSPPDISCONN=0
```

3.5.8 [ESP32 Only] AT+BTSPPSEND: Send Data to Remote Classic Bluetooth SPP Device

Execute Command

Function:

Enter Classic Bluetooth SPP mode.

Command:

```
AT+BTSPPSEND
```

Response:

```
>
```

Set Command

Function:

Send data to the remote Classic Bluetooth SPP device.

Command:

```
AT+BTSPPSEND=<conn_index>,<data_len>
```

Response:

```
OK
```

Parameters

- **<conn_index>**: index of Classic Bluetooth SPP connection. Only 0 is supported for the single connection right now.
- **<data_len>**: the length of the data which is ready to be sent.

Notes

- The wrap return is > after this command is executed. Then, the ESP device enters UART Bluetooth passthrough mode. When the packet which only contains `+++` is received, the device returns to normal command mode. Please wait for at least one second before sending the next AT command.

Example

```
AT+BTSPPSEND=0,100
AT+BTSPPSEND
```

3.5.9 [ESP32 Only] AT+BTSPPSTART: Start Classic Bluetooth SPP Profile

Execute Command

Function:

Start Classic Bluetooth SPP profile.

Command:

```
AT+BTSPPSTART
```

Response:

```
OK
```

Note

- During the SPP transmission, AT will not prompt any connection status changes unless bit2 of *AT+SYSMSG* is 1.

Example

```
AT+BTSPPSTART
```

3.5.10 [ESP32 Only] AT+BTA2DPINIT: Classic Bluetooth A2DP Profile Initialization

Query Command

Function:

Query the initialization status of Classic Bluetooth A2DP profile.

Command:

```
AT+BTA2DPINIT?
```

Response:

If Classic Bluetooth A2DP profile is initialized, AT will return:

```
+BTA2DPINIT:1
OK
```

Otherwise, AT will return:

```
+BTA2DPINIT:0
OK
```

Set Command

Function:

Initialize or deinitialize Classic Bluetooth A2DP profile.

Command:

```
AT+BTA2DPINIT=<role>,<init_val>
```

Response:

```
OK
```

Parameters

- **<role>**:
  - 0: source.
  - 1: sink.
- **<init_val>**:
  - 0: deinitialize Classic Bluetooth A2DP profile.
  - 1: initialize Classic Bluetooth A2DP profile.

Example

```
AT+BTA2DPINIT=0,1
```

3.5.11 [ESP32 Only] AT+BTA2DPCONN: Query/Establish A2DP Connection

Query Command

Function:

Query Classic Bluetooth A2DP connection.

Command:

```
AT+BTA2DPCONN?
```

Response:

```
+BTA2DPCONN:<conn_index>,<remote_address>
OK
```

If the connection has not been established, AT will NOT return the parameter <conn_index> and <remote_address>.

Set Command

Function:

Establish the Classic Bluetooth A2DP connection.

Command:

```
AT+BTA2DPCONN=<conn_index>,<remote_address>
```

Response:

```
OK
```

If the connection is established successfully, it will prompt the message below:

```
+BTA2DPCONN:<conn_index>,<remote_address>
```

Otherwise, it will return:

```
+BTA2DPCONN:<conn_index>,-1
```

Parameters

- **<conn_index>**: index of Classic Bluetooth A2DP connection. Only 0 is supported for the single connection right now.
- **<remote_address>**: remote Classic Bluetooth A2DP device address.

Example

```
AT+BTA2DPCONN=0,0,0,"24:0a:c4:09:34:23"
```

3.5.12 [ESP32 Only] AT+BTA2DPDISCONN: End A2DP Connection

Execute Command

Function:

End the Classic Bluetooth A2DP connection.

Command:

```
AT+BTA2DPDISCONN=<conn_index>
```

Response:

```
+BTA2DPDISCONN:<conn_index>,<remote_address>
OK
```

Parameters

- **<conn_index>**: index of Classic Bluetooth A2DP connection. Only 0 is supported for the single connection right now.
- **<remote_address>**: remote Classic Bluetooth A2DP device address.

Example

```
AT+BTA2DPDISCONN=0
```

3.5.13 [ESP32 Only] AT+BTA2DPSRC: Query/Set the Audio File URL

Query Command

Function:

Query the audio file URL.

Command:

```
AT+BTA2DPSRC?
```

Response:

```
+BTA2DPSRC:<url>,<type>
OK
```

Execute Command

Function:

Set the audio file URL.

Command:

```
AT+BTA2DPSRC=<conn_index>,<url>
```

Response:

```
OK
```

Parameters

- **<conn_index>**: index of Classic Bluetooth A2DP connection. Only 0 is supported for the single connection right now.
- **<url>**: the path of the source file. HTTP, HTTPS and FLASH are currently supported.
- **<type>**: the type of audio file, such as "mp3".

Note

- Only mp3 format is currently supported.

Example

```
AT+BTA2DPSRC=0,"https://dl.espressif.com/dl/audio/ff-16b-2c-44100hz.mp3"
AT+BTA2DPSRC=0,"flash://spiffs/zhifubao.mp3"
```

3.5.14 [ESP32 Only] AT+BTA2DPCTRL: Control the Audio Play

Execute Command

Function:

Control the audio play.

Command:

```
AT+BTA2DPCTRL=<conn_index>,<ctrl>
```

Response:

```
OK
```

Parameters

- **<conn_index>**: index of Classic Bluetooth A2DP connection. Only 0 is supported for the single connection right now.
- **<ctrl>**: types of control.
  - 0: A2DP Sink, stop play.
  - 1: A2DP Sink, start play.
  - 2: A2DP Sink, forward.
  - 3: A2DP Sink, backward.
  - 4: A2DP Sink, fastward start.
  - 5: A2DP Sink, fastward stop.
  - 0: A2DP Source, stop play.
  - 1: A2DP Source, start play.
  - 2: A2DP Source, suspend.

Example

```
AT+BTA2DPCTRL=0,1  // start play audio
```

3.5.15 [ESP32 Only] AT+BTSECPARAM: Query/Set the Classic Bluetooth Security Parameters

Query Command

Function:

Query Classic Bluetooth security parameters.

Command:

```
AT+BTSECPARAM?
```

Response:

```
+BTSECPARAM:<io_cap>,<pin_type>,<pin_code>
OK
```

Set Command

Function:

Set the Classic Bluetooth security parameters.

Command:

```
AT+BTSECPARAM=<io_cap>,<pin_type>,<pin_code>
```

Response:

```
OK
```

Parameters

- **<io_cap>**: input and output capability.
  - 0: DisplayOnly.
  - 1: DisplayYesNo.
  - 2: KeyboardOnly.
  - 3: NoInputNoOutput.
- **<pin_type>**: use variable or fixed PIN.
  - 0: variable.
  - 1: fixed.
- **<pin_code>**: Legacy Pair PIN Code. Maximum: 16 bytes.

Note

- If you set the parameter `<pin_type>` to 0, `<pin_code>` will be ignored.

Example

```
AT+BTSECPARAM=3,1,"9527"
```

3.5.16 [ESP32 Only] AT+BTKEYREPLY: Input the Simple Pair Key

Execute Command

Function:

Input the Simple Pair Key.

Command:

```
AT+BTKEYREPLY=<conn_index>,<Key>
```

Response:

```
OK
```

Parameters

- **<conn_index>**: index of Classic Bluetooth connection. Currently, only 0 is supported for the single connection.
- **<Key>**: the Simple Pair Key.

Example

```
AT+BTKEYREPLY=0,123456
```

3.5.17 [ESP32 Only] AT+BTPINREPLY: Input the Legacy Pair PIN Code

Execute Command

Function:

Input the Legacy Pair PIN Code.

Command:

```
AT+BTPINREPLY=<conn_index>,<Pin>
```

Response:

```
OK
```

Parameters

- **<conn_index>**: index of Classic Bluetooth connection. Currently, only 0 is supported for the single connection.
- **<Pin>**: the Legacy Pair PIN Code.

Example

```
AT+BTPINREPLY=0,"6688"
```

3.5.18 [ESP32 Only] AT+BTSECCFM: Reply the Confirm Value to the Peer Device in the Legacy Connection Stage

Execute Command

Function:

Reply the confirm value to the peer device in the legacy connection stage.

Command:

```
AT+BTSECCFM=<conn_index>,<accept>
```

Response:

```
OK
```

Parameters

- **<conn_index>**: index of Classic Bluetooth connection. Currently, only 0 is supported for the single connection.
- **<accept>**: reject or accept.
  - 0: reject.
  - 1: accept.

Example

```
AT+BTSECCFM=0,1
```

3.5.19 [ESP32 Only] AT+BTENCDEV: Query Classic Bluetooth Encryption Device List

Query Command

Function:

Query the bound devices.

Command:

```
AT+BTENCDEV?
```

Response:

```
+BTENCDEV:<enc_dev_index>,<mac_address>
OK
```

Parameters

- **<enc_dev_index>**: index of the bound devices.
- **<mac_address>**: MAC address.

Example

```
AT+BTENCDEV?
```

3.5.20 [ESP32 Only] AT+BTENCCLEAR: Clear Classic Bluetooth Encryption Device List

Set Command

Function:

Remove a device from the security database list with a specific index.

Command:

```
AT+BTENCCLEAR=<enc_dev_index>
```

Response:

```
OK
```

Execute Command

Function:

Remove all devices from the security database.

Command:

```
AT+BLEENCCLEAR
```

Response:

```
OK
```

Parameter

- **<enc_dev_index>**: index of the bound devices.

Example

```
AT+BTENCCLEAR
```

3.5.21 [ESP32 Only] AT+BTCOD: Set Class of Devices

Set Command

Function:

Set the Classic Bluetooth class of devices.

Command:

```
AT+BTCOD=<major>,<minor>,<service>
```

Response:

```
OK
```

Parameters

- **<major>**: major class.
- **<minor>**: minor class.
- **<service>**: service class.

Example

```
AT+BTCOD=6,32,32   // the printer
```

3.5.22 [ESP32 Only] AT+BTPOWER: Query/Set TX power of Classic Bluetooth

Query Command

Function:

Query Classic Bluetooth tx power level.

Command:

```
AT+BTPOWER?
```

Response:

```
+BTPOWER:<min_tx_power>,<max_tx_power>
OK
```

Set Command

Function:

Set the Classic Bluetooth tx power.

Command:

```
AT+BTPOWER=<min_tx_power>,<max_tx_power>
```

Response:

```
OK
```

Parameters

- **<min_tx_power>**: The minimum power level. Range: [0,7].
- **<max_tx_power>**: The maximum power level. Range: [0,7].

Example

```
AT+BTPOWER=5,6   // set Classic Bluetooth tx power.
```

3.6 MQTT AT Commands

[]

- *AT+MQTTUSERCFG*: Set MQTT user configuration

3.6.1 AT+MQTTUSERCFG: Set MQTT User Configuration

Set Command

Function:

Set MQTT User Configuration.

Command:

```
AT+MQTTUSERCFG=<LinkID>,<scheme>,<"client_id">,<"username">,<"password">,<cert_key_ID>
↪,<CA_ID>,<"path">
```

Response:

```
OK
```

Parameters

- **<LinkID>**: currently only supports link ID 0.
- **<scheme>**:
  - 1: MQTT over TCP.
  - 2: MQTT over TLS (no certificate verify).
  - 3: MQTT over TLS (verify server certificate).
  - 4: MQTT over TLS (provide client certificate).
  - 5: MQTT over TLS (verify server certificate and provide client certificate).
  - 6: MQTT over WebSocket (based on TCP).
  - 7: MQTT over WebSocket Secure (based on TLS, no certificate verify).
  - 8: MQTT over WebSocket Secure (based on TLS, verify server certificate).
  - 9: MQTT over WebSocket Secure (based on TLS, provide client certificate).

  - 10: MQTT over WebSocket Secure (based on TLS, verify server certificate and provide client certificate).
- **<client_id>**: MQTT client ID. Maximum length: 256 bytes.
- **<username>**: the username to login to the MQTT broker. Maximum length: 64 bytes.
- **<password>**: the password to login to the MQTT broker. Maximum length: 64 bytes.
- **<cert_key_ID>**: certificate ID. Currently, ESP-AT only supports one certificate for ID 0.
- **<CA_ID>**: CA ID. Currently, ESP-AT only supports one CA for ID 0.
- **<path>**: the path of the resource. Maximum length: 32 bytes.

Note

- The length of the entire AT command should be less than 256 bytes.

3.6.2 AT+MQTTCLIENTID: Set MQTT Client ID

Set Command

Function:

Set MQTT Client ID.

Command:

```
AT+MQTTCLIENTID=<LinkID>,<"client_id">
```

Response:

```
OK
```

Parameters

- **<LinkID>**: currently only supports link ID 0.
- **<client_id>**: MQTT client ID.

Notes

- The length of the entire AT command should be less than 256 bytes.
- The command *AT+MQTTUSERCFG* can also set MQTT client ID. The differences between the two commands include:
  - You can use `AT+MQTTCLIENTID` to set a relatively long client ID since there is a limitation on the length of the `AT+MQTTUSERCFG` command.
  - You should set `AT+MQTTCLIENTID` after setting the `AT+MQTTUSERCFG` command.

3.6.3 AT+MQTTUSERNAME: Set MQTT Username

Set Command

Function:

Set MQTT username.

Command:

```
AT+MQTTUSERNAME=<LinkID>,<"username">
```

Response:

```
OK
```

Parameters

- **<LinkID>**: only supports link ID 0 currently.
- **<username>**: the username to login to the MQTT broker.

Notes

- The length of the entire AT command should be less than 256 bytes.
- The command *AT+MQTTUSERCFG* can also set MQTT username. The differences between the two commands include:
  - You can use `AT+MQTTUSERNAME` to set a relatively long username since there is a limitation on the length of the `AT+MQTTUSERCFG` command.
  - You should set `AT+MQTTUSERNAME` after setting the command `AT+MQTTUSERCFG`.

3.6.4 AT+MQTTPASSWORD: Set MQTT Password

Set Command

Function:

Set MQTT password.

Command:

```
AT+MQTTPASSWORD=<LinkID>,<"password">
```

Response:

```
OK
```

Parameters

- **<LinkID>**: only supports link ID 0 currently.
- **<password>**: the password to login to the MQTT broker.

Notes

- The length of the entire AT command should be less than 256 bytes.
- The command *AT+MQTTUSERCFG* can also set MQTT password. The differences between the two commands include:
  - You can use `AT+MQTTPASSWORD` to set a relatively long password since there is a limitation on the length of the `AT+MQTTUSERCFG` command.
  - You should set `AT+MQTTPASSWORD` after setting the command `AT+MQTTUSERCFG`.

3.6.5 AT+MQTTCONNCFG: Set Configuration of MQTT Connection

Set Command

Function:

Set configuration of MQTT Connection.

Command:

```
AT+MQTTCONNCFG=<LinkID>,<keepalive>,<disable_clean_session>,<"lwt_topic">,<"lwt_msg">,
↪<lwt_qos>,<lwt_retain>
```

Response:

```
OK
```

Parameters

- **<LinkID>**: only supports link ID 0 currently.
- **<keepalive>**: timeout of MQTT ping. Unit: second. Range [0,7200]. The default value is 0, which will be force-changed to 120 s.
- **<disable_clean_session>**: set MQTT clean session. For more details about this parameter, please refer to the section Clean Session in *MQTT Version 3.1.1*.
  - 0: enable clean session.
  - 1: disable clean session.
- **<lwt_topic>**: LWT (Last Will and Testament) message topic. Maximum length: 128 bytes.
- **<lwt_msg>**: LWT message. Maximum length: 64 bytes.
- **<lwt_qos>**: LWT QoS, which can be set to 0, 1, or 2. Default: 0.
- **<lwt_retain>**: LWT retain, which can be set to 0 or 1. Default: 0.

3.6.6 AT+MQTTCONN: Connect to MQTT Brokers

Query Command

Function:

Query the MQTT broker that ESP devices are connected to.

Command:

```
AT+MQTTCONN?
```

Response:

```
+MQTTCONN:<LinkID>,<state>,<scheme><"host">,<port>,<"path">,<reconnect>
OK
```

Set Command

Function:

Connect to an MQTT broker.

Command:

```
AT+MQTTCONN=<LinkID>,<"host">,<port>,<reconnect>
```

Response:

```
OK
```

Parameters

- **<LinkID>**: only supports link ID 0 currently.
- **<host>**: MQTT broker domain. Maximum length: 128 bytes.
- **<port>**: MQTT broker port. Maximum: port 65535.
- **<path>**: path. Maximum length: 32 bytes.
- **<reconnect>**:
  - 0: MQTT will not reconnect automatically.
  - 1: MQTT will reconnect automatically. It takes more resources.
- **<state>**: MQTT state.
  - 0: MQTT uninitialized.
  - 1: already set `AT+MQTTUSERCFG`.
  - 2: already set `AT+MQTTCONNCFG`.
  - 3: connection disconnected.
  - 4: connection established.
  - 5: connected, but did not subscribe to any topic.
  - 6: connected, and subscribed to MQTT topics.
- **<scheme>**:
  - 1: MQTT over TCP.
  - 2: MQTT over TLS (no certificate verify).
  - 3: MQTT over TLS (verify server certificate).

  - 4: MQTT over TLS (provide client certificate).
  - 5: MQTT over TLS (verify server certificate and provide client certificate).
  - 6: MQTT over WebSocket (based on TCP).
  - 7: MQTT over WebSocket Secure (based on TLS, verify no certificate).
  - 8: MQTT over WebSocket Secure (based on TLS, verify server certificate).
  - 9: MQTT over WebSocket Secure (based on TLS, provide client certificate).
  - 10: MQTT over WebSocket Secure (based on TLS, verify server certificate and provide client certificate).

3.6.7 AT+MQTTPUB: Publish MQTT Messages in String

Set Command

Function:

Publish MQTT messages in string to a defined topic. If you need to publish messages in binary, please use the *AT+MQTTPUBRAW* command.

Command:

```
AT+MQTTPUB=<LinkID>,<"topic">,<"data">,<qos>,<retain>
```

Response:

```
OK
```

Parameters

- **<LinkID>**: only supports link ID 0 currently.
- **<topic>**: MQTT topic. Maximum length: 128 bytes.
- **<data>**: MQTT message in string.
- **<qos>**: QoS of message, which can be set to 0, 1, or 2. Default: 0.
- **<retain>**: retain flag.

Notes

- The length of the entire AT command should be less than 256 bytes.
- This command cannot send data `\0`. If you need to send `\0`, please use the command *AT+MQTTPUBRAW* instead.

3.6.8 AT+MQTTPUBRAW: Publish MQTT Messages in Binary

Set Command

Function:

Publish MQTT messages in binary to a defined topic.

Command:

```
AT+MQTTPUBRAW=<LinkID>,<"topic">,<length>,<qos>,<retain>
```

Response:

```
OK
>
```

The symbol > indicates that AT is ready for receiving serial data, and you can enter the data now. When the requirement of message length determined by the parameter `<length>` is met, the transmission starts.

If the transmission is successful, AT returns:

```
+MQTTPUB:OK
```

Otherwise, it returns:

```
+MQTTPUB:FAIL
```

Parameters

- **<LinkID>**: only supports link ID 0 currently.
- **<topic>**: MQTT topic. Maximum length: 128 bytes.
- **<length>**: length of MQTT message. The maximum length varies on different ESP devices.
  - For ESP32 devices, the maximum length is limited by available memory.
  - For ESP8266 devices, the maximum is limited by available memory and the marco `MQTT_BUFFER_SIZE_BYTE`. The default value of the macro is `512`. You can change the max length limitation by setting it in `build.py menuconfig`. `MQTT_BUFFER_SIZE_BYTE` equals maximum published data length plus the MQTT header length (depends on topic name length).
- **<qos>**: QoS of the published message, which can be set to 0, 1, or 2. Default is 0.
- **<retain>**: retain flag.

3.6.9 AT+MQTTSUB: Subscribe to MQTT Topics

Query Command

Function:

List all MQTT topics that have been already subscribed.

Command:

```
AT+MQTTSUB?
```

Response:

```
+MQTTSUB:<LinkID>,<state>,<"topic1">,<qos>
+MQTTSUB:<LinkID>,<state>,<"topic2">,<qos>
+MQTTSUB:<LinkID>,<state>,<"topic3">,<qos>
...
OK
```

Set Command

Function:

Subscribe to defined MQTT topics with defined QoS. It supports subscribing to multiple topics.

Command:

```
AT+MQTTSUB=<LinkID>,<"topic">,<qos>
```

Response:

```
OK
```

When AT receives MQTT messages of the subscribed topic, it will prompt:

```
+MQTTSUBRECV:<LinkID>,<"topic">,<data_length>,data
```

If the topic has been subscribed before, it will prompt:

```
ALREADY SUBSCRIBE
```

Parameters

- **<LinkID>**: only supports link ID 0 currently.
- **<state>**: MQTT state.
  - 0: MQTT uninitialized.
  - 1: already set `AT+MQTTUSERCFG`.
  - 2: already set `AT+MQTTCONNCFG`.
  - 3: connection disconnected.
  - 4: connection established.
  - 5: connected, but subscribe to no topic.
  - 6: connected, and subscribed to MQTT topics.
- **<topic>**: the topic that is subscribed to.
- **<qos>**: the QoS that is subscribed to.

3.6.10 AT+MQTTUNSUB: Unsubscribe from MQTT Topics

Set Command

Function:

Unsubscribe the client from defined topics. This command can be called multiple times to unsubscribe from different topics.

Command:

```
AT+MQTTUNSUB=<LinkID>,<"topic">
```

Response:

```
OK
```

If the topic has not been subscribed, AT will prompt:

```
NO UNSUBSCRIBE

OK
```

Parameters

- **<LinkID>**: only supports link ID 0 currently.
- **<topic>**: MQTT topic. Maximum length: 128 bytes.

3.6.11 AT+MQTTCLEAN: Close MQTT Connections

Set Command

Function:

Close the MQTT connection and release the resource.

Command:

```
AT+MQTTCLEAN=<LinkID>
```

Response:

```
OK
```

Parameter

- **<LinkID>**: only supports link ID 0 currently

3.6.12 MQTT AT Error Codes

The MQTT Error code will be prompted as `ERR CODE:0x<%08x>`.

Error Type	Error Code
AT_MQTT_NO_CONFIGURED	0x6001
AT_MQTT_NOT_IN_CONFIGURED_STATE	0x6002
AT_MQTT_UNINITIATED_OR_ALREADY_CLEAN	0x6003
AT_MQTT_ALREADY_CONNECTED	0x6004
AT_MQTT_MALLOC_FAILED	0x6005
AT_MQTT_NULL_LINK	0x6006
AT_MQTT_NULL_PARAMTER	0x6007
AT_MQTT_PARAMETER_COUNTS_IS_WRONG	0x6008
AT_MQTT_TLS_CONFIG_ERROR	0x6009
AT_MQTT_PARAM_PREPARE_ERROR	0x600A
AT_MQTT_CLIENT_START_FAILED	0x600B

Continued on next page

Table 3 – continued from previous page

Error Type	Error Code
AT_MQTT_CLIENT_PUBLISH_FAILED	0x600C
AT_MQTT_CLIENT_SUBSCRIBE_FAILED	0x600D
AT_MQTT_CLIENT_UNSUBSCRIBE_FAILED	0x600E
AT_MQTT_CLIENT_DISCONNECT_FAILED	0x600F
AT_MQTT_LINK_ID_READ_FAILED	0x6010
AT_MQTT_LINK_ID_VALUE_IS_WRONG	0x6011
AT_MQTT_SCHEME_READ_FAILED	0x6012
AT_MQTT_SCHEME_VALUE_IS_WRONG	0x6013
AT_MQTT_CLIENT_ID_READ_FAILED	0x6014
AT_MQTT_CLIENT_ID_IS_NULL	0x6015
AT_MQTT_CLIENT_ID_IS_OVERLENGTH	0x6016
AT_MQTT_USERNAME_READ_FAILED	0x6017
AT_MQTT_USERNAME_IS_NULL	0x6018
AT_MQTT_USERNAME_IS_OVERLENGTH	0x6019
AT_MQTT_PASSWORD_READ_FAILED	0x601A
AT_MQTT_PASSWORD_IS_NULL	0x601B
AT_MQTT_PASSWORD_IS_OVERLENGTH	0x601C
AT_MQTT_CERT_KEY_ID_READ_FAILED	0x601D
AT_MQTT_CERT_KEY_ID_VALUE_IS_WRONG	0x601E
AT_MQTT_CA_ID_READ_FAILED	0x601F
AT_MQTT_CA_ID_VALUE_IS_WRONG	0x6020
AT_MQTT_CA_LENGTH_ERROR	0x6021
AT_MQTT_CA_READ_FAILED	0x6022
AT_MQTT_CERT_LENGTH_ERROR	0x6023
AT_MQTT_CERT_READ_FAILED	0x6024
AT_MQTT_KEY_LENGTH_ERROR	0x6025
AT_MQTT_KEY_READ_FAILED	0x6026
AT_MQTT_PATH_READ_FAILED	0x6027
AT_MQTT_PATH_IS_NULL	0x6028
AT_MQTT_PATH_IS_OVERLENGTH	0x6029
AT_MQTT_VERSION_READ_FAILED	0x602A
AT_MQTT_KEEPALIVE_READ_FAILED	0x602B
AT_MQTT_KEEPALIVE_IS_NULL	0x602C
AT_MQTT_KEEPALIVE_VALUE_IS_WRONG	0x602D
AT_MQTT_DISABLE_CLEAN_SESSION_READ_FAILED	0x602E
AT_MQTT_DISABLE_CLEAN_SESSION_VALUE_IS_WRONG	0x602F
AT_MQTT_LWT_TOPIC_READ_FAILED	0x6030
AT_MQTT_LWT_TOPIC_IS_NULL	0x6031
AT_MQTT_LWT_TOPIC_IS_OVERLENGTH	0x6032
AT_MQTT_LWT_MSG_READ_FAILED	0x6033
AT_MQTT_LWT_MSG_IS_NULL	0x6034
AT_MQTT_LWT_MSG_IS_OVERLENGTH	0x6035
AT_MQTT_LWT_QOS_READ_FAILED	0x6036
AT_MQTT_LWT_QOS_VALUE_IS_WRONG	0x6037
AT_MQTT_LWT_RETAIN_READ_FAILED	0x6038
AT_MQTT_LWT_RETAIN_VALUE_IS_WRONG	0x6039
AT_MQTT_HOST_READ_FAILED	0x603A
AT_MQTT_HOST_IS_NULL	0x603B
AT_MQTT_HOST_IS_OVERLENGTH	0x603C

Continued on next page

Table 3 – continued from previous page

Error Type	Error Code
AT_MQTT_PORT_READ_FAILED	0x603D
AT_MQTT_PORT_VALUE_IS_WRONG	0x603E
AT_MQTT_RECONNECT_READ_FAILED	0x603F
AT_MQTT_RECONNECT_VALUE_IS_WRONG	0x6040
AT_MQTT_TOPIC_READ_FAILED	0x6041
AT_MQTT_TOPIC_IS_NULL	0x6042
AT_MQTT_TOPIC_IS_OVERLENGTH	0x6043
AT_MQTT_DATA_READ_FAILED	0x6044
AT_MQTT_DATA_IS_NULL	0x6045
AT_MQTT_DATA_IS_OVERLENGTH	0x6046
AT_MQTT_QOS_READ_FAILED	0x6047
AT_MQTT_QOS_VALUE_IS_WRONG	0x6048
AT_MQTT_RETAIN_READ_FAILED	0x6049
AT_MQTT_RETAIN_VALUE_IS_WRONG	0x604A
AT_MQTT_PUBLISH_LENGTH_READ_FAILED	0x604B
AT_MQTT_PUBLISH_LENGTH_VALUE_IS_WRONG	0x604C
AT_MQTT_RECV_LENGTH_IS_WRONG	0x604D
AT_MQTT_CREATE_SEMA_FAILED	0x604E
AT_MQTT_CREATE_EVENT_GROUP_FAILED	0x604F
AT_MQTT_URI_PARSE_FAILED	0x6050
AT_MQTT_IN_DISCONNECTED_STATE	0x6051
AT_MQTT_HOSTNAME_VERIFY_FAILED	0x6052

3.6.13 MQTT AT Notes

- In general, AT MQTT commands responds within 10 s, except the command `AT+MQTTCONN`. For example, if the router fails to access the Internet, the command `AT+MQTTPUB` will respond within 10 s. But the command `AT+MQTTCONN` may need more time due to packet retransmission in a bad network environment.
- If the `AT+MQTTCONN` is based on a TLS connection, the timeout of each packet is 10 s, and the total timeout will be much longer depending on the handshake packets count.
- When the MQTT connection ends, it will prompt the message `+MQTTDISCONNECTED:<LinkID>`.
- When the MQTT connection established, it will prompt the message `+MQTTCONNECTED:<LinkID>,<scheme>,<"host">,port,<"path">,<reconnect>`.

3.7 HTTP AT Commands

[]

- *AT+HTTPCLIENT*: Send HTTP Client Request
- *AT+HTTPGETSIZE*: Get HTTP Resource Size
- *AT+HTTPCPOST*: Post HTTP data of specified length
- *HTTP AT Error Codes*

3.7.1 AT+HTTPCLIENT: Send HTTP Client Request

Set Command

Command:

```
AT+HTTPCLIENT=<opt>,<content-type>,<"url">,[<"host">],[<"path">],<transport_type>[,<
↪"data">][,<"http_req_header">][,<"http_req_header">][...]
```

Response:

```
+HTTPCLIENT:<size>,<data>

OK
```

Parameters

- **<opt>**: method of HTTP client request.
  - 1: HEAD
  - 2: GET
  - 3: POST
  - 4: PUT
  - 5: DELETE
- **<content-type>**: data type of HTTP client request.
  - 0: `application/x-www-form-urlencoded`
  - 1: `application/json`
  - 2: `multipart/form-data`
  - 3: `text/xml`
- **<"url">**: HTTP URL. The parameter can override the `<host>` and `<path>` parameters if they are null.
- **<"host">**: domain name or IP address.
- **<"path">**: HTTP Path.
- **<transport_type>**: HTTP Client transport type. Default: 1.
  - 1: `HTTP_TRANSPORT_OVER_TCP`
  - 2: `HTTP_TRANSPORT_OVER_SSL`
- **<"data">**: If `<opt>` is a POST request, this parameter holds the data you send to the HTTP server. If not, this parameter does not exist, which means there is no need to input a comma to indicate this parameter.
- **<http_req_header>**: you can send more than one request header to the server.

Notes

- If the `url` parameter is not null, HTTP client will use it and ignore the `host` parameter and `path` parameter; If the `url` parameter is omited or null string, HTTP client will use `host` parameter and `path` parameter.
- In some released firmware, HTTP client commands are not supported (see *How to understand the differences of each type of module*), but you can enable it by `./build.py menuconfig` > `Component config` > `AT` > `AT http command support` and build the project (see *Build Your Own ESP-AT Project*).

Example

```
// HEAD Request
AT+HTTPCLIENT=1,0,"http://httpbin.org/get","httpbin.org","/get",1

// GET Request
AT+HTTPCLIENT=2,0,"http://httpbin.org/get","httpbin.org","/get",1

// POST Request
AT+HTTPCLIENT=3,0,"http://httpbin.org/post","httpbin.org","/post",1,"field1=value1&
↪field2=value2"
```

3.7.2 AT+HTTPGETSIZE: Get HTTP Resource Size

Set Command

Command:

```
AT+HTTPGETSIZE=<url>
```

Response:

```
+HTTPGETSIZE:<size>

OK
```

Parameters

- **<url>**: HTTP URL.
- **<size>**: HTTP resource size.

Note

- In some released firmware, HTTP client commands are not supported (see *How to understand the differences of each type of module*), but you can enable it by `./build.py menuconfig` > `Component config` > `AT` > `AT http command support` and build the project (see *Build Your Own ESP-AT Project*).

Example

```
AT+HTTPGETSIZE="http://www.baidu.com/img/bdlogo.gif"
```

3.7.3 AT+HTTPCPOST: Post HTTP data of specified length

Set Command

Command:

```
AT+HTTPCPOST=<url>,<length>[,<http_req_header_cnt>][,<http_req_header>..<http_req_
↪header>]
```

Response:

```
OK
>
```

The symbol > indicates that AT is ready for receiving serial data, and you can enter the data now. When the requirement of message length determined by the parameter `<length>` is met, the transmission starts.

If the transmission is successful, AT returns:

```
SEND OK
```

Otherwise, it returns:

```
SEND FAIL
```

Parameters

- **<url>**: HTTP URL.
- **<length>**: HTTP data length to POST. The maximum length is equal to the system allocable heap size.
- **<http_req_header_cnt>**: the number of <http_req_header> parameters.
- **[<http_req_header>]**: you can send more than one request header to the server.

3.7.4 HTTP AT Error Codes

- HTTP Client:

HTTP Client Error Code	Description
0x7190	Bad Request
0x7191	Unauthorized
0x7192	Payment Required
0x7193	Forbidden
0x7194	Not Found
0x7195	Method Not Allowed
0x7196	Not Acceptable
0x7197	Proxy Authentication Required
0x7198	Request Timeout
0x7199	Conflict
0x719a	Gone
0x719b	Length Required
0x719c	Precondition Failed
0x719d	Request Entity Too Large
0x719e	Request-URI Too Long
0x719f	Unsupported Media Type
0x71a0	Requested Range Not Satisfiable
0x71a1	Expectation Failed

- HTTP Server:

HTTP Server Error Code	Description
0x71f4	Internal Server Error
0x71f5	Not Implemented
0x71f6	Bad Gateway
0x71f7	Service Unavailable
0x71f8	Gateway Timeout
0x71f9	HTTP Version Not Supported

- HTTP AT:
  - The error code of command `AT+HTTPCLIENT` will be `0x7000+Standard HTTP Error Code` (For more details about Standard HTTP/1.1 Error Code, see RFC 2616).
  - For example, if AT gets the HTTP error 404 when calling command `AT+HTTPCLIENT`, it will respond with error code of `0x7194` (`hex(0x7000+404)=0x7194`).

3.8 [ESP32 Only] Ethernet AT Commands

[]

- *Prerequisite*
- [ESP32 Only] *AT+CIPETHMAC*: Query/Set the MAC address of the ESP Ethernet.
- [ESP32 Only] *AT+CIPETH*: Query/Set the IP address of the ESP Ethernet.

3.8.1 Prerequisite

Before you run any Ethernet AT Commands, please make the following preparations:

Note: This prerequisite takes ESP32-Ethernet-Kit as an example. If you use other modules or development boards, please refer to corresponding datasheets for RX/TX pins.

- Change AT UART pins (because default AT UART pins are in conflict with the Ethernet function pins):
  - Open factory_param_data.csv file.
  - In the row of module `WROVER-32`, change `uart_tx_pin` from GPIO22 to GPIO2, `uart_rx_pin` from GPIO19 to GPIO4, `uart_cts_pin` from GPIO15 to GPIO1, and `uart_rts_pin` from GPIO14 to GPIO1 (flow control is optional and is not used here). See *How to Set AT Port Pins* for more information.
- Enable `AT ethernet support`. See *How to enable ESP-AT Ethernet* for more information.
- Compile and flash the project onto ESP32-Ethernet-Kit.
- Connect your hardware:
  - Connect Host MCU (PC with USB to serial converter) to GPIO2 (TX) and GPIO4 (RX) of ESP32-Ethernet-Kit when the flow control function is not enabled.
  - Connect ESP32-Ethernet-Kit with Ethernet network.

3.8.2 [ESP32 Only] AT+CIPETHMAC: Query/Set the MAC Address of the ESP Ethernet

Query Command

Function:

Query the MAC address of the ESP Ethernet.

Command:

```
AT+CIPETHMAC?
```

Response:

```
+CIPETHMAC:<"mac">
OK
```

Set Command

Function:

Set the MAC address of the ESP Ethernet.

Command:

```
AT+CIPETHMAC=<"mac">
```

Response:

```
OK
```

Parameter

- **<"mac">**: string parameter showing the MAC address of the Ethernet interface.

Notes

- The default firmware does not support Ethernet AT commands (see *How to understand the differences of each type of module*), but you can enable it by `./build.py menuconfig` > `Component config` > `AT` > `AT ethernet support` and compile the project (see *Build Your Own ESP-AT Project*).
- The configuration changes will be saved in the NVS area if *AT+SYSSTORE=1*.
- Please make sure the MAC address of Ethernet interface you set is different from those of other interfaces.
- Bit0 of the ESP MAC address CANNOT be 1. For example, a MAC address can be "1a:. . . " but not "15:. . . ".
- `FF:FF:FF:FF:FF:FF` and `00:00:00:00:00:00` are invalid MAC addresses and cannot be set.

Example

```
AT+CIPETHMAC="1a:fe:35:98:d4:7b"
```

3.8.3 [ESP32 Only] AT+CIPETH: Query/Set the IP Address of the the ESP Ethernet

Query Command

Function:

Query the IP address of the ESP Ethernet.

Command:

```
AT+CIPETH?
```

Response:

```
+CIPETH:ip:<ip>
+CIPETH:gateway:<gateway>
+CIPETH:netmask:<netmask>
OK
```

Set Command

Function:

Set the IP address of the ESP Ethernet.

Command:

```
AT+CIPETH=<ip>[,<gateway>,<netmask>]
```

Response:

```
OK
```

Parameters

- **<ip>**: string parameter showing the IP address of the ESP Ethernet.
- **[<gateway>]**: gateway.
- **[<netmask>]**: netmask.

Notes

- The default firmware does not support Ethernet AT commands (see *How to understand the differences of each type of module*), but you can enable it by `./build.py menuconfig` > `Component config` > `AT` > `AT ethernet support` and compile the project (see *Build Your Own ESP-AT Project*).
- The configuration changes will be saved in the NVS area if *AT+SYSSTORE=1*.
- This Set Command correlates with DHCP commands, such as *AT+CWDHCP*:
  - If static IP is enabled, DHCP will be disabled.
  - If DHCP is enabled, static IP will be disabled.
  - The last configuration overwrites the previous configuration.

Example

```
AT+CIPETH="192.168.6.100","192.168.6.1","255.255.255.0"
```

3.9 [ESP8266 Only] Signaling Test AT Commands

[]

- *AT+FACTPLCP*: Send with long or short PLCP (Physical Layer Convergence Procedure)

3.9.1 [ESP8266 Only] AT+FACTPLCP: Send with Long or Short PLCP

Set Command

Command:

```
AT+FACTPLCP=<enable>,<tx_with_long>
```

Response:

```
OK
```

Parameters

- **<enable>**: Enable or disable manual configuration.
  - 0: Disable manual configuration. The default value for the parameter `<tx_with_long>` will be used.
  - 1: Enable manual configuration. The type of PLCP that AT sends depends on `<tx_with_long>`.
- **<tx_with_long>**: Send with long PLCP or short PLCP.
  - 0: Send with short PLCP (default).
  - 1: Send with long PLCP.

3.10 Web server AT Commands

- *AT+WEBSERVER*: Enable/disable Wi-Fi connection configuration via web server.

3.10.1 AT+WEBSERVER: Enable/disable Wi-Fi connection configuration via web server

Set Command

Command:

```
AT+WEBSERVER=<enable>,<server_port>,<connection_timeout>
```

Response:

```
OK
```

Parameters

- **<enable>**: Enable or disable web server.
  - 0: Disable the web server and release related resources.
  - 1: Enable web server, which means that you can use WeChat or a browser to configure Wi-Fi connection information.
- **<server_port>**: The web server port number.
- **<connection_timeout>**: The timeout for the every connection. Unit: second. Range:[21,60].

Notes

- There are two ways to provide the HTML files needed by the web server. One is to use fatfs file system (non ESP8266 chips), and you need to enable AT FS command at this time. The other one is to use embedded files to storge HTML files (default setting).
- Please make sure that the max number of open sockets is not less than 12, you may change the number by `./build.py menuconfig` > `Component config` > `LWIP` > `Max bumber of open sockets` and compile the project (see *Build Your Own ESP-AT Project*).
- The default firmware does not support web server AT commands (see *How to understand the differences of each type of module*), but you can enable it by `./build.py menuconfig` > `Component config` > `AT` > `AT WEB Server command support` and compile the project (see *Build Your Own ESP-AT Project*).
- For ESP8266 devices, you may need to turn off some unnecessary options to store the necessary html files.
- For more examples, please refer to *Web Server AT Example*.

Example

```
// Enable the web server with port 80, and the timeout for the every connection is 50␣
↪seconds:
AT+WEBSERVER=1,80,50

// Disable the web server, the command should be:
AT+WEBSERVER=0
```

3.11 [ESP32 & ESP32-S2 & ESP32-C3] Driver AT Commands

[]

- *AT+DRVADC*: Read ADC channel value.
- *AT+DRVPWMINIT*: Initialize PWM driver.
- *AT+DRVPWMDUTY*: Set PWM duty.
- *AT+DRVPWMFADE*: Set PWM fade.
- *AT+DRVI2CINIT*: Initialize I2C master driver.

- *AT+DRVI2CRD*: Read I2C data.
- *AT+DRVI2CWRDATA*: Write I2C data.
- *AT+DRVI2CWRBYTES*: Write no more than 4 bytes I2C data.
- *AT+DRVSPICONFGPIO*: Configure SPI GPIO.
- *AT+DRVSPIINIT*: Initialize SPI master driver.
- *AT+DRVSPIRD*: Read SPI data.
- *AT+DRVSPIWR*: Write SPI data.

3.11.1 AT+DRVADC: Read ADC Channel Value

Set Command

Command:

```
AT+DRVADC=<channel>,<atten>
```

Response:

```
+DRVADC:<raw data>
OK
```

Parameters

- **<channel>**: ADC1 channel. Range: 0 ~ 7. See datasheets for corresponding pins.
- **<atten>**: attenuation.
  - 0: 0 dB attenuation gives full-scale voltage 1.1 V.
  - 1: 2.5 dB attenuation gives full-scale voltage 1.5 V.
  - 2: 6 dB attenuation gives full-scale voltage 2.2 V.
  - 3: 11 dB attenuation gives full-scale voltage 3.9 V.
- **<raw data>**: ADC channel value. The voltage value is raw_data/2 ^{width} * atten.

Notes

- ESP-AT only supports ADC1.
- ESP32 and ESP32-C3 support 12-bit width, and ESP32-S2 only supports 13-bit width.

Example

```
AT+DRVADC=0,0    // ADC1 0 channel, voltage: 0 ~ 1.1 V
+DRVADC:2048     // For ESP32 and ESP32-C3, the voltage is 2048 / 4096 * 1.1 = 0.55
                 // For ESP32-S2, the voltage is 2048 / 8192 * 1.1 = 0.264
OK
```

3.11.2 AT+DRVPWMINIT: Initialize PWM Driver

Set Command

Command:

```
AT+DRVPWMINIT=<freq>,<duty_res>,<ch0_gpio>[,...,<ch3_gpio>]
```

Response:

```
OK
```

Parameters

- **<freq>**: LEDC timer frequency. Unit: Hz. Range: 1 Hz ~ 8 MHz.
- **<duty_res>**: LEDC channel duty resolution. Range: 0 ~ 20 bits.
- **<chx_gpio>**: LEDC output GPIO number of channel x. For example, if you want to use GPIO16 as channel 0, set <ch0_gpio> to 16.

Notes

- AT can support a maximum of 4 channels.
- The number of channels that you initialize using this command will determine how many channels you can set using other PWM commands, including *AT+DRVPWMDUTY* and *AT+DRVPWMFADE*. For example, if you initialize two channels, you can only change the two channels' PWM duty using command `AT+DRVPWMDUTY`.
- The frequency and the duty resolution are interdependent. See Supported Range of Frequency and Duty Resolutions for more details.

Example

```
AT+DRVPWMINIT=5000,13,17,16,18,19  // set 4 channels; frequency: 5 kHz; duty␣
↪resolution: 13 bits
AT+DRVPWMINIT=10000,10,17          // only use channel 0, frequency: 10 kHz; duty␣
↪resolution: 10 bits; other PMW commands can only set one channel
```

3.11.3 AT+DRVPWMDUTY: Set PWM Duty

Set Command

Command:

```
AT+DRVPWMDUTY=<ch0_duty>[,...,<ch3_duty>]
```

Response:

```
OK
```

Parameter

- **<duty>**: LEDC channel duty. Range: [0,2 $^{duty\_resolution}$].

Notes

- AT can support a maximum of 4 channels.
- If you do not want to set `<duty>` for a specific channel, just omit it.

Example

```
AT+DRVPWMDUTY=255,512   // set channel 0 to duty 255, set channel 1 to duty 512
AT+DRVPWMDUTY=,,0       // set channel 2 to duty 0
```

3.11.4 AT+DRVPWMFADE: Set PWM Fade

Set Command

Command:

```
AT+DRVPWMFADE=<ch0_target_duty>,<ch0_fade_time>[,...,<ch3_target_duty>,<ch3_fade_time>
↪]
```

Response:

```
OK
```

Parameters

- **<target_duty>**: target duty of fading. Range: [0, 2 $^{duty\_resolution}$–1].
- **<fade_time>**: the maximum time of fading. Unit: millisecond.

Notes

- AT can support a maximum of 4 channels.
- If you do not want to set `<target_duty>` and `<fade_time>` for a specific channel, just omit them.

Example

```
AT+DRVPWMFADE=,,0,1000           // use one second to change channel 1 duty to 0
AT+DRVPWMFADE=1024,1000,0,2000,  // use one second time to change channel 0 duty to␣
↪1024, two seconds to change channel 1 duty to 0
```

3.11.5 AT+DRVI2CINIT: Initialize I2C Master Driver

Set Command

Command:

```
AT+DRVI2CINIT=<num>,<scl_io>,<sda_io>,<clock>
```

Response:

```
OK
```

Parameters

- **<num>**: I2C port number. Range: 0 ~ 1. If the following parameters are not set, AT will deinitialize the I2C port.
- **<scl_io>**: GPIO number for I2C SCL signal.
- **<sda_io>**: GPIO number for I2C SDA signal.
- **<clock>**: I2C clock frequency for master mode. Unit: Hz. Maximum: 1 MHz.

Note

- This command only supports I2C masters.

Example

```
AT+DRVI2CINIT=0,25,26,1000  // initialize I2C0; GPIO25 is SCL; GPIO26 is SDA; I2C␣
↪clock is 1 kHz
AT+DRVI2CINIT=0             // deinitialize I2C0
```

3.11.6 AT+DRVI2CRD: Read I2C Data

Set Command

Command:

```
AT+DRVI2CRD=<num>,<address>,<length>
```

Response:

```
+DRVI2CRD:<read data>
OK
```

Parameters

- **<num>**: I2C port number. Range: 0 ~ 1.
- **<address>**: I2C slave device address.

  - 7-bit address: 0 ~ 0x7F.
  - 10-bit address: The first seven bits of the first byte are the combination 1111 0XX of which the last two bits (XX) are the two Most Significant Bits (MSBs) of the 10-bit address. For example, if the 10-bit address is 0x2FF (b'1011111111), the input address should be 0x7AFF (b'111101011111111).
- **<length>**: I2C data length. Range: 1 ~ 2048.
- **<read data>**: I2C data.

Note

- I2C transmission timeout is one second.

Example

```
AT+DRVI2CRD=0,0x34,1      // I2C0 reads one byte data from address 0x34
AT+DRVI2CRD=0,0x7AFF,1    // I2C0 reads one byte data from 10-bit address 0x2FF

// I2C0 reads address 0x34, register address 0x27, read 2 bytes
AT+DRVI2CWRBYTES=0,0x34,1,0x27      // I2C0 first writes device address 0x34, register␣
↪address 0x27
AT+DRVI2CRD=0,0x34,2                // I2C0 reads 2 bytes
```

3.11.7 AT+DRVI2CWRDATA: Write I2C Data

Set Command

Command:

```
AT+DRVI2CWRDATA=<num>,<address>,<length>
```

Response:

```
OK
>
```

This response indicates that you should enter the data you want to write. When the requirement of data length is met, the data transmission starts.

If the data is transmitted successfully, AT returns:

```
OK
```

If the data transmission fails, AT returns:

```
ERROR
```

Parameters

- **<num>**: I2C port number. Range: 0 ~ 1.
- **<address>**: I2C slave device address.

  - 7-bit address: 0 ~ 0x7F.
  - 10-bit address: The first seven bits of the first byte are the combination 1111 0XX of which the last two bits (XX) are the two Most Significant Bits (MSBs) of the 10-bit address. For example, if the 10-bit address is 0x2FF (b'1011111111), the input address should be 0x7AFF (b'111101011111111).
- **<length>**: I2C data length. Range: 1 ~ 2048.

Note

- I2C transmission timeout is one second.

Example

```
AT+DRVI2CWRDATA=0,0x34,10   // I2C0 writes 10 bytes data to address 0x34
```

3.11.8 AT+DRVI2CWRBYTES: Write No More Than 4 Bytes I2C Data

Set Command

Command:

```
AT+DRVI2CWRBYTES=<num>,<address>,<length>,<data>
```

Response:

```
OK
```

Parameters

- **<num>**: I2C port number. Range: 0 ~ 1.
- **<address>**: I2C slave device address.
  - 7-bit address: 0 ~ 0x7F.
  - 10-bit address: The first seven bits of the first byte are the combination 1111 0XX of which the last two bits (XX) are the two Most Significant Bits (MSBs) of the 10-bit address. For example, if the 10-bit address is 0x2FF (b'1011111111), the input address should be 0x7AFF (b'111101011111111).
- **<length>**: the length of the I2C data you want to write. Range: 1 ~ 4 bytes.
- **<data>**: the data of `<length>` long. Range: 0 ~ 0xFFFFFFFF.

Note

- I2C transmission timeout is one second.

Example

```
AT+DRVI2CWRBYTES=0,0x34,2,0x1234     // I2C0 writes 2 bytes data 0x1234 to address␣
↪0x34
AT+DRVI2CWRBYTES=0,0x7AFF,2,0x1234   // I2C0 writes 2 bytes data 0x1234 to 10-bit␣
↪address 0x2FF

// I2C0 writes address 0x34; register address: 0x27; data: c0xFF
AT+DRVI2CWRBYTES=0,0x34,2,0x27FF
```

3.11.9 AT+DRVSPICONFGPIO: Configure SPI GPIO

Set Command

Command:

```
AT+DRVSPICONFGPIO=<mosi>,<miso>,<sclk>,<cs>
```

Response:

```
OK
```

Parameters

- **<mosi>**: GPIO pin for Master Out Slave In signal.
- **<miso>**: GPIO pin for Master In Slave Out signal, or -1 if not used.
- **<sclk>**: GPIO pin for SPI Clock signal.
- **<cs>**: GPIO pin for slave selection signal, or -1 if not used.

3.11.10 AT+DRVSPIINIT: Initialize SPI Master Driver

Set Command

Command:

```
AT+DRVSPIINIT=<clock>,<mode>,<cmd_bit>,<addr_bit>,<dma_chan>[,bits_msb]
```

Response:

```
OK
```

Parameters

- **<clock>**: Clock speed, divisors of 80 MHz. Unit: Hz. Maximum: 40 MHz.
- **<mode>**: SPI mode. Range: 0 ~ 3.
- **<cmd_bit>**: Default amount of bits in command phase. Range: 0 ~ 16.
- **<addr_bit>**: Default amount of bits in address phase. Range: 0 ~ 64.
- **<dma_chan>**: Either channel 1 or 2, or 0 in the case when no DMA is required.

- **<bits_msb>**: SPI data format:
  - Bit0:
    - 0: Transmit MSB first (default).
    - 1: Transmit LSB first.
  - Bit1:
    - 0: Receive data MSB first (default).
    - 1: Receive data LSB first.

Note

- You should configure SPI GPIO before SPI initialization.

Example

```
AT+DRVSPIINIT=102400,0,0,0,0,3 // SPI clock: 100 kHz; mode: 0; both command and␣
↪address bits are 0; not use DMA; transmit and receive LSB first
OK
AT+DRVSPIINIT=0   // delete SPI Driver
OK
```

3.11.11 AT+DRVSPIRD: Read SPI Data

Set Command

Command:

```
AT+DRVSPIRD=<data_len>[,<cmd>,<cmd_len>][,<addr>,<addr_len>]
```

Response:

```
+DRVSPIRD:<read data>
OK
```

Parameters

- **<data_len>**: length of SPI data you want to read. Range: 1 ~ 4092 bytes.
- **<cmd>**: command data. The length of the data is set in `<cmd_len>`.
- **<cmd_len>**: command length in this transaction. Range: 0 ~ 2 bytes.
- **<addr>**: command address. The length of the address is set in `<addr_len>`.
- **<addr_len>**: The address length in this transaction. Range: 0 ~ 4 bytes.

Note

- If you don't use DMA, the maximum `<data_len>` you can set is 64 bytes each time.

Example

```
AT+DRVSPIRD=2  // read 2 bytes data
+DRVI2CREAD:ffff
OK

AT+DRVSPIRD=2,0x03,1,0x001000,3  // read 2 bytes data; <cmd> is 0x03; <cmd_len> is 1␣
↪byte; <addr> is 0x1000; <addr_len> is 3 bytes
+DRVI2CREAD:ffff
OK
```

3.11.12 AT+DRVSPIWR: Write SPI Data

Set Command

Command:

```
AT+DRVSPIWR=<data_len>[,<cmd>,<cmd_len>][,<addr>,<addr_len>]
```

Response:

When `<data_len>` is larger than 0, AT returns:

```
OK
>
```

This response indicates that you should enter the data you want to write. When the requirement of data length is met, the data transmission starts.

If the data is transmitted successfully, AT returns:

```
OK
```

When `<data_len>` is equal to 0, which means AT transmits commands and addresses only, and no SPI data, AT returns:

```
OK
```

Parameters

- **<data_len>**: SPI data length. Range: 0 ~ 4092.
- **<cmd>**: command data. The length of the data is set in `<cmd_len>`.
- **<cmd_len>**: command length in this transaction. Range: 0 ~ 2 bytes.
- **<addr>**: command address. The length of the address is set in `<addr_len>`.
- **<addr_len>**: The address length in this transaction. Range: 0 ~ 4 bytes.

Note

- If you don't use DMA, the maximum `<data_len>` you can set is 64 bytes each time.

Example

```
AT+DRVSPIWR=2  // write 2 bytes data
OK
>              // begin receiving serial data
OK

AT+DRVSPIWR=0,0x03,1,0x001000,3  // write 0 byte data; <cmd> is 0x03; <cmd_len> is 1 byte; <addr> is 0x1000; <addr_len> is 3 bytes
OK
```

3.12 AT Command Set Comparison

[]

This document lists the differences between the AT commands supported by the old NONOS-AT version and those by the new ESP-AT version that you should pay attention to before migrating from the old to new.

- NONOS-AT: **Not recommended** version for ESP8266 series of chips since its base SDK, ESP8266_NONOS_SDK, is no longer updated.
- ESP-AT: **Recommended** version. It supports all series of chips and a richer set of commands compared with NONOS-AT, such as Bluetooth commands, Bluetooth Low Energy commands, Ethernet commands, driver commands, and so on. For more details about ESP-AT, please refer to *What is ESP-AT* and *AT Binary Lists*.

Note: The table below covers all the commands supported by the NONOS-AT, but not all by ESP-AT. See Section *AT Command Set* for a full list of commands supported by ESP-AT.

The links in the first column of the table point to ESP-AT commands.

Table 4: AT Command Set Comparison

Command	Function	NONOS-AT	ESP-AT
AT	Test AT startup		
AT+RST	Restart a module		
AT+GMR	Check version information		
AT+GSLP	Enter Deep-sleep mode		
ATE	AT commands echoing		
AT+RESTORE	Restore factory default settings		
AT+UART_CUR	Current UART configuration, not saved in flash		
AT+UART_DEF	Default UART configuration, saved in flash		
AT+SLEEP	Set the sleep mode	1: light sleep 2: modem sleep	ESP8266 ESP32 ESP32-S2 Note[1]

Continued on next page

Table 4 – continued from previous page

Command	Function	NONOS-AT	ESP-AT
AT+WAKEUPGPIO	Wakeup from light sleep on GPIO interrupt		[3] Similar command: *AT+SLEEPWKCFG*
AT+RFPOWER	Set RF TX Power	Range: [0,82] Unit: 0.25 dBm	[2]
AT+RFVDD	Set RF TX power according to VDD33		
AT+SYSRAM	Query the remaining space of RAM		Added the parameter of minimum heap size
AT+SYSADC	Read ADC value		
AT+SYSIOSETCFG	Set IO working mode		
AT+SYSIOGETCFG	Query IO working mode		
AT+SYSGPIODIR	Set GPIO direction		
AT+SYSGPIOWRITE	Set GPIO output level		
AT+SYSGPIOREAD	Read GPIO input level		
AT+SYSMSG	Set system prompt info		
AT+SYSMSG_CUR	Set system prompt info, not saved in flash		[3] Similar command: *AT+SYSMSG*
AT+SYSMSG_DEF	Set default system prompt info, saved in flash		[3] Similar command: *AT+SYSMSG*
AT+CWMODE	Set Wi-Fi mode		Added automatic connection after mode switching
AT+CWMODE_CUR	Set current Wi-Fi mode, not saved in flash		[3] Similar command: *AT+CWMODE*
AT+CWMODE_DEF	Set default Wi-Fi mode, saved in flash		[3] Similar command: *AT+CWMODE*
AT+CWJAP	Connect to an AP		Added more functionality
AT+CWJAP_CUR	Connect to an AP, not saved in flash		[3] Similar command: *AT+CWJAP*
AT+CWJAP_DEF	Connect to an AP, saved in flash		[3] Similar command: *AT+CWJAP*
AT+CWLAPOPT	Set configuration for *AT+CWLAP*		Added more functionality

Continued on next page

Table 4 – continued from previous page

Command	Function	NONOS-AT	ESP-AT
AT+CWLAP	List available APs		Response is different
AT+CWQAP	Disconnect from an AP		
AT+CWSAP	Set softAP parameters		
AT+CWSAP_CUR	Set softAP parameters, not saved in flash		[3] Similar command: *AT+CWSAP*
AT+CWSAP_DEF	Set softAP parameters, saved in flash		[3] Similar command: *AT+CWSAP*
AT+CWLIF	Query info of the station that connects to a softAP		
AT+CWDHCP	Set DHCP		
AT+CWDHCP_CUR	Set DHCP, not saved in flash		[3] Similar command: *AT+CWDHCP*
AT+CWDHCP_DEF	Set DHCP, saved in flash		[3] Similar command: *AT+CWDHCP*
AT+CWDHCPS	Set the IP addresses allocated by an ESP softAP DHCP server		
AT+CWDHCPS_CUR	Same as above, but not saved in flash		[3] Similar command: *AT+CWDHCPS*
AT+CWDHCPS_DEF	Same as above, but saved in flash		[3] Similar command: *AT+CWDHCPS*
AT+CWAUTOCONN	Connect to an AP automatically when powered on		
AT+CIPSTAMAC	Set the MAC address of an ESP station		
AT+CIPSTAMAC_CUR	Same as above, but not saved in flash		[3] Similar command: *AT+CIPSTAMAC*
AT+CIPSTAMAC_DEF	Same as above, but saved in flash		[3] Similar command: *AT+CIPSTAMAC*
AT+CIPAPMAC	Set the IP address of an ESP softAP		
AT+CIPAPMAC_CUR	Same as above, but not saved in flash		[3] Similar command: *AT+CIPAPMAC*

Continued on next page

Table 4 – continued from previous page

Command	Function	NONOS-AT	ESP-AT
AT+CIPAPMAC_DEF	Same as above, but saved in flash		[3] Similar command: *AT+CIPAPMAC*
AT+CIPSTA	Set the IP address of an ESP station		
AT+CIPSTA_CUR	Same as above, but not saved in flash		[3] Similar command: *AT+CIPSTA*
AT+CIPSTA_DEF	Same as above, but saved in flash		[3] Similar command: *AT+CIPSTA*
AT+CIPAP	Set the IP address of an ESP softAP		
AT+CIPAP_CUR	Same as above, but not saved in flash		[3] Similar command: *AT+CIPAP*
AT+CIPAP_DEF	Same as above, but saved in flash		[3] Similar command: *AT+CIPAP*
AT+CWSTARTSMART	Start SmartConfig		
AT+CWSTOPSMART	Stop SmartConfig		
AT+CWSTARTDISCOVER	Enable the mode that an ESP device can be found by WeChat		
AT+CWSTOPDISCOVER	Disable the mode that an ESP device can be found by WeChat		
AT+WPS	Enable the WPS function		
AT+MDNS	Configure the mDNS function		
AT+CWJEAP	Connect to a WPA2 Enterprise AP		ESP8266 ESP32 ESP32-S2
AT+CWJEAP_CUR	Same as above, but not saved in flash		
AT+CWJEAP_DEF	Same as above, but saved in flash		
AT+CWHOSTNAME	Set the host name of an ESP station		
AT+CWCOUNTRY	Set the Wi-Fi Country Code		
AT+CWCOUNTRY_CUR	Same as above, but not saved in flash		[3] Similar command: *AT+CWCOUNTRY*

Continued on next page

Table 4 – continued from previous page

Command	Function	NONOS-AT	ESP-AT
AT+CWCOUNTRY_DEF	Same as above, but saved in flash		[3] Similar command: *AT+CWCOUNTRY*
AT+CIPSTATUS	Obtain the TCP/UDP/SSL connection status and info		
AT+CIPDOMAIN	Resolve a domain name		
AT+CIPSTART	Establish a TCP/UDP/SSL connection		
AT+CIPSSLSIZE	Set SSL buffer size		
AT+CIPSSLCCONF	Set SSL clients		Parameters are different
AT+CIPSEND	Send data		
AT+CIPSENDEX	Send data of specified length, or use `\0` to trigger data transmission		
AT+CIPSENDBUF	Write data into the TCP-Send-Buffer		[3]
AT+CIPBUFRESET	Reset the segment ID count		[3]
AT+CIPBUFSTATUS	Query the status of the TCP-Send-Buffer		[3]
AT+CIPCHECKSEQ	Query if a specific segment was successfully sent		[3]
AT+CIPCLOSEMODE	Set the close mode of TCP connection		[3]
AT+CIPCLOSE	Close TCP/UDP/SSL connection		
AT+CIFSR	Obtain the local IP address		
AT+CIPMUX	Set multiple connections		
AT+CIPSERVER	Create a TCP/SSL server	SSL server not supported	SSL server supported for ESP32 and ESP32-S2, not for ESP8266
AT+CIPSERVERMAXCONN	Set the maximum connections allowed by a server		
AT+CIPMODE	Set the transmission mode		
AT+SAVETRANSLINK	Set whether to enter Wi-Fi *Passthrough Mode* on power-up		
AT+CIPSTO	Set the local TCP server timeout		
AT+PING	Ping the remote host		
AT+CIUPDATE	Upgrade the firmware through Wi-Fi		More parameters supported
AT+CIPDINFO	Show the remote IP and port with +IPD		
AT+CIPRECVMODE	Set TCP Receive Mode		
AT+CIPRECVDATA	Get TCP data in passive receive mode		Response is different
AT+CIPRECVLEN	Get TCP data length in passive receive mode		
AT+CIPSNTPCFG	Set the time zone and SNTP server		Added more functionality
AT+CIPSNTPTIME	Query SNTP time		

Continued on next page

Table 4 – continued from previous page

Command	Function	NONOS-AT	ESP-AT
AT+CIPDNS	Set DNS server information		
AT+CIPDNS_CUR	Same as above, but not saved in flash		[3] Similar command: *AT+CIPDNS*
AT+CIPDNS_DEF	Same as above, but saved in flash		[3] Similar command: *AT+CIPDNS*
AT+SYSFLASH	Set user partitions in flash		

Tip: Click the footnote number to jump (back) to where it is marked in the table above.

It is strongly recommended to read the following sections for some common information on AT commands before you dive into the details of each command.

- *AT Command Types*
- *AT Commands with Configuration Saved in the Flash*
- *AT Messages*

3.13 AT Command Types

Generic AT command has four types:

Type	Command Format	Description
Test Command	AT+<CommamdName>=?	Query the Set Commands' internal parameters and their range of values.
Query Command	AT+<CommamdName>?	Return the current value of parameters.
Set Command	AT+<CommamdName>=<...>	Set the value of user-defined parameters in commands, and run these commands.
Execute Command	AT+<CommamdName>	Run commands with no user-defined parameters.

- Not all AT commands support all of the four types mentioned above.
- Currently, only strings and integer numbers are supported as input parameters in AT commands.

[1] AT+SLEEP in ESP-AT:
- ESP8266 and ESP32
  - 1: modem sleep by DTIM
  - 2: light sleep
  - 3: modem sleep by listen interval
- ESP32-S2

[3] This command will not be added to the ESP-AT version.
[2] AT+RFPOWER in ESP-AT:
- ESP8266 . Range: [40,82]. Unit: 0.25 dBm
- ESP32 . Range: [40,78]. Unit: 0.25 dBm. Support Bluetooth LE.
- ESP32-S2 . Range: [40,78]. Unit: 0.25 dBm

- Angle brackets < > designate parameters that can not be omitted.
- Square brackets [ ] designate optional parameters that can be omitted. The default value of the parameter will be used instead when you omit it. Below are examples of entering the command *AT+CWJAP* with some parameters omitted.

```
AT+CWJAP="ssid","password"
AT+CWJAP="ssid","password","11:22:33:44:55:66"
```

- If the parameter you want to omit is followed by a parameter(s), you must give a `,` to indicate it.

```
AT+CWJAP="ssid","password",,1
```

- String values need to be included in double quotation marks.

```
AT+CWSAP="ESP756290","21030826",1,4
```

- Escape character syntax is needed if a string contains special characters, such as `,`, `"`, or `\`:
  - `\\`: escape the backslash itself
  - `\,`: escape comma which is not used to separate each parameter
  - `\"`: escape double quotation mark which is not used to mark string input
  - `\<any>`: escape `<any>` character means that drop backslash symbol and only use `<any>` character
- Escape is needed in AT commands only, not elsewhere. For example, when AT command port prints > and wait for your input, the input does not need to be escaped.

```
AT+CWJAP="comma\,backslash\\ssid","1234567890"
AT+MQTTPUB=0,"topic","\"{\"sensor\":012}\"",1,0
```

- The default baud rate of AT command is 115200.
- The length of each AT command should be no more than 256 bytes.
- AT commands end with a new-line (CR-LF), so the serial tool should be set to "New Line Mode".
- Definitions of AT command error codes are provided in *AT API Reference*:
  - *esp_at_error_code*
  - *esp_at_para_parse_result_type*
  - *esp_at_result_code_string_index*

3.14 AT Commands with Configuration Saved in the Flash

Configuration settings entered by the following AT Commands will always be saved in the flash NVS Area, so they can be automatically restored on reset:

- *AT+UART_DEF*: `AT+UART_DEF=115200,8,1,0,3`
- *AT+SAVETRANSLINK*: `AT+SAVETRANSLINK=1,"192.168.6.10",1001`
- *AT+CWAUTOCONN*: `AT+CWAUTOCONN=1`

Saving of configuration settings by several other commands can be switched on or off with *AT+SYSSTORE* command, which is mentioned in the Note section of these commands.

3.15 AT Messages

There are two types of ESP-AT messages returned from the ESP-AT command port:

- ESP-AT Response Messages (passive)

  Each ESP-AT command input returns response messages to tell the sender the result of the ESP-AT command.

Table 5: ESP-AT Response Messages

AT Response Messages	Description
OK	AT command process done and return OK
ERROR	AT command error or error occurred during the execution
SEND OK	Data has been sent to the protocol stack (specific to *AT+CIPSEND* and *AT+CIPSENDEX* command). It doesn't mean that the data has been sent to the opposite end
SEND FAIL	Error occurred during sending the data to the protocol stack (specific to *AT+CIPSEND* and *AT+CIPSENDEX* command
+<Command Name>: . . .	Response to the sender that describes AT command process results in details

- ESP-AT Message Reports (active)

  ESP-AT will report important state changes or messages in the system.

Table 6: ESP-AT Message Reports

ESP-AT Message Report	Description
ready	The ESP-AT firmware is ready
busy p...	Busy processing. The system is in process of handling the previous command, thus CANNOT accept the new input
ERR CODE:`<0x%08x>`	Error code for different commands
Will force to restart!!!	Module restart right now
smartconfig type:`<xxx>`	Smartconfig type
Smart get wifi info	Smartconfig has got the SSID and PASSWORD information
smartconfig connected wifi	Smartconfig done. ESP-AT has connected to the Wi-Fi
WIFI CONNECTED	Wi-Fi station interface has connected to an AP
WIFI GOT IP	Wi-Fi station interface has got the IPv4 address
WIFI GOT IPv6 LL	Wi-Fi station interface has got the IPv6 LinkLocal address
WIFI GOT IPv6 GL	Wi-Fi station interface has got the IPv6 Global address
WIFI DISCONNECT	Wi-Fi station interface has disconnected from an AP
+ETH_CONNECTED	Ethernet interface has connected
+ETH_GOT_IP	Ethernet interface has got the IPv4 address
+ETH_DISCONNECTED	Ethernet interface has disconnected
[<conn_id>,]CONNECT	A network connection of which ID is `<conn_id>` is established (ID=0 by default)
[<conn_id>,]CLOSED	A network connection of which ID is `<conn_id>` ends (ID=0 by default)
+LINK_CONN	Detailed connection information of TCP/UDP/SSL

Continued on next page

Table 6 – continued from previous page

ESP-AT Message Report	Description
+STA_CONNECTED: <sta_mac>	A station has connected to the Wi-Fi softAP interface of ESP-AT
+DIST_STA_IP: <sta_mac>,<sta_ip>	The Wi-Fi softAP interface of ESP-AT distributes an IP address to the station
+STA_DISCONNECTED: <sta_mac>	A station disconnected from the Wi-Fi softAP interface of ESP-AT
>	ESP-AT is waiting for more data to be received
Recv `<xxx>` bytes	ESP-AT has already received `<xxx>` bytes from the ESP-AT command port
+IPD	ESP-AT received the data from the network
SEND Canceled	Cancel to send in *Wi-Fi normal sending mode*
Have `<xxx>` Connections	Has reached the maximum connection counts for server
+QUITT	ESP-AT quits from the Wi-Fi *Passthrough Mode*
NO CERT FOUND	No valid device certificate found in custom partition
NO PRVT_KEY FOUND	No valid private key found in custom partition
NO CA FOUND	No valid CA certificate found in custom partition
+MQTTCONNECTED	MQTT connected to the broker
+MQTTDISCONNECTED	MQTT disconnected from the broker
+MQTTSUBRECV	MQTT received the data from the broker
+MQTTPUB:FAIL	MQTT failed to publish data
+MQTTPUB:OK	MQTT publish data done
+BLECONN	A Bluetooth LE connection established
+BLEDISCONN	A Bluetooth LE connection ends
+READ	A read operation from Bluetooth LE connection
+WRITE	A write operation from Bluetooth LE connection
+NOTIFY	A notification from Bluetooth LE connection
+INDICATE	An indication from Bluetooth LE connection
+BLESECNTFYKEY	Bluetooth LE SMP key
+BLESECREQ:<conn_index>	Received encryption request which index is <conn_index>
+BLEAUTHCMPL:<conn_index>,<enc_result>	Bluetooth LE SMP pairing completed

CHAPTER 4

AT Command Examples

[]

4.1 TCP/IP AT Examples

- Example 1. *ESP as a TCP Client in Single Connection*
- Example 2. *ESP as a TCP Server in Multiple Connections*
- Example 3. *UDP Transmission*
- Example 4. *UART-Wi-Fi Passthrough Transmission*

4.1.1 Example 1. ESP as a TCP Client in Single Connection

1. Set the Wi-Fi mode:

```
Command:
AT+CWMODE=3                    // SoftAP+Station mode

Response:
OK
```

2. Connect to the router:

```
AT+CWJAP="SSID","password"               // SSID and password of router

Response:
OK
```

3. Query the device's IP:

```
AT+CIFSR

Response:
192.168.3.106                          // device got an IP from router
```

4. Connect the PC to the same router which ESP is connected to. Use a network tool on the PC to create a TCP server. For example, the TCP server on PC is 192.168.3.116, port 8080.
5. ESP is connected to the TCP server as a client:

```
AT+CIPSTART="TCP","192.168.3.116",8080   // protocolserver IP & port
```

6. Send data:

```
AT+CIPSEND=4                 // set date length which will be sent, such as 4 bytes
>TEST                         // enter the data, no CR

Response:
SEND OK
```

   Note If the number of bytes inputted are more than the length (n) set by `AT+CIPSEND`, the system will reply `busy`, and send the first n bytes. And after sending the first n bytes, the system will reply `SEND OK`.

7. Receive data:

```
+IPD,n:xxxxxxxxxx                  // received n bytes, data=xxxxxxxxxxx
```

4.1.2 Example 2. ESP as a TCP Server in Multiple Connections

When ESP works as a TCP server, multiple connections should be enabled; that is to say, there should be more than one client connecting to ESP.Below is an example showing how a TCP server is established when ESP works in the SoftAP mode. If ESP works as a Station, set up a server in the same way after connecting ESP to the router.

1. Set the Wi-Fi mode:

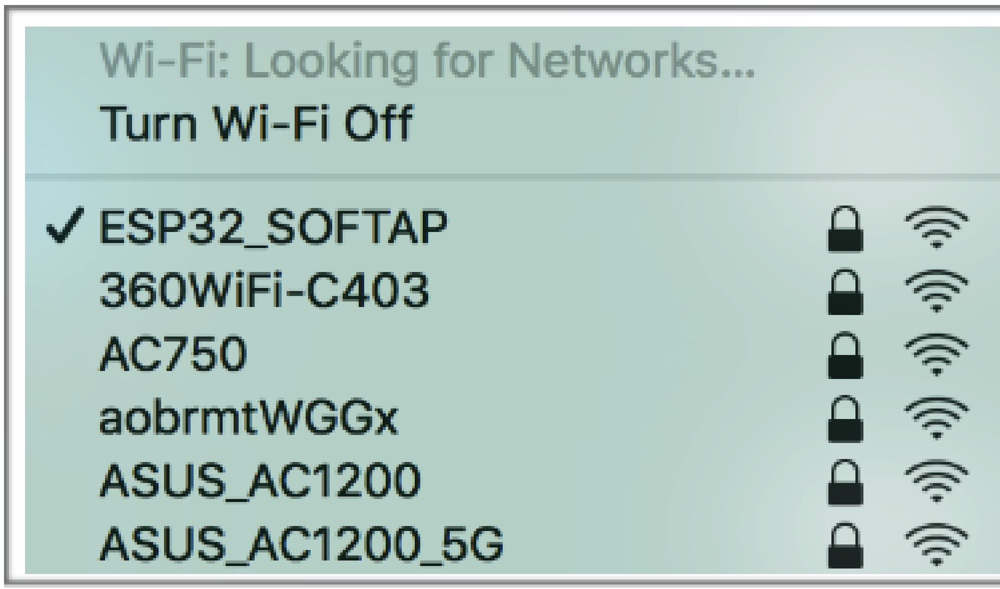
```
Command:
AT+CWMODE=3                      // SoftAP+Station mode

Response:
OK
```

2. Enable multiple connections.

```
AT+CIPMUX=1

Response:
OK
```

3. Set up a TCP server.

```
AT+CIPSERVER=1                         // default port = 333

Response:
OK
```



4. Connect the PC to the ESP SoftAP.
5. Using a network tool on PC to create a TCP client and connect to the TCP server that ESP created.**Notice**:When ESP works as a TCP server, there is a timeout mechanism. If the TCP client is connected to the ESP TCP server, while there is no data transmission for a period of time, the server will disconnect from the client. To avoid such a problem, please set up a data transmission cycle every two seconds.
6. Send data:

```
// ID number of the first connection is defaulted to be 0
AT+CIPSEND=0,4              // send 4 bytes to connection NO.0
>TEST                       // enter the data, no CR

Response:
SEND OK
```

Note If the number of bytes inputted are more than the length (n) set by `AT+CIPSEND`, the system will reply `busy`, and send the first n bytes. And after sending the first n bytes, the system will reply `SEND OK`.

7. Receive data:

```
+IPD,0,n:xxxxxxxxxx                    // received n bytes, data=xxxxxxxxxxx
```

8. Close the TCP connection.

```
AT+CIPCLOSE=0

Response:
0,CLOSED
OK
```

4.1.3 Example 3. UDP Transmission

1. Set the Wi-Fi mode:

```
Command:
AT+CWMODE=3                    // SoftAP+Station mode
```

(continues on next page)

(continued from previous page)

```
Response:
OK
```

2. Connect to the router:

```
AT+CWJAP="SSID","password"                 // SSID and password of router

Response:
OK
```

3. Query the device's IP:

```
AT+CIFSR

Response:
+CIFSR:STAIP,"192.168.101.104"              // device got an IP from router
```

4. Connect the PC to the same router which ESP is connected to. Use a network tool on the PC to create UDP transmission. For example, the PC's IP address is 192.168.101.116, port 8080.

5. Below are two examples of UDP transmission.

Example 3.1. UDP Transmission with Fixed Remote IP and Port

In UDP transmission, whether the remote IP and port are fixed or not is determined by the last parameter of `AT+CIPSTART`. 0 means that the remote IP and port are fixed and cannot be changed. A specific ID is given to such a connection, ensuring that the data sender and receiver will not be replaced by other devices.

1. Enable multiple connections:

```
AT+CIPMUX=1

Response:
OK
```

2. Create a UDP transmission, with the ID being 4, for example.

```
AT+CIPSTART=4,"UDP","192.168.101.110",8080,1112,0

Response:
4,CONNECT
OK
```

Notes:

- `"192.168.101.110"` and `8080` are the remote IP and port of UDP transmission on the remote side, i.e., the UDP configuration set by PC.
- `1112` is the local port number of ESP. Users can define this port number. A random port will be used if this parameter is not set.
- `0` means that the remote IP and port are fixed and cannot be changed. For example, if another PC also creates a UDP entity and sends data to ESP port 1112, ESP can receive the data sent from UDP port 1112. But when data are sent using AT command `AT+CIPSEND=4,X`, it will still be sent to the first PC end. If parameter `0` is not used, the data will be sent to the new PC.

3. Send data:

```
AT+CIPSEND=4,7                // send 7 bytes to transmission NO.4
>UDPtest                      // enter the data, no CR

Response:
SEND OK
```

Note If the number of bytes inputted are more than the length (n) set by `AT+CIPSEND`, the system will reply `busy`, and send the first n bytes. And after sending the first n bytes, the system will reply `SEND OK`.

4. Receive data:

```
+IPD,4,n:xxxxxxxxxx                // received n bytes, data=xxxxxxxxxxx
```

5. Close UDP transmission No.4

```
AT+CIPCLOSE=4

Response:
4,CLOSED
OK
```

Example 3.2. UDP Transmission with Changeable Remote IP and Port

1. Create a UDP transmission with the last parameter being 2.

```
AT+CIPSTART="UDP","192.168.101.110",8080,1112,2

Response:
CONNECT
OK
```

Notes:

- `"192.168.101.110"` and `8080` here refer to the IP and port of the remote UDP transmission terminal which is created on a PC in above Example 2.
- `1112` is the local port of ESP. Users can define this port. A random port will be opened if this parameter is not set.
- `2` means the means the opposite terminal of UDP transmission can be changed. The remote IP and port will be automatically changed to those of the last UDP connection to ESP.

2. Send data:

```
AT+CIPSEND=7                  // send 7 bytes
>UDPtest                      // enter the data, no CR

Response:
SEND OK
```

Note If the number of bytes inputted are more than the length (n) set by `AT+CIPSEND`, the system will reply `busy`, and send the first n bytes. And after sending the first n bytes, the system will reply `SEND OK`.

3. If you want to send data to any other UDP terminals, please designate the IP and port of the target terminal in the command.

```
AT+CIPSEND=6,"192.168.101.111",1000           // send six bytes
>abcdef                                       // enter the data, no CR

Response:
SEND OK
```

4. Receive data:

```
+IPD,n:xxxxxxxxxx                // received n bytes, data=xxxxxxxxxxx
```

5. Close UDP transmission.

```
AT+CIPCLOSE

Response:
CLOSED
OK
```

4.1.4 Example 4. UART-Wi-Fi Passthrough Transmission

ESP-AT supports UART-Wi-Fi passthrough transmission only when ESP works as a TCP client in single connection or UDP transmission.

Example 4.1. ESP as a TCP Client in UART-Wi-Fi Passthrough (Single Connection Mode)

1. Set the Wi-Fi mode:

```
Command:
AT+CWMODE=3                  // SoftAP+Station mode

Response:
OK
```

2. Connect to the router:

```
AT+CWJAP="SSID","password"                  // SSID and password of router

Response:
OK
```

3. Query the device's IP:

```
AT+CIFSR

Response:
+CIFSR:STAIP,"192.168.101.105"                 // device got an IP from router
```

4. Connect the PC to the same router which ESP is connected to. Use a network tool on the PC to create a TCP server. For example, the PC's IP address is 192.168.101.110, port 8080.

5. Connect the ESP device to the TCP server as a TCP client.

```
AT+CIPSTART="TCP","192.168.101.110",8080
```

(continues on next page)

(continued from previous page)

```
Response:
CONNECT
OK
```

6. Enable the UART-WiFi transmission mode.

```
AT+CIPMODE=1

Response:
OK
```

7. Send data.

```
AT+CIPSEND

Response:
>                // From now on, data received from UART will be transparent␣
↪transmitted to server
```

8. Stop sending data.When receiving a packet that contains only +++, the UART-WiFi passthrough transmission process will be stopped. Then please wait at least 1 second before sending next AT command.Please be noted that if you input +++ directly by typing, the +++, may not be recognised as three consecutive + because of the Prolonged time when typing.**Notice**:The aim of ending the packet with **+++** is to exit transparent transmission and to accept normal AT commands, while TCP still remains connected. However, users can also deploy command `AT+CIPSEND` to go back into transparent transmission.

9. Exit the UART-WiFi passthrough mode.

```
AT+CIPMODE=0

Response:
OK
```

10. Close the TCP connection.

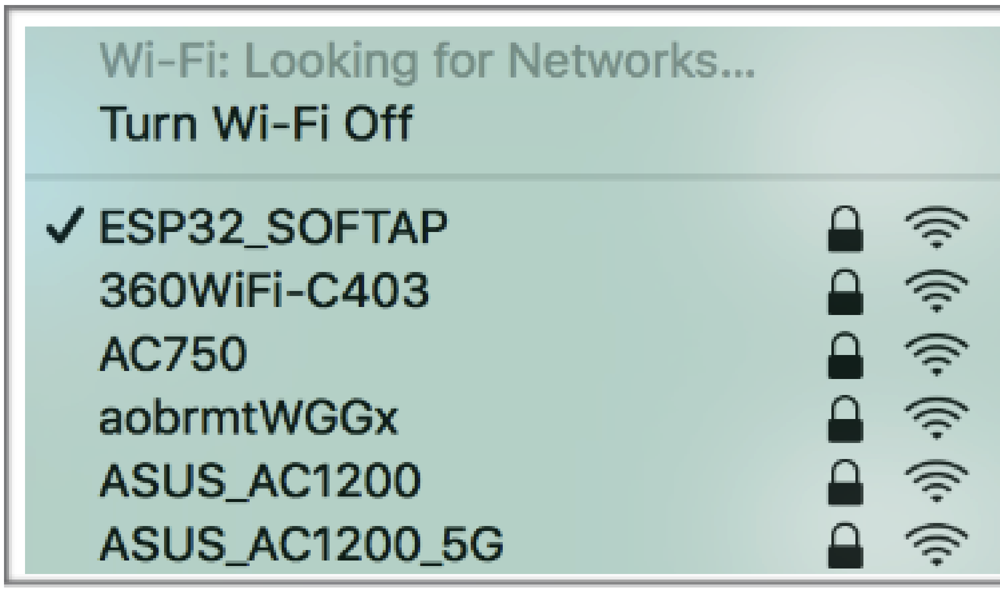
```
AT+CIPCLOSE

Response:
CLOSED
OK
```

Example 4.2. UDP Transmission in UART-Wi-Fi Passthrough Mode

Here is an example of the ESP working as a SoftAP in UDP transparent transmission.

1. Set the Wi-Fi mode:

```
Command:
AT+CWMODE=3                 // SoftAP+Station mode

Response:
OK
```



2. Connect the PC to the ESP SoftAP.
3. Use a network tool on PC to create a UDP endpoint. For example, the PC's IP address is `192.168.4.2` and the port is `1001`.
4. Create a UDP transmission between ESP32 and the PC with a fixed remote IP and port.

```
AT+CIPSTART="UDP","192.168.4.2",1001,2233,0

Response:
CONNECT
OK
```

5. Enable the UART-WiFi transmission mode.

```
AT+CIPMODE=1

Response:
OK
```

6. Send data.

```
AT+CIPSEND

Response:
>                // From now on, data received from UART will be transparent␣
↪transmitted to server
```

7. Stop sending data.When receiving a packet that contains only +++, the UART-WiFi passthrough transmission process will be stopped. Then please wait at least 1 second before sending next AT command.Please be noted that if you input +++ directly by typing, the +++, may not be recognised as three consecutive + because of the Prolonged time when typing.**Notice**:The aim of ending the packet with +++ is to exit transparent transmission and to accept normal AT commands, while TCP still remains connected. However, users can also deploy command `AT+CIPSEND` to go back into transparent transmission.
8. Exit the UART-WiFi passthrough mode.

```
AT+CIPMODE=0
```

(continues on next page)

(continued from previous page)

```
Response:
OK
```

9. Close the UDP transmission.

```
AT+CIPCLOSE

Response:
CLOSED
OK
```

4.2 [ESP32 Only] BLE AT Examples

- Example 1. [ESP32 Only] *BLE AT Examples*
- Example 2. [ESP32 Only] *iBeacon Examples*
- Example 3. [ESP32 Only] *UART-BLE Passthrough Mode*

4.2.1 Example 1. [ESP32 Only] BLE AT Example

Below is an example of using two ESP32 modules, one as a BLE server (hereafter named "ESP32 Server"), the other one as a BLE client (hereafter named "ESP32 Client"). The example shows how to use BLE functions with AT commands.***Notice:***

- The ESP32 Server needs to download a "service bin" into Flash to provide BLE services.
  - To learn how to generate a "service bin", please refer to esp-at/tools/readme.md.
  - The download address of the "service bin" is the address of "ble_data" in esp-at/partitions_at.csv.

1. BLE initialization:

   ESP32 Server:

   ```
   Command:
   AT+BLEINIT=2                              // server role

   Response:
   OK
   ```

   ESP32 Client:

   ```
   Command:
   AT+BLEINIT=1                              // client role

   Response:
   OK
   ```

2. Establish BLE connection:

   ESP32 Server:(1) Query the BLE address. For example, its address is "24:0a:c4:03:f4:d6".

```
Command:
AT+BLEADDR?                                 // get server's BLE address

Response:
+BLEADDR:24:0a:c4:03:f4:d6
OK
```

(2) Start advertising.

```
Command:
AT+BLEADVSTART

Response:
OK
```

ESP32 Client:(1) Start scanning.

```
Command:
AT+BLESCAN=1,3

Response:
+BLESCAN:<BLE address>,<rssi>,<adv_data>,<scan_rsp_data>
OK
```

(2) Establish the BLE connection, when the server is scanned successfully.

```
AT+BLECONN=0,"24:0a:c4:03:f4:d6"

Response:
OK
+BLECONN:0,"24:0a:c4:03:f4:d6"
```

Notes:

- If the BLE connection is established successfully, it will prompt `+BLECONN:<conn_index>,<remote_BLE_address>`
- If the BLE connection is broken, it will prompt `+BLEDISCONN:<conn_index>,<remote_BLE_address>`

3. Read/Write a characteristic:

   ESP32 Server:(1) Create services.

```
AT+BLEGATTSSRVCRE

Response:
OK
```

(2) Start services.

```
AT+BLEGATTSSRVSTART

Response:
OK
```

(3) Discover characteristics.

```
AT+BLEGATTSCHAR?

Response:
+BLEGATTSCHAR:"char",1,1,0xC300
+BLEGATTSCHAR:"desc",1,1,1
+BLEGATTSCHAR:"char",1,2,0xC301
+BLEGATTSCHAR:"desc",1,2,1
+BLEGATTSCHAR:"char",1,3,0xC302
+BLEGATTSCHAR:"desc",1,3,1
OK
```

ESP32 Client:(1) Discover services.

```
AT+BLEGATTCPRIMSRV=0

Response:
+BLEGATTCPRIMSRV:0,1,0x1801,1
+BLEGATTCPRIMSRV:0,2,0x1800,1
+BLEGATTCPRIMSRV:0,3,0xA002,1
OK
```

Notice:

- When discovering services, the ESP32 Client will get two more default services (UUID:0x1800 and 0x1801) than what the ESP32 Server will get.
- So, for the same service, the <srv_index> received by the ESP32 Client equals the <srv_index> received by ESP32 Server + 2.
- For example, the <srv_index> of the above-mentioned service, 0xA002, is 3 when the ESP32 Client is in the process of discovering services. But if the ESP32 Server tries to discover it with command `AT+BLEGATTSSRV?`, the <srv_index> will be 1.

(2) Discover characteristics.

```
AT+BLEGATTCCHAR=0,3

Response:
+BLEGATTCCHAR:"char",0,3,1,0xC300,2
+BLEGATTCCHAR:"desc",0,3,1,1,0x2901
+BLEGATTCCHAR:"char",0,3,2,0xC301,2
+BLEGATTCCHAR:"desc",0,3,2,1,0x2901
+BLEGATTCCHAR:"char",0,3,3,0xC302,8
+BLEGATTCCHAR:"desc",0,3,3,1,0x2901
+BLEGATTCCHAR:"char",0,3,4,0xC303,4
+BLEGATTCCHAR:"desc",0,3,4,1,0x2901
+BLEGATTCCHAR:"char",0,3,5,0xC304,8
+BLEGATTCCHAR:"char",0,3,6,0xC305,16
+BLEGATTCCHAR:"desc",0,3,6,1,0x2902
+BLEGATTCCHAR:"char",0,3,7,0xC306,32
+BLEGATTCCHAR:"desc",0,3,7,1,0x2902
OK
```

(3) Read a characteristic. Please note that the target characteristic's property has to support the read operation.

```
AT+BLEGATTCRD=0,3,1

Response:
```

(continues on next page)

(continued from previous page)

```
+BLEGATTCRD:0,1,30
OK
```

Note:

- If the ESP32 Client reads the characteristic successfully, message `+READ:<conn_index>,<remote BLE address>` will be prompted on the ESP32 Server side.

(4) Write a characteristic. Please note that the target characteristic's property has to support the write operation.

```
AT+BLEGATTCWR=0,3,3,,2

Response:
>       // waiting for data
OK
```

Note:

- If the ESP32 Client writes the characteristic successfully, message `+WRITE:<conn_index>,<srv_index>,<char_index>,[<desc_index>],<len>,<value>` will be prompted on the ESP32 Server side.

4. Notify of a characteristic:

   ESP32 Client:(1) Configure the characteristic's descriptor. Please note that the target characteristic's property has to support notifications.

```
AT+BLEGATTCWR=0,3,6,1,2

Response:
>       // waiting for data
OK
```

Note:

- If the ESP32 Client writes the descriptor successfully, message `+WRITE:<conn_index>,<srv_index>,<char_index>,<desc_index>,<len>,<value>` will be prompted on the ESP32 Server side.

ESP32 Server:(1) Notify of a characteristic. Please note that the target characteristic's property has to support notifications.

```
AT+BLEGATTSNTFY=0,1,6,3

Response:
>       // waiting for data
OK
```

Note:

- If the ESP32 Client receives the notification, it will prompt message `+NOTIFY:<conn_index>,<srv_index>,<char_index>,<len>,<value>`.
- For the same service, the <srv_index> on the ESP32 Client side equals the <srv_index> on the ESP32 Server side + 2.

5. Indicate a characteristic:

   ESP32 Client:(1) Configure the characteristic's descriptor. Please note that the target characteristic's property has to support the indicate operation.

```
AT+BLEGATTCWR=0,3,7,1,2

Response:
>       // waiting for data
OK
```

Note:

- If the ESP32 Client writes the descriptor successfully, message `+WRITE:<conn_index>,<srv_index>,<char_index>,<desc_index>,<len>,<value>` will be prompted on the ESP32 Server side.

ESP32 Server:(1) Indicate characteristic. Please note that the target characteristic's property has to support the indicate operation.

```
AT+BLEGATTSIND=0,1,7,3

Response:
>       // waiting for data
OK
```

Note:

- If the ESP32 Client receives the indication, it will prompt message `+INDICATE:<conn_index>,<srv_index>,<char_index>,<len>,<value>`
- For the same service, the <srv_index> on the ESP32 Client side equals the <srv_index> on the ESP32 Server side + 2.

4.2.2 Example 2. [ESP32 Only] iBeacon Examples

The following demonstrates two examples of iBeacon for ESP32 modules:

- ESP32 advertising iBeacons, which can be discovered by the "Shake Nearby" function of WeChat.
- ESP32 scanning iBeacons.

This is the structure of iBeacon Frame.

Example 2.1. ESP32 Device Advertising iBeacons

1. Initialize the role of the ESP32 device as a BLE server:

```
AT+BLEINIT=2                              // server role

Response
OK
```

2. Start advertising. Configure the parameters of the iBeacon advertisement as the following table shows:

The AT command should be as below:

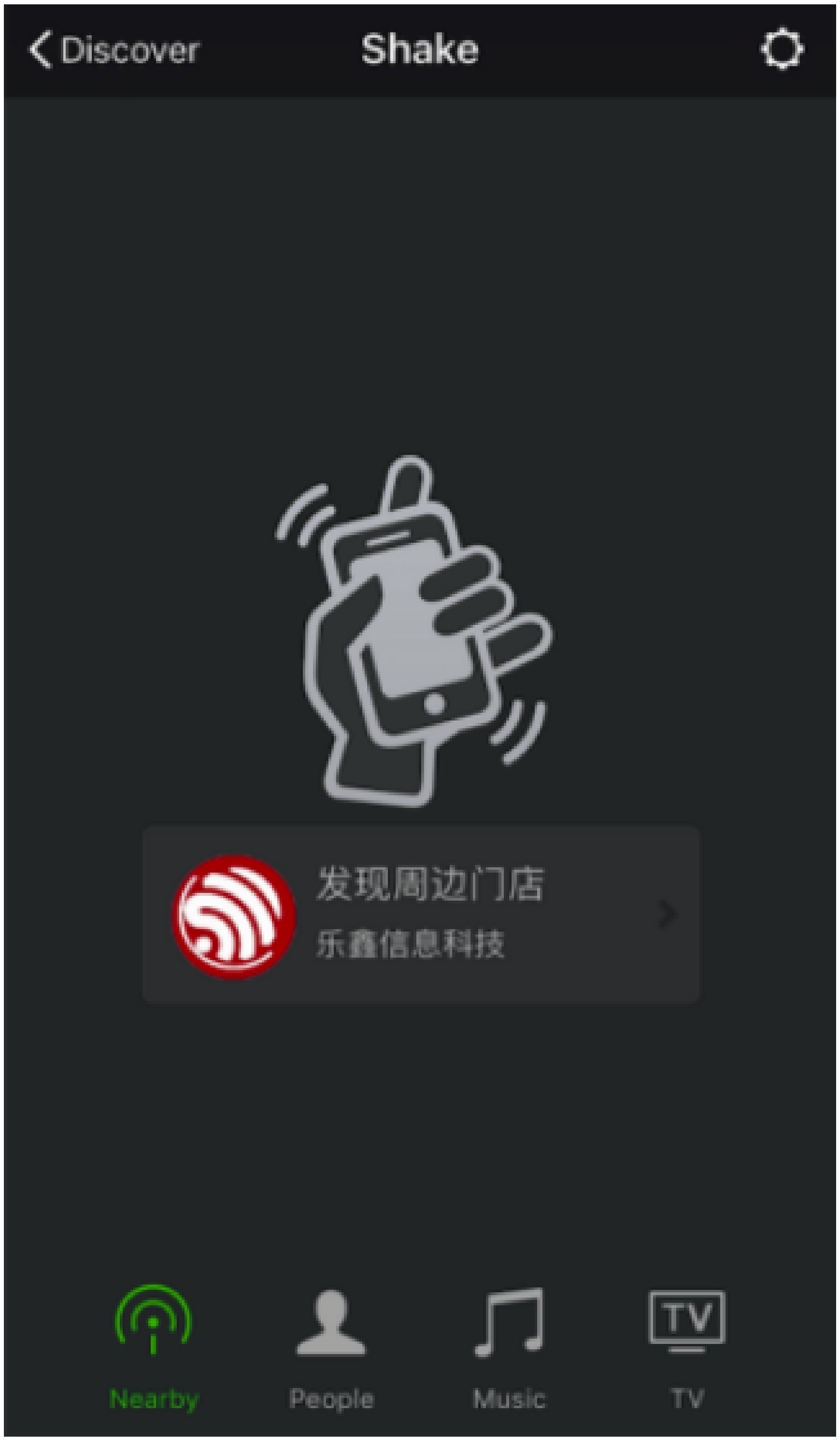
```
```
AT+BLEADVDATA="0201061aff4c000215fda50693a4e24fb1afcfc6eb0764782527b7f206c5"

OK

```

(continues on next page)
```

(continued from previous page)

```
AT+BLEADVSTART               // Start advertising

OK
```
```

Open WeChat on your mobile phone and then select "Shake Nearby" to discover the ESP32 device that is advertising.
```



avatar

Example 2.2. ESP32 Device Scanning for iBeacons

Not only can the ESP32 device transmits iBeacons, but it can also work as a BLE client that scans for iBeacons and gets the advertisement data which can then be parsed by the host MCU.**Notice:**If the ESP32 device has already been initialized as a BLE server, you need to call AT+BLEINIT=0 to de-init it first, and then re-init it as a BLE client.

1. Initialize the role of the ESP32 device as a BLE client:

```
AT+BLEINIT=1                                  // client role

Response
OK
```

2. Enable a scanning for three seconds:

```
AT+BLESCAN=1,3

Response
OK
```

You will get a scanning result that looks like:

```
```
+BLESCAN:24:0a:c4:02:10:0e,-33,
↪0201061aff4c000215fda50693a4e24fb1afcfc6eb0764782527b7f206c5,
+BLESCAN:24:0a:c4:01:4d:fe,-74,02010207097a4f68664b43020aeb051220004000,
+BLESCAN:24:0a:c4:02:10:0e,-33,
↪0201061aff4c000215fda50693a4e24fb1afcfc6eb0764782527b7f206c5,
```
```

4.2.3 Example 3. [ESP32 Only] UART-BLE Passthrough Mode

Below is an example of using two ESP32 modules, one as a BLE server (hereafter named "ESP32 Server"), the other one as a BLE client (hereafter named "ESP32 Client"). The example shows how to build BLE SPP (Serial Port Profile, UART-BLE passthrough mode) with AT commands. ***Notice:***

- The ESP32 Server needs to download a "service bin" into Flash to provide BLE services.
  - To learn how to generate a "service bin", please refer to esp-at/tools/readme.md.
  - The download address of the "service bin" is the address of "ble_data" in esp-at/partitions_at.csv.

1. BLE initialization:

   ESP32 Server:

```
AT+BLEINIT=2                                  // server role

OK

AT+BLEGATTSSRVCRE                             // Create services

OK

AT+BLEGATTSSRVSTART                           // Start services

OK
```

   ESP32 Client:

```
AT+BLEINIT=1                                  // client role

OK
```

2. Establish BLE connection:

   ESP32 Server:(1) Query the BLE address. For example, its address is “24:0a:c4:03:f4:d6”.

```
Command:
AT+BLEADDR?                                 // get server's BLE address

Response:
+BLEADDR:24:0a:c4:03:f4:d6

OK
```

   (2) Optional Configuration, configure advertisement data. Without the configuration, the payload of the broadcasting packet will be empty.

```
Command:
AT+BLEADVDATA="0201060A09457370726573736966030302A0"

/*  The adv data is
 *  02 01 06  //<length>,<type>,<data>
 *  0A 09 457370726573736966 //<length>,<type>,<data>
 *  03 03 02A0  //<length>,<type>,<data>
 */

Response:
OK
```

   (3) Start advertising.

```
Command:
AT+BLEADVSTART

Response:
OK
```

   ESP32 Client:(1) Start scanning.

```
Command:
AT+BLESCAN=1,3

Response:
+BLESCAN:<BLE address>,<rssi>,<adv_data>,<scan_rsp_data>

OK
```

   (2) Establish the BLE connection, when the server is scanned successfully.

```
AT+BLECONN=0,"24:0a:c4:03:f4:d6"

Response:
OK
+BLECONN:0,"24:0a:c4:03:f4:d6"
```

   Notes:

   - If the BLE connection is established successfully, it will prompt `+BLECONN:<conn_index>, <remote_BLE_address>`
   - If the BLE connection is broken, it will prompt `+BLEDISCONN:<conn_index>, <remote_BLE_address>`

3. Discover services.

   ESP32 Server:(1) Discover local services.

```
AT+BLEGATTSSRV?

Response:
+BLEGATTSSRV:1,1,0xA002,1

OK
```

(2) Discover characteristics.

```
AT+BLEGATTSCHAR?

Response:
+BLEGATTSCHAR:"char",1,1,0xC300
+BLEGATTSCHAR:"desc",1,1,1
+BLEGATTSCHAR:"char",1,2,0xC301
+BLEGATTSCHAR:"desc",1,2,1
+BLEGATTSCHAR:"char",1,3,0xC302
+BLEGATTSCHAR:"desc",1,3,1

OK
```

ESP32 Client:(1) Discover services.

```
AT+BLEGATTCPRIMSRV=0

Response:
+BLEGATTCPRIMSRV:0,1,0x1801,1
+BLEGATTCPRIMSRV:0,2,0x1800,1
+BLEGATTCPRIMSRV:0,3,0xA002,1

OK
```

Notice:

- When discovering services, the ESP32 Client will get two more default services (UUID:0x1800 and 0x1801) than what the ESP32 Server will get.
- So, for the same service, the <srv_index> received by the ESP32 Client equals the <srv_index> received by ESP32 Server + 2.
- For example, the <srv_index> of the above-mentioned service, 0xA002, is 3 when the ESP32 Client is in the process of discovering services. But if the ESP32 Server tries to discover it with command `AT+BLEGATTSSRV?`, the <srv_index> will be 1.

(2) Discover characteristics.

```
AT+BLEGATTCCHAR=0,3

Response:
+BLEGATTCCHAR:"char",0,3,1,0xC300,2
+BLEGATTCCHAR:"desc",0,3,1,1,0x2901
+BLEGATTCCHAR:"char",0,3,2,0xC301,2
+BLEGATTCCHAR:"desc",0,3,2,1,0x2901
+BLEGATTCCHAR:"char",0,3,3,0xC302,8
+BLEGATTCCHAR:"desc",0,3,3,1,0x2901
```

(continues on next page)

(continued from previous page)

```
+BLEGATTCCHAR:"char",0,3,4,0xC303,4
+BLEGATTCCHAR:"desc",0,3,4,1,0x2901
+BLEGATTCCHAR:"char",0,3,5,0xC304,8
+BLEGATTCCHAR:"char",0,3,6,0xC305,16
+BLEGATTCCHAR:"desc",0,3,6,1,0x2902
+BLEGATTCCHAR:"char",0,3,7,0xC306,32
+BLEGATTCCHAR:"desc",0,3,7,1,0x2902

OK
```

4. Configure BLE SPP:

   ESP32 Client:(1) Set a characteristic that enables writing permission to TX channel for sending data. Set another characteristic that supports notification or indication to RX channel for receiving data.

```
AT+BLESPPCFG=1,3,5,3,7

Response:
OK
```

   (2) Enable BLE SPP:

```
AT+BLESPP

Response:
OK
>                          // waiting for serial data
```

Note: After ESP32 Client enabling BLE SPP, data received from serial port will be transmitted to the BLE server directly.

```
ESP32 Server:
(1) Set a characteristic that supports notification or indication to TX channel for␣
↪sending data. Set another characteristic that enables writing permission to RX␣
↪channel for receiving data.

    AT+BLESPPCFG=1,1,7,1,5

    Response:
    OK
(2) Enable BLE SPP:

    AT+BLESPP

    Response:
    OK
    >                         // waiting for serial data
```

Notes:

- After ESP32 Server enables BLE SPP, the data received from serial port will be transmitted to the BLE client directly.
- If the ESP32 Client does not enable BLE SPP first, or uses other device as BLE client, then the BLE client needs to listen to the notification or indication first. For example, if the ESP32 Client does not enable BLE SPP first, then it should enable listening with command `AT+BLEGATTCWR=0,3,7,1,1` first for the ESP32 Server to transmit successfully.

- For the same service, the `<srv_index>` on the ESP32 Client side equals the `<srv_index>` on the ESP32 Server side plus 2.

4.3 MQTT AT Examples

- *Example 1: MQTT over TCP*
- *Example 2: MQTT over TLS*
- *Example 3: MQTT over WSS*

4.3.1 Example 1: MQTT over TCP (with a Local MQTT Broker)

Create a local MQTT broker. For example, the MQTT broker's IP address is "192.168.31.113", port 1883. Then the example of communicating with the MQTT broker will be as the following steps.

```
AT+MQTTUSERCFG=0,1,"ESP32","espressif","1234567890",0,0,""
AT+MQTTCONN=0,"192.168.31.113",1883,0
AT+MQTTSUB=0,"topic",1
AT+MQTTPUB=0,"topic","test",1,0
AT+MQTTCLEAN=0
```

4.3.2 Example 2: MQTT over TLS (with a Local MQTT Broker)

Create a local MQTT broker. For example, the MQTT broker's IP address is "192.168.31.113", port 1883. Then the example of communicating with the MQTT broker will be as the following steps.

```
AT+CIPSNTPCFG=1,8,"ntp1.aliyun.com"
AT+CIPSNTPTIME?
AT+MQTTUSERCFG=0,3,"ESP32","espressif","1234567890",0,0,""
AT+MQTTCONNCFG=0,0,0,"lwtt","lwtm",0,0
AT+MQTTCONN=0,"192.168.31.113",1883,0
AT+MQTTSUB=0,"topic",1
AT+MQTTPUB=0,"topic","test",1,0
AT+MQTTCLEAN=0
```

4.3.3 Example 3: MQTT over WSS

This is an example of communicating with MQTT broker: iot.eclipse.org, of which port is 443.

```
AT+CIPSNTPCFG=1,8,"ntp1.aliyun.com"
AT+CIPSNTPTIME?
AT+MQTTUSERCFG=0,7,"ESP32","espressif","1234567890",0,0,"wss"
AT+MQTTCONN=0,"iot.eclipse.org",443,0
AT+MQTTSUB=0,"topic",1
AT+MQTTPUB=0,"topic","test",1,0
AT+MQTTCLEAN=0
```

4.4 [ESP32 Only] Ethernet AT Examples

Note: Before you run any Ethernet AT commands, please make sure you have followed the *Prerequisite*.

- *Establish a TCP Connection on Ethernet Network*

4.4.1 Establish a TCP Connection on Ethernet Network

1. Enable multiple connections.
   - Command

   ```
   AT+CIPMUX=1
   ```

   - Response

   ```
   OK
   ```

2. Create a TCP server.
   - Command

   ```
   AT+CIPSERVER=1,8081
   ```

   - Response

   ```
   OK
   ```

3. Obtain the IP address of the server.
   - Command

   ```
   AT+CIPETH?
   ```

   - Response

   ```
   +CIPETH:ip:192.168.105.24
   +CIPETH:gateway:192.168.105.1
   +CIPETH:netmask:255.255.255.0
   OK
   ```

4. Use a network tool on PC to create a TCP client and connect to the TCP server that ESP created. (IP: 192.168.105.24, Port: 8081).

5. Send data in *Normal Transmission Mode*.
   - Command

   ```
   AT+CIPSEND=0,4     // send 4 bytes to connection ID 0
   ```

   - Response

   ```
   OK
   >
   ```

   - Enter the data.

- Response

```
SEND OK
```

Note: If the length of the data you entered is more than the value of <length> (n) set by `AT+CIPSEND`, the system will prompt `busy`, and send the first n bytes. After sending the first n bytes, the system will reply `SEND OK`.

6. Receive data in *Normal Transmission Mode*.

   When the ESP server receives data, AT will prompt:

```
+IPD,n:xxxxxxxxxx    // received n bytes, data=xxxxxxxxxxx
```

7. Close TCP Connection.

   - Command

```
AT+CIPCLOSE=0
```

   - Response

```
OK
```

8. Delete the TCP server.

   - Command

```
AT+CIPSERVER=0
```

   - Response

```
OK
```

4.5 Web Server AT Example

[]

This document mainly introduces the use of AT web server, mainly involving the following applications:

Note: The default firmware does not support web server AT commands, please refer to *Web server AT Commands* to enable the web server.

4.5.1 Wi-Fi Provisioning Using a Browser

Introduction

With the web server, mobile phone or PC is able to control ESP device‘s Wi-Fi provisioning service. You can use a mobile phone or computer to connect to the SoftAP of the ESP device, open the web pages via browser, start provisioning service, and then the ESP device can connect to the target router as you set.

Introduction to Operation Steps

The whole process can be divided into the following three steps:

- *Use STA Device to Connect to ESP Device*
- *Use the Browser to Send Wi-Fi Connection Information*
- *Get the Result of Wi-Fi Connection*

Use STA Device to Connect to ESP Device

Firstly, ESP device needs to be configured to softAP + STA mode, and creates a web server to wait for Wi-Fi provisioning messages. In this case, a mobile phone or a PC can connect to the ESP softAP as a station. The corresponding AT commands are as follows:

1. Clear the previous Wi-Fi provisioning information.

   - Command

   ```
   AT+RESTORE
   ```

2. Set the Wi-Fi mode to Station+SoftAP.

   - Command

   ```
   AT+CWMODE=3
   ```

3. Set the configuration of an ESP SoftAP.For example, set the default connection ssid to "pos_softap", Wi-Fi without password.

   - Command

   ```
   AT+CWSAP="pos_softap","",11,0,3
   ```

4. Enable multiple connections.

   - Command

   ```
   AT+CIPMUX=1
   ```

5. Create a web server, port: 80, connection timeout: 25 s (default maximum is 60 s).

   - Command

   ```
   AT+WEBSERVER=1,80,25
   ```

After starting the web sever according to the above commands, you can turn on the Wi-Fi connection function on your STA device, and connect it to the softAP of the ESP device:

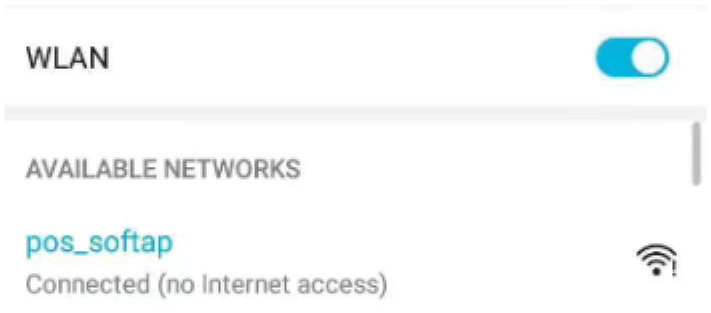


Fig. 1: The browser connects to the ESP AP

Use the Browser to Send Wi-Fi Connection Information

After your STA device connected to the ESP softAP, it can send Wi-Fi connection information to ESP in an HTTP request. Please note that if your target AP is the hotspot of the device which opens the web pages, you may not receive the Wi-Fi connection result. You can enter the default IP address of the web server in the browser (the default IP is 192.168.4.1, or you can query the current SoftAP IP address by command AT+CIPAP?), open the Wi-Fi provisioning interface, and enter the ssid and password of the router to be connected, click "Connect" to let ESP device start connecting to the router:

Or you can click the drop-down box of SSID to list all APs nearby, select the target AP and enter the password, and then click "Connect" to let the ESP device start connecting to the router:

Get the Result of Wi-Fi Connection

After the Wi-Fi connection is established successfully, the web page will be displayed as follows:

Note 1: After the Wi-Fi connection is established successfully, the webpage will be closed automatically. If you want to continue to access the webpage, please re-enter the IP address of the ESP device and reopen the webpage.

At the same time, the following messages will be returned from the ESP-AT command port:

```
+WEBSERVERRSP:1       // meaning that ESP device has received Wi-Fi connection␣
↪information
WIFI CONNECTED        // meaning that ESP device is connecting
WIFI GOT IP           // meaning that ESP device connect successfully to the␣
↪destination rounter
+WEBSERVERRSP:2       // meaning that STA device has received Wi-Fi connection result,␣
↪and web resources can be released
```

If the ESP device fails to connect to the router, the web page will display:

At the same time, the following messages will be returned from the ESP-AT command port:

```
+WEBSERVERRSP:1       // meaning that ESP device has received Wi-Fi connection␣
↪information, but failed to connect to the rounter.
```

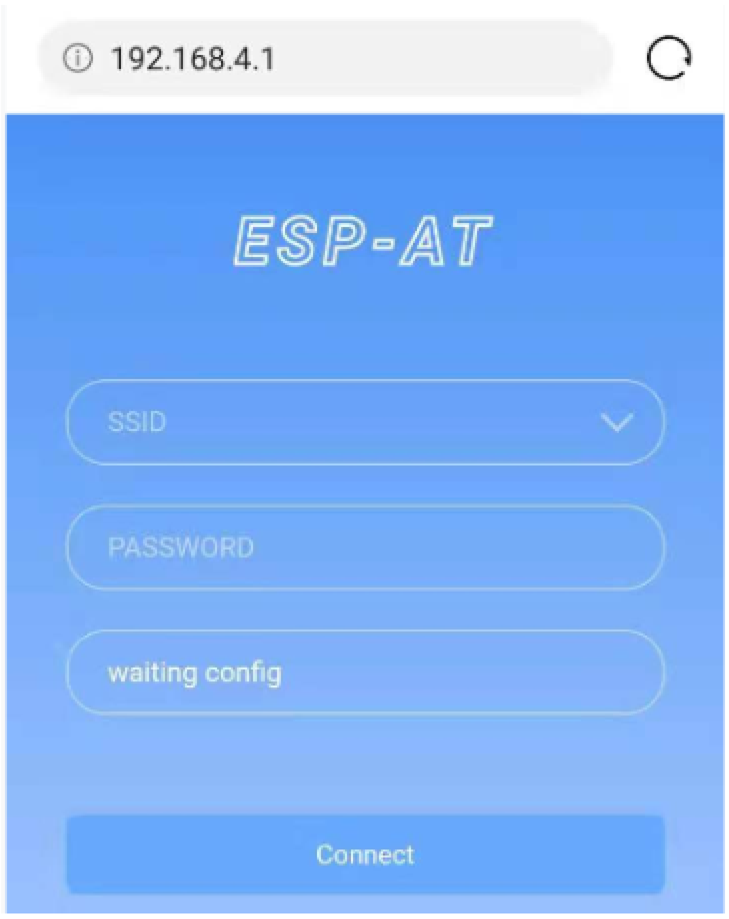


Fig. 2: The browser opens the Wi-Fi provisioning interface

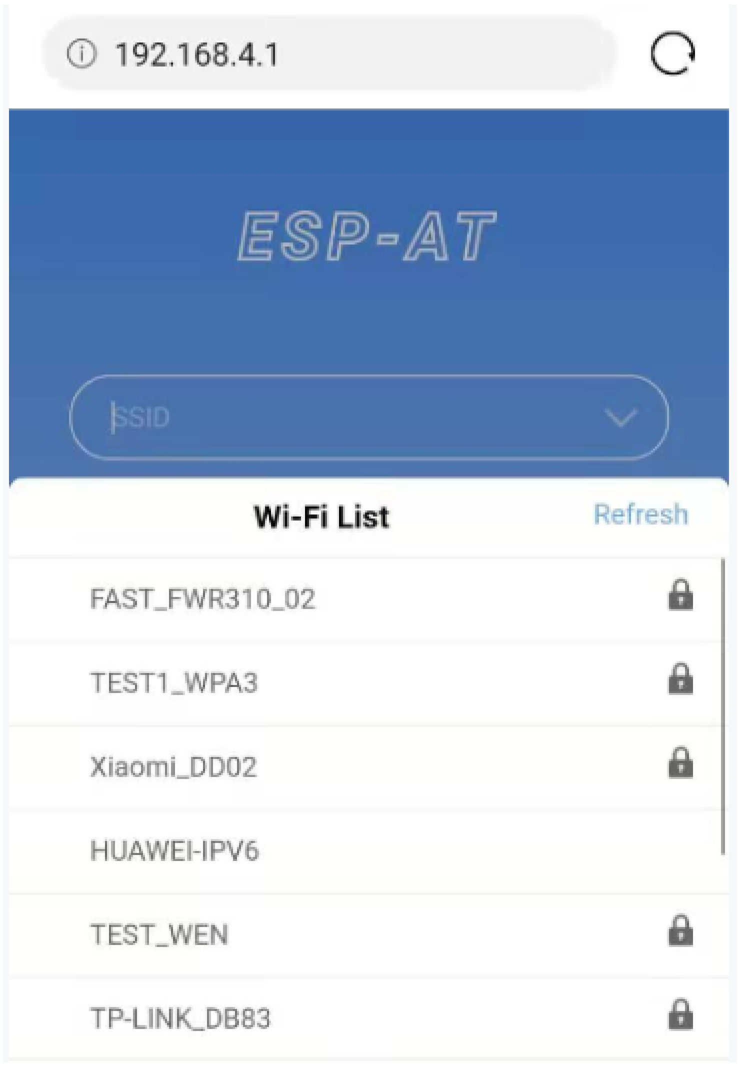
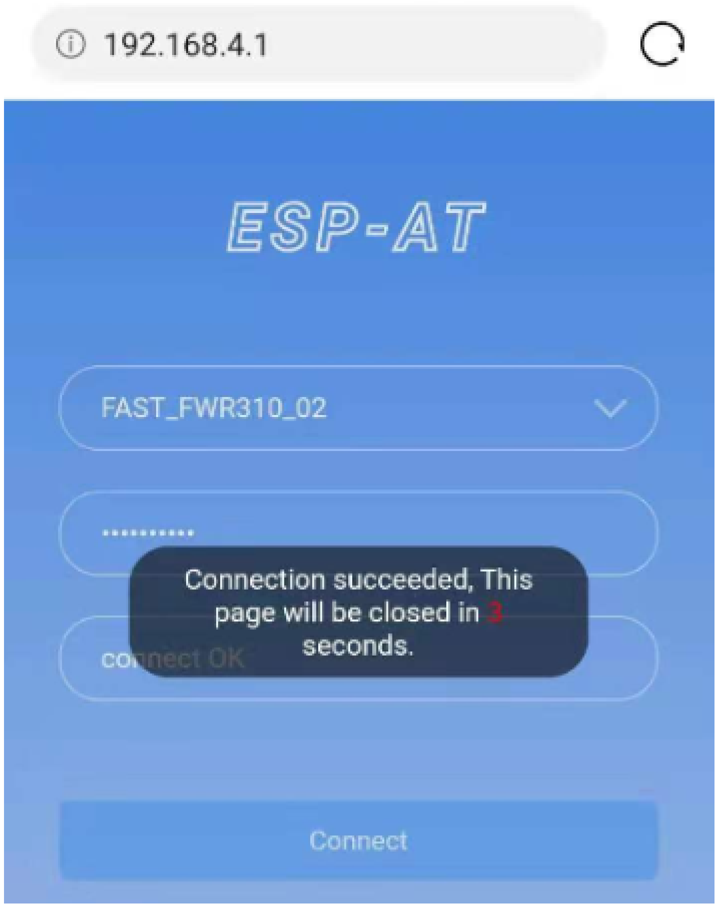


Fig. 3: Schematic diagram of browser obtaining Wi-Fi AP list



Connection succeeded, This page will be closed in 3 seconds.

Fig. 4: Wi-Fi connection is established successfully

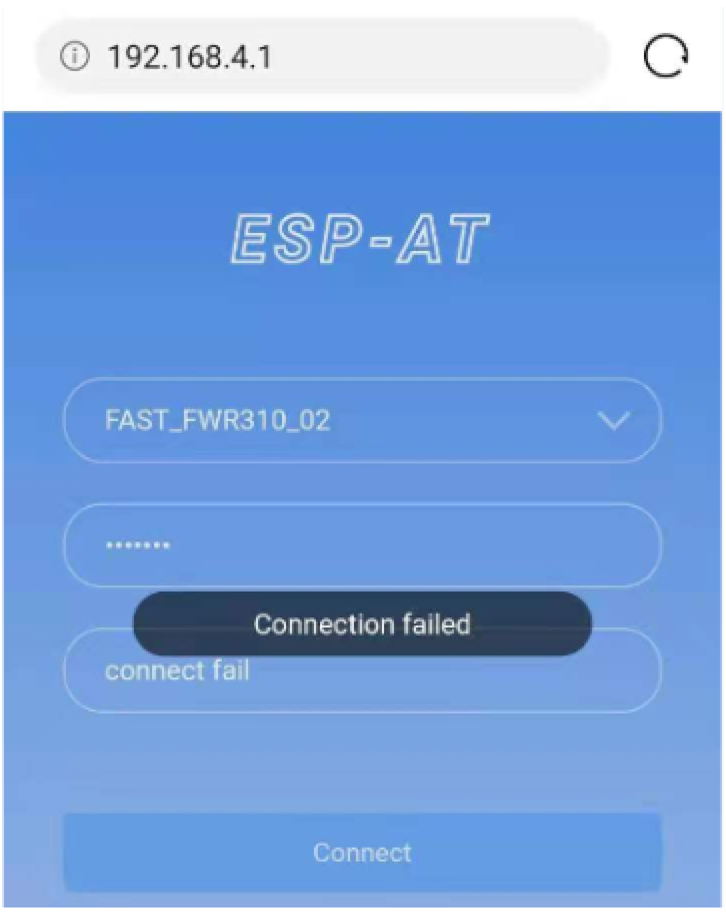


Fig. 5: ESP device fails to connect to the router

Troubleshooting

Note 1: The network configuration page received a prompt "Connection failed". Please check whether the Wi-Fi AP of the ESP module is correctly turned on, and the relevant configuration of the AP, and confirm that the correct AT command has been entered to successfully enable the web server.

4.5.2 OTA Firmware Upgrade Using a Browser

Introduction

After the browser opens the web page of the web server, you can choose to enter the OTA upgrade page to upgrade the firmware of the ESP device through the web page.

Introduction to Operation Steps

Open the OTA Configuration Page

As shown in the figure, click on the "OTA" option in the lower right corner of the web page, and after opening the OTA configuration page, you can view the current firmware version and AT Core version:

Note 1: The configuration interface can only be opened when the STA device is connected to the AP of the ESP device, or the STA device accessing the OTA configuration page is connected to the ESP device in the same subnet.

Note 2: The "current app version" displayed on the webpage is the version number of the application. You can change the version number through `./build.py menuconfig` -> `Component config` -> `AT` -> `AT firmware version` (see *Build Your Own ESP-AT Project*). In this case, you can manage your application firmware version.

Select and Send the New Firmware

As shown in the figure, click the "Browse" button on the page and select the new firmware to be sent:

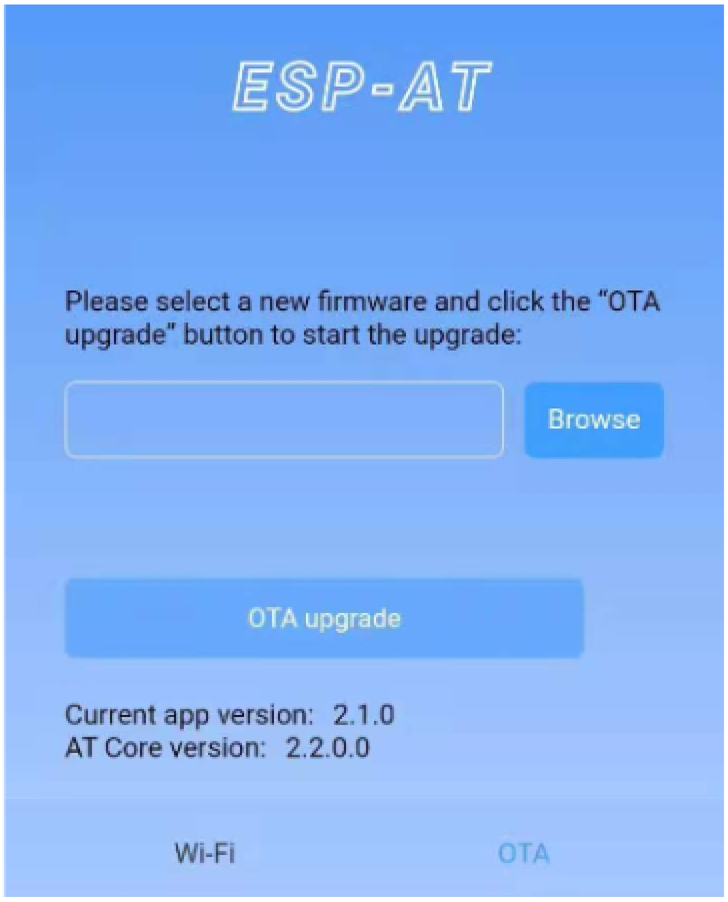
Note 1: Before sending the new firmware, the system will check the selected firmware. The suffix of the firmware name must be .bin, and its size should not exceed 2M.

Get the Result of OTA

As shown in the figure, if the ESP device OTA successfully, it will prompt "OTA Succeeded":

At the same time, the following messages will be returned from the ESP-AT command port:

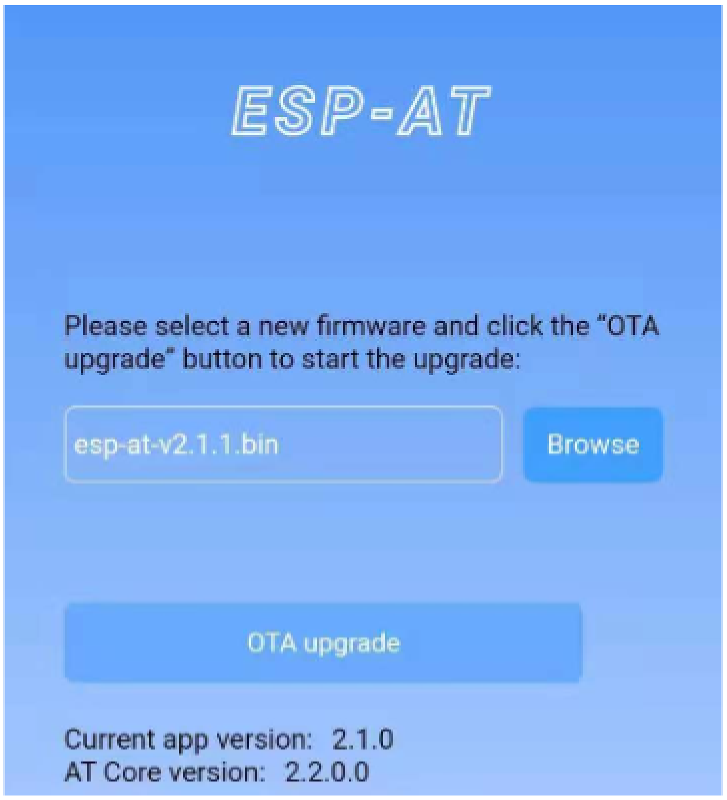
```
+WEBSERVERRSP:3      // meaning that ESP device begin to receive ota data
+WEBSERVERRSP:4      // meaning that ESP device has received all firmware dataand you␣
↪can choose to restart the ESP device to apply the new firmware
```



Please select a new firmware and click the "OTA upgrade" button to start the upgrade:

Current app version: 2.1.0
AT Core version: 2.2.0.0

Fig. 6: OTA configuration page



ESP-AT
Please select a new firmware and click the "OTA upgrade" button to start the upgrade:
esp-at-v2.1.1.bin
Browse
OTA upgrade
Current app version: 2.1.0
AT Core version: 2.2.0.0

Fig. 7: Select the new version of firmware to be sent

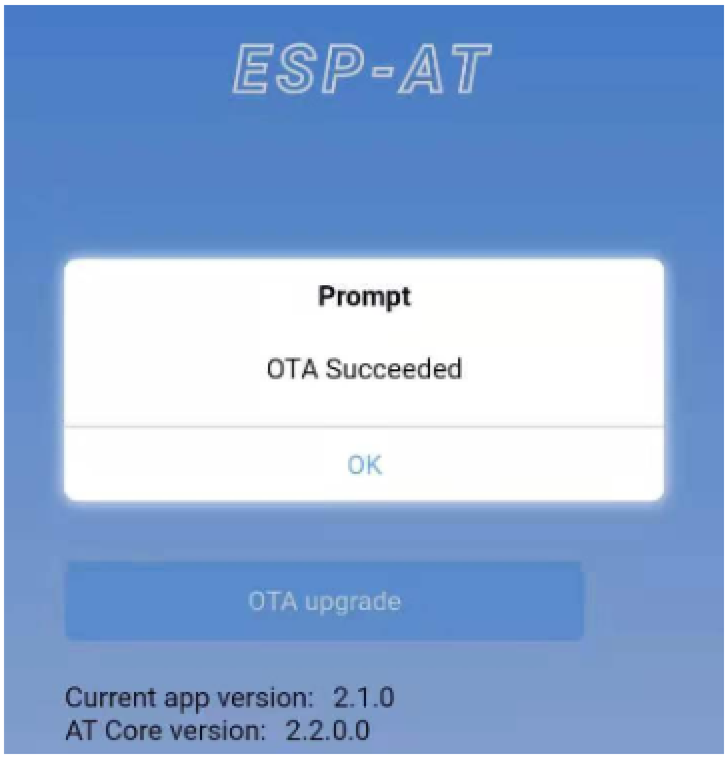


Fig. 8: The new firmware was sent successfully

If the received firmware data verification fails, the following message will be received on the serial port:

```
+WEBSERVERRSP:3       // meaning that ESP device begin to receive ota data
+WEBSERVERRSP:5       // meaning that the received OTA data verification failed. You␣
↪can choose to reopen the OTA configuration interface and follow the above steps to␣
↪restart the firmware upgrade
```

4.5.3 Wi-Fi Provisioning Using a WeChat Applet

Introduction

The WeChat applet can automatically connect to the WiFi AP of the ESP device, and then send the ssid and password required by the ESP device to connect to the network.

Introduction to Operation Steps

The whole process can be divided into the following four steps:

- *Configure ESP Device Parameters*
- *Load WeChat Applet*
- *Target AP Selection*
- *Use the WeChat Applet to Send Wi-Fi Connection Information*

Configure ESP Device Parameters

Firstly, ESP device needs to be configured to softAP + STA mode, and creates a web server to wait for Wi-Fi provisioning messages. In this case, a mobile phone or a PC can connect to the ESP softAP as a station. The corresponding AT commands are as follows:

1. Clear the previous Wi-Fi provisioning information.

   - Command

   ```
   AT+RESTORE
   ```

2. Set the Wi-Fi mode to Station+SoftAP.

   - Command

   ```
   AT+CWMODE=3
   ```

3. Set the configuration of an ESP SoftAP.For example, set the default connection ssid to “pos_softap” , and password to “espressif”.

   - Command

   ```
   AT+CWSAP="pos_softap","espressif",11,3,3
   ```

Note: By default, the WeChat applet initiates a connection to the SoftAP whose ssid is *pos_softap* and password is *espressif*. Please make sure to set the parameters of the ESP device according to the above configuration.

1. Enable multiple connections.

   - Command

   ```
   AT+CIPMUX=1
   ```

2. Create a web server, port: 80, connection timeout: 40 s (default maximum is 60 s).

   - Command

   ```
   AT+WEBSERVER=1,80,40
   ```

Load WeChat Applet

Open the mobile phone WeChat, scan the following QR code:

Fig. 9: Get the QR code of the applet

Open the WeChat applet and enter the Wi-Fi provisioning interface:

Target AP Selection

After loading the WeChat applet, there are two situations according to different target AP:

Situation 1. If your target AP is the hotspot of the mobile phone which running the WeChat applet, please select the "Local phone hotspot" option box on the WeChat applet page.

Situation 2. If your target AP is just another AP, not as the special situation one as above, then please do not select the "Local phone hotspot" option box.

Use the WeChat Applet to Send Wi-Fi Connection Information

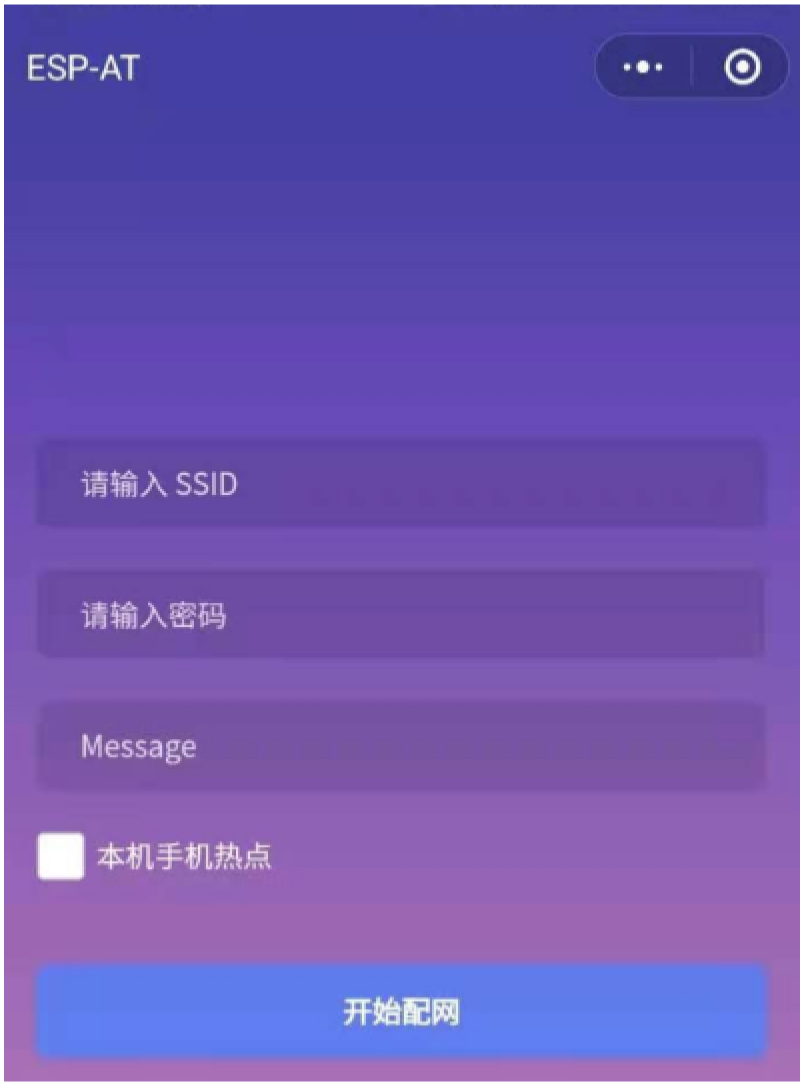


Fig. 10: Wi-Fi provisioning interface

The target AP to be accessed is not the hotspot provided by the mobile phone which loading the WeChat applet.

Here, take connecting to a router as an example, the process of Wi-Fi Connection configuration is introduced:

1.Turn on the mobile Wi-Fi and connect to the router:

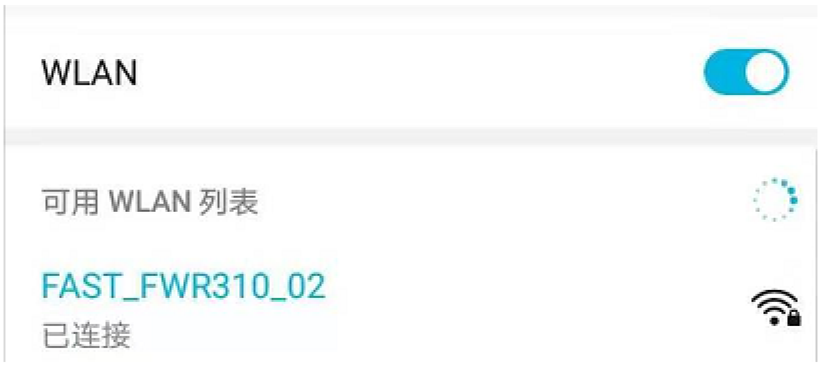


Fig. 11: connect to the rounter

2.Open the WeChat applet, you can see that the applet page has automatically displayed the ssid of the current router as "FAST_FWR310_02".

Note: If the ssid of the connected router is not displayed on the current page, please click "Re-enter applet" in the following figure to refresh the current page:

3.After entering the password of the router, click "Connect".

4.After the Wi-Fi connection is established successfully, the web page will be displayed as follows:

At the same time, the following messages will be returned from the ESP-AT command port:

```
+WEBSERVERRSP:1       // meaning that ESP device has received Wi-Fi connection␣
↪information
WIFI CONNECTED        // meaning that ESP device is connecting
WIFI GOT IP           // meaning that ESP device connect successfully to the␣
↪destination rounter
+WEBSERVERRSP:2       // meaning that STA device has received Wi-Fi connection result,␣
↪and web resources can be released
```

5.If the ESP device fails to connect to the router, the page will display:

At the same time, the following messages will be returned from the ESP-AT command port:

```
+WEBSERVERRSP:1       // meaning that ESP device has received Wi-Fi connection␣
↪information, but failed to connect to the rounter.
```

The target AP to be accessed is the hotspot provided by the mobile phone which loading the WeChat applet.

If the target AP to be accessed is the hotspot provided by the mobile phone which loading the WeChat applet, it is not necessary to enter the ssid, but only needs to enter the password of the AP, and turn on the mobile AP in time according to the prompts.

1.Select the "Local phone hotspot" option box on the WeChat applet page, enter the password of the local hotspot, and click "Connect".

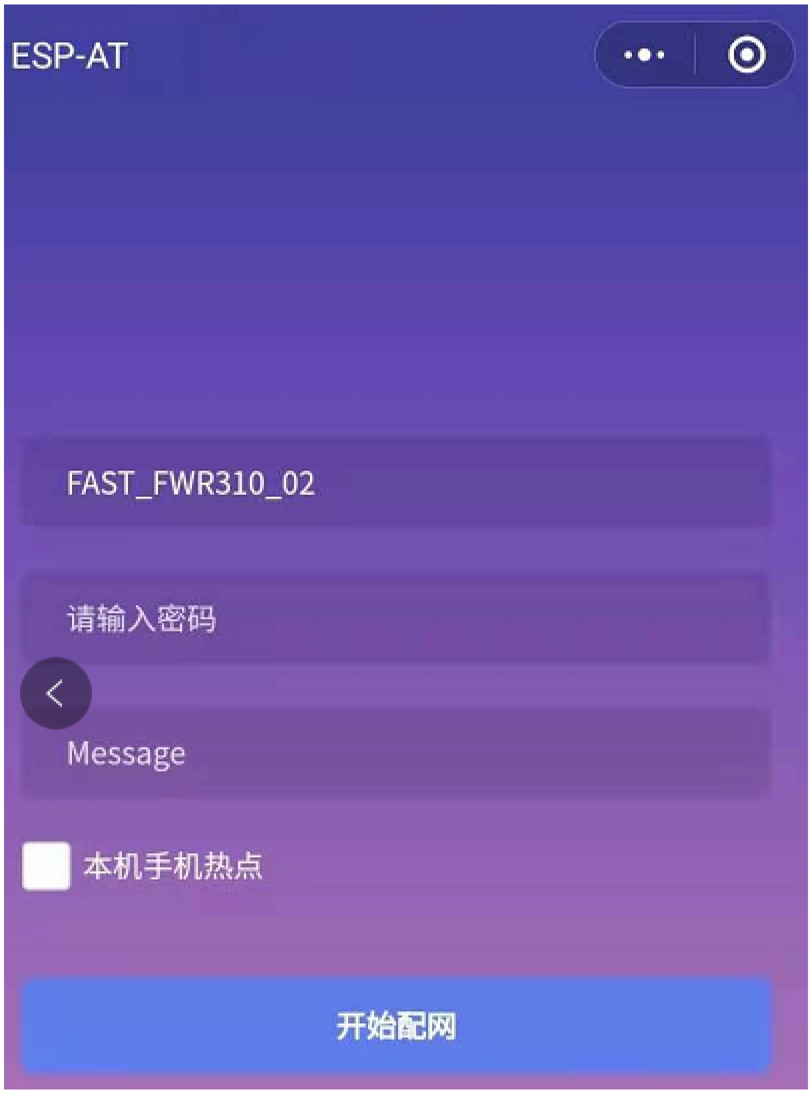


Fig. 12: The applet obtains the information of the router to be connected

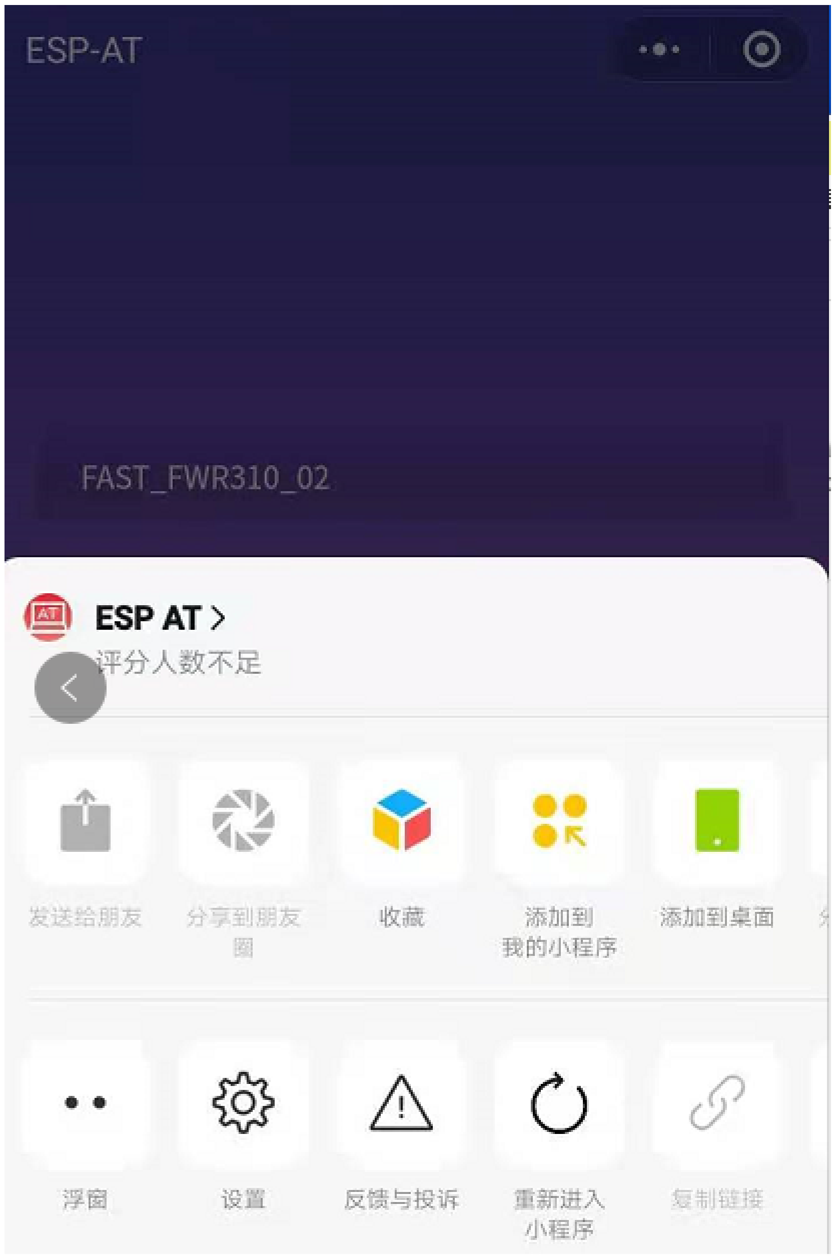


Fig. 13: Re-enter the applet

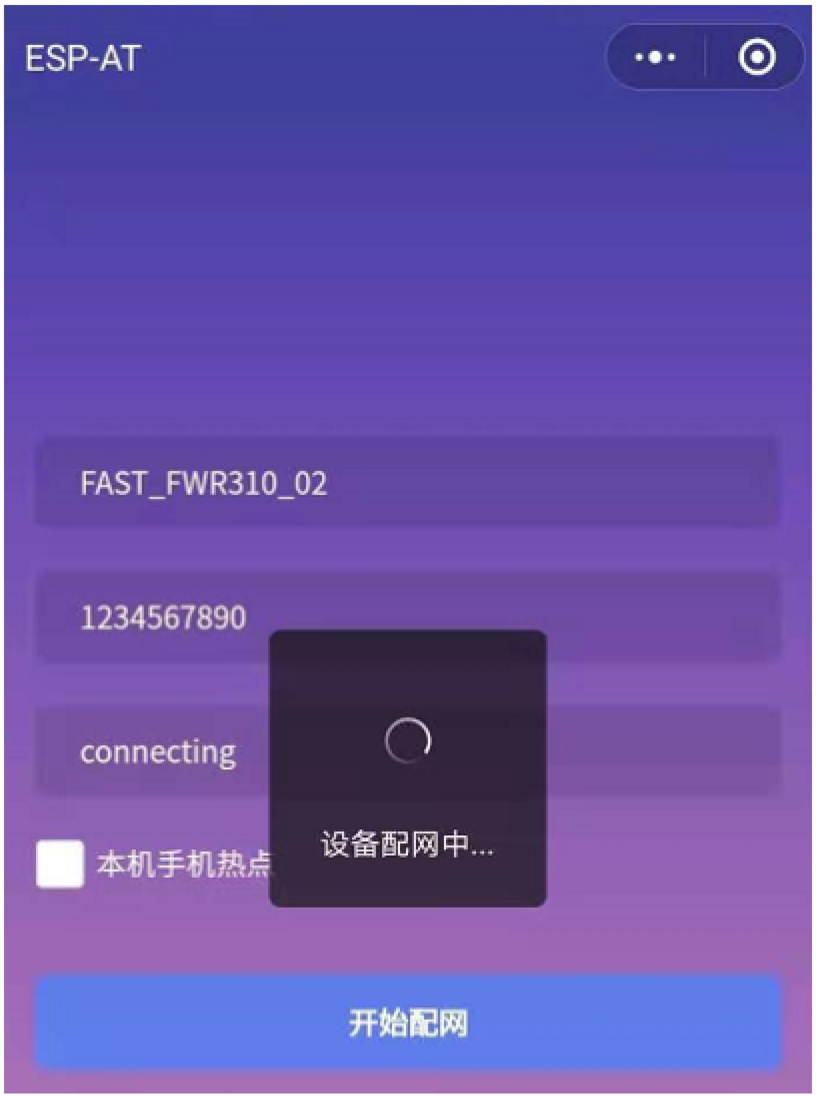


Fig. 14: The applet starts the ESP device to connect to the router

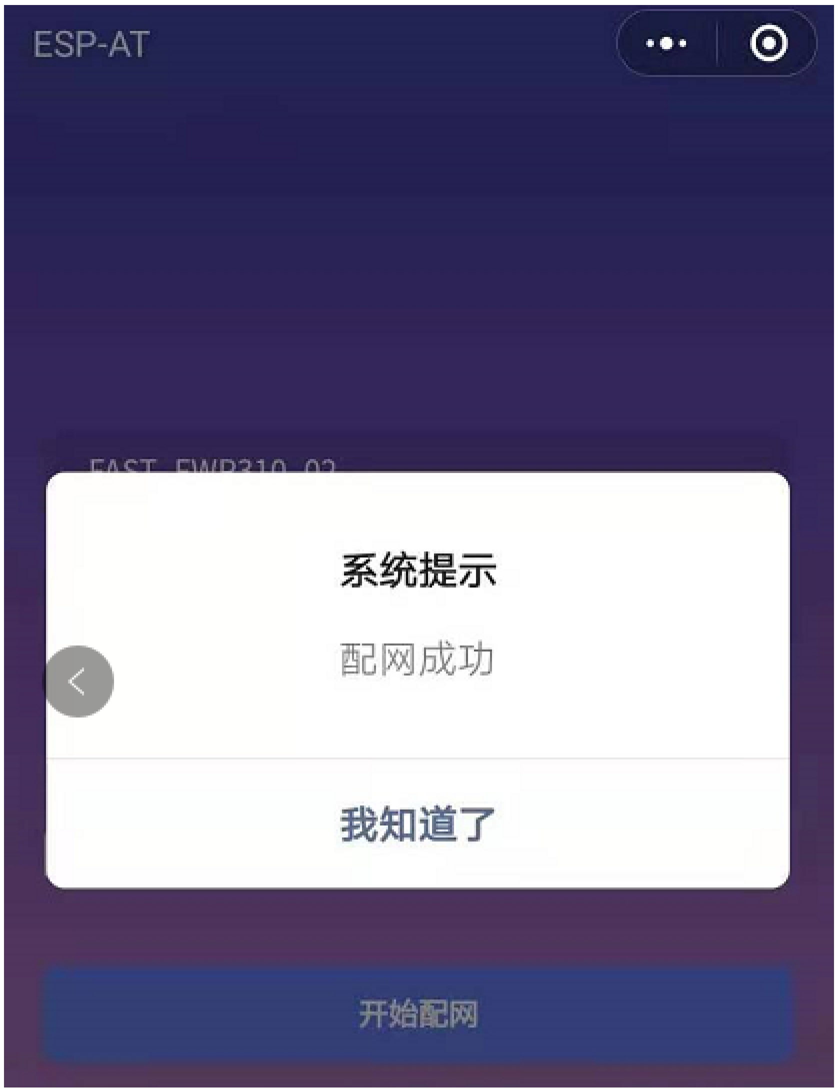


Fig. 15: The applet Wi-Fi provisioning is successful

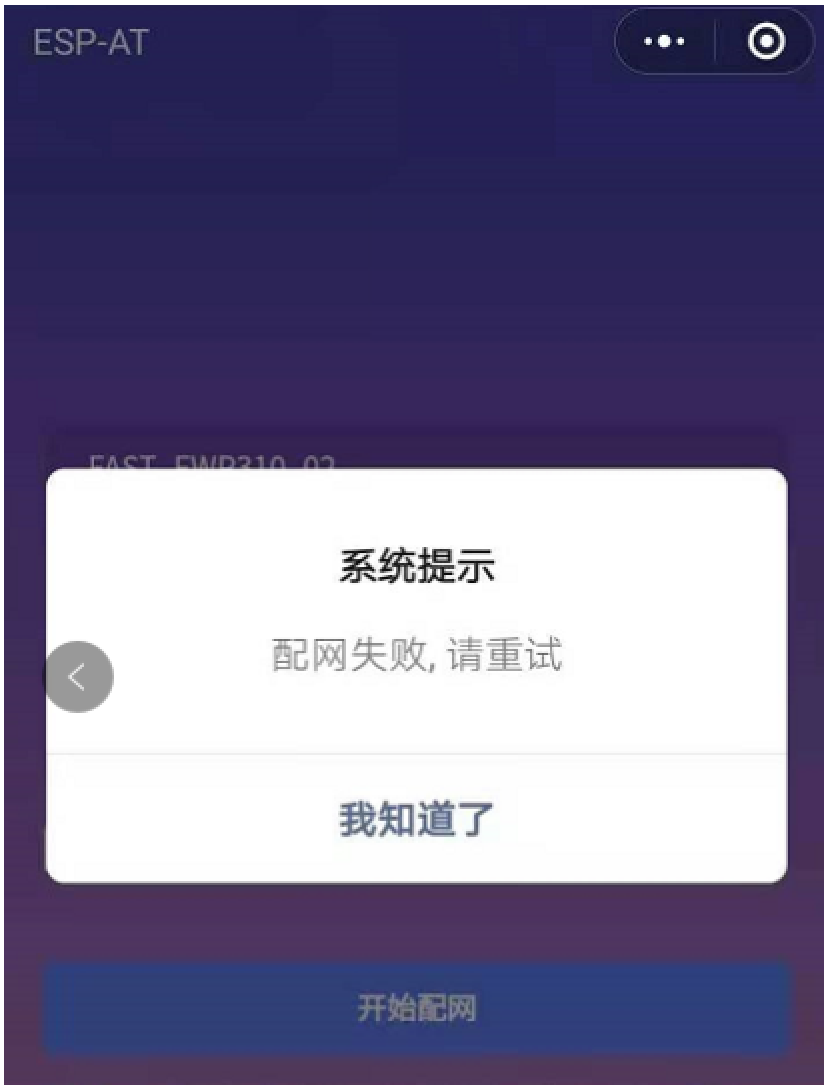


Fig. 16: The applet Wi-Fi provisioning is failed

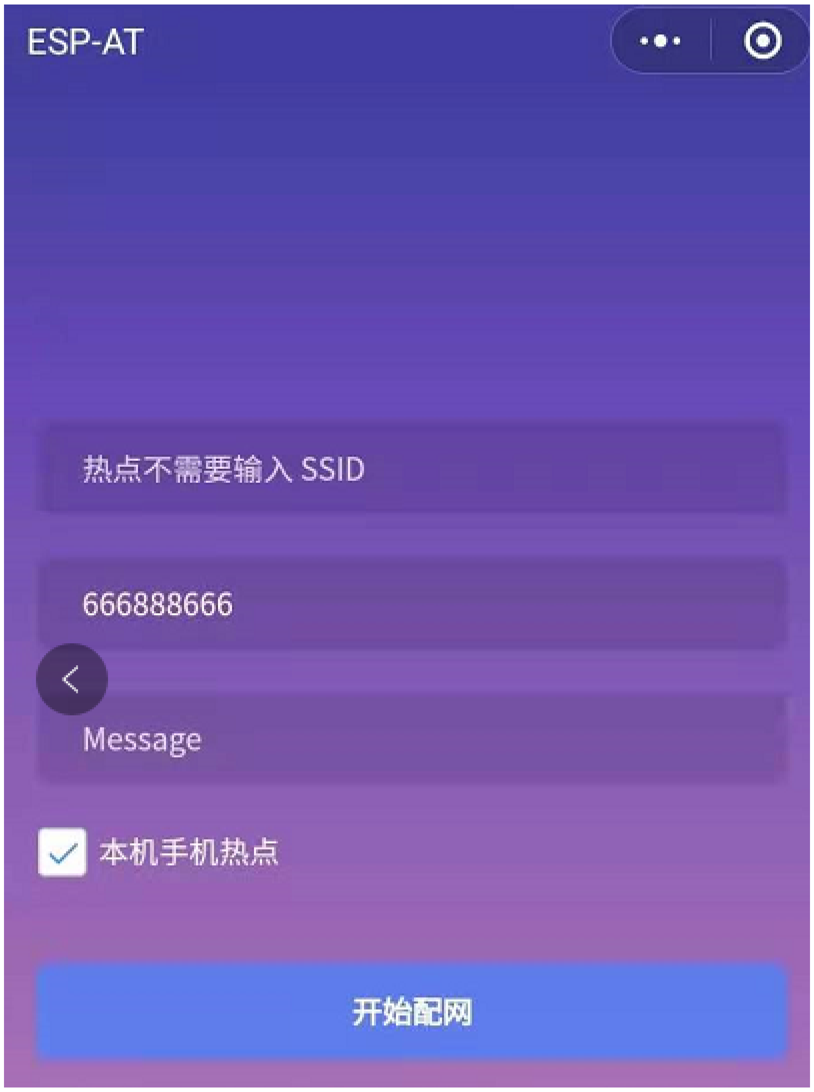


Fig. 17: Enter the password of the AP

2.After receiving the prompt "Connecting to the mobile phone hotspot", please check that the local mobile phone hotspot is turned on. At this time, the ESP device will automatically scan the surrounding hotspots and initiate a connection.

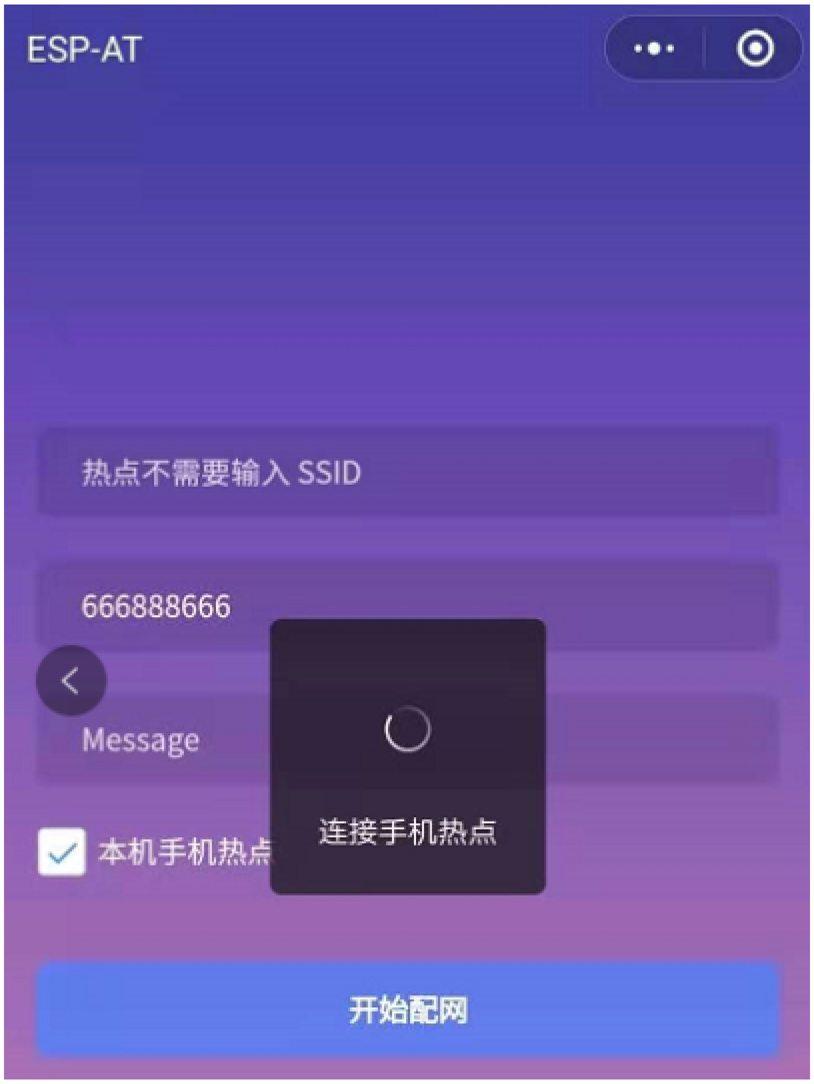


Fig. 18: Start to connect to the AP

3.The display of the WiFi connection results on the applet page and the data output from the serial port are the same as the above-mentioned "The target AP to be accessed is not the hotspot provided by the mobile phone which loading the WeChat applet.", please refer to the above.

Troubleshooting

Note 1: The Wi-Fi provisioning page received a prompt "Data transmission failed". Please check whether the Wi-Fi AP of the ESP device is correctly turned on, and the relevant configuration of the AP, and confirm that the correct AT command has been entered to successfully enable the web server.

Note 2: The Wi-Fi provisioning page receives a prompt "Failed to connect to the AP". Please check whether the Wi-Fi connection function of the mobile phone is turned on, check whether the Wi-Fi AP of the ESP device is correctly turned on, and whether the ssid and password of the AP are configured according to the above steps.

Note 3: The Wi-Fi provisioning page receives a prompt "The Wi-Fi provisioning saved by the system expired". Please manually connect the ESP device AP with a mobile phone, and confirm that the ssid and password of the ESP module have been configured according to the above steps.

4.5.4 OTA Firmware Upgrade Using a WeChat Applet

The WeChat applet support online firmware upgrade , please refer to the above-described *Configure ESP Device Parameters* specific steps performed ESP device configuration (if the configuration has been completed, do not repeat configuration). Once configured, the device performs OTA firmware upgrade processes is similar as *OTA Firmware Upgrade Using a Browser* .

4.5.5 [ESP32][ESP32-S Series][ESP32-C Series] Using Capture Portal

Introduction

Captive Portal is commonly used to present a specified page to newly connected devices of a Wi-Fi or wired network. For more information about Captive Portal, please refer to Captive Portal Wiki .

Note: The default firmware does not support web server Captive Portal, you may enable it by `./build.py menuconfig` > `Component config` > `AT` > `AT WEB Server command support` > `AT WEB captive portal support` and build the project (see *Build Your Own ESP-AT Project*). In addition, enabling this feature may cause page skipping when using wechat applet for Wi-Fi provisioning or OTA firmware upgrade. It is recommended that this feature be enabled only when accessing at web using browser.

Introduction to Operation Steps

After Enable Captive Portal support, please refer to *Use STA Device to Connect to ESP Device* to complete the configuration of the ESP device, and then connect to the AP of the ESP device:

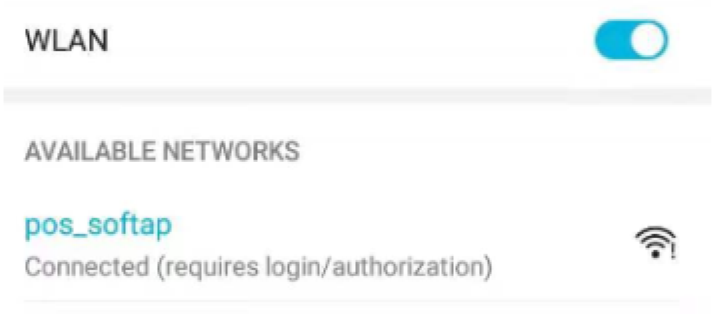


Fig. 19: Connect to the AP with Captive Portal enabled

As shown in the figure above, after the Station device is connected to the AP of the ESP device with the Captive Portal function enabled, it will prompt "requires login/authentication", and then the browser will automatically open and jump to the main interface of AT Web. If it cannot be redirected automatically, please follow the instructions of the Station device, click "Authentication" or click the name of the "pos_softap" hotspot in the figure above to manually trigger the Captive Portal to automatically open the browser and enter the main interface of AT Web.

Troubleshooting

Note 1: Both Station device and AP device support the Captive Portal function to ensure the normal use of this function. Therefore, if the device is connected to the AP of the ESP device, but it does not prompt "Login/Authentication", it may be that the Station device does not support this function. In this case, please refer to the specific steps of *Use the Browser to Send Wi-Fi Connection Information* above to open the main interface of AT Web.

CHAPTER 5

How to compile and develop your own AT project

[]

5.1 Build Your Own ESP-AT Project

This document details how to build your own ESP-AT project and flash the generated binary files into your ESP devices, including ESP32, ESP32-S2, ESP32-C3, and ESP8266. It comes in handy when the default *AT Binary Lists* cannot meet your needs, for example, to customize the *AT port* pins, Bluetooth service, partitions, and so on.

The structure of this document is as follows:

- *Overview*: Overview of the steps to build an ESP-AT project.
- *ESP32, ESP32-S2 and ESP32-C3 Series*: Details steps to build a project for ESP32, ESP32-S2, and ESP32-C3 series.
- *ESP8266 Series*: Details steps to build a project for ESP8266.

5.1.1 Overview

Before compiling an ESP-AT project, you need first get started with ESP-IDF and set up the environment for ESP-IDF, because ESP-AT is based on ESP-IDF.

After the environment is ready, install the tools and ESP-AT SDK. Then, connect your ESP device to PC. Use `./build.py menuconfig` to set up some configuration for the project. Build the project and flash the generated bin files onto your ESP device.

Note: Please pay attention to possible conflicts of pins. If choosing `AT through HSPI`, you can get the information of the HSPI pin by `./build.py menuconfig` –> `Component config` –> `AT` –> `AT hspi settings`.

5.1.2 ESP32, ESP32-S2 and ESP32-C3 Series

This section describes how to compile an ESP-AT project for ESP32, ESP32-S2 and ESP32-C3 series.

Get Started with ESP-IDF

Get started with ESP-IDF before compiling an ESP-AT project, because ESP-AT is developed based on ESP-IDF, and the supported version varies from series to series:

Table 1: ESP-IDF Versions for Different Series

Project	IDF Version	IDF Documentation Version
ESP32 ESP-AT	release/v4.2	ESP-IDF Get Started Guide v4.2
ESP32-S2 ESP-AT	release/v4.2	ESP-IDF Get Started Guide v4.2
ESP32-C3 ESP-AT	release/v4.3	ESP-IDF Get Started Guide v4.3

First, set up the development environment for ESP-IDF according to Step 1 to 4 of *ESP-IDF Get Started Guide* (click the corresponding link in the table above to navigate to the documentation).

Then, start a simple project of `hello_world` according to Step 5 to 10 of *ESP-IDF Get Started Guide* to make sure your environment works and familiarize yourself with the process of starting a new project based on ESP-IDF. It is not a must, but you are strongly recommended to do so.

After finishing all the ten steps (if not, at least the first four steps), you can move onto the following steps that are ESP-AT specific.

Note: Please do not set `IDF_PATH` during the process, otherwise, you would encounter some unexpected issues when compiling ESP-AT projects later.

Get ESP-AT

To compile an ESP-AT project, you need the software libraries provided by Espressif in the ESP-AT repository.

To get ESP-AT, navigate to your installation directory and clone the repository with git clone, following instructions below specific to your operating system.

- Linux or macOS

```
cd ~/esp
git clone --recursive https://github.com/espressif/esp-at.git
```

- Windows

```
cd %userprofile%\esp
git clone --recursive https://github.com/espressif/esp-at.git
```

If you are located in China or have difficulties to access GitHub, you can also use `git clone https://gitee.com/EspressifSystems/esp-at.git` to get ESP-AT, which may be faster.

ESP-AT will be downloaded into `~/esp/esp-at` on Linux or macOS, or `%userprofile%\esp\esp-at` on Windows.

Note: This guide uses the directory `~/esp` on Linux or macOS, or `%userprofile%\esp` on Windows as an installation folder for ESP-AT. You can use any directory, but you will need to adjust paths for the commands respectively. Keep in mind that ESP-AT does not support spaces in paths.

Connect Your Device

Connect your device to the PC with a USB cable to download firmware and log output. See *Hardware Connection* for more information. Note that you do not need to set up the "AT command/response" connection if you do not send AT commands and receive AT responses during the compiling process. You can change default port pins referring to *How to Set AT Port Pins*.

Configure

In this step, you will clone the `esp-idf` folder into the `esp-at` folder, set up the development environment in the newly cloned folder, and configure your project.

1. Navigate to `~/esp/esp-at` directory.
2. Run the project configuration utility `menuconfig` to configure.

```
./build.py menuconfig
```

3. Select the following configuration options for your ESP device if it is your first time.

- Select the `Platform name` for your ESP device. For example, select `PLATFORM_ESP32` for ESP32 series of products, `PLATFORM_ESP32S2` for ESP32-S2 series of products. `Platform name` is defined in factory_param_data.csv .
- Select the `Module name` for your ESP device. For example, select `WROOM-32` for the ESP32-WROOM-32D module. `Module name` is defined in factory_param_data.csv .
- Enable or disable `silence mode`. If enabled, it will remove some logs and reduce the firmware size. Generally, it should be disabled.
- The above three option items will not appear if the file `build/module_info.json` exists. So please delete it if you want to reconfigure the module information.

4. Now, the `esp-idf` folder is created in `esp-at` folder. This `esp-idf` is different from that in Step **Get Started with ESP-IDF**.

- If the terminal prompt an error message like the following, please proceed with the next step to set up the develop environment in the `esp-at/esp-idf`.

```
The following Python requirements are not satisfied:
...
Please follow the instructions found in the "Set up the tools" section of ESP-IDF␣
↪Get Started Guide.
```

- If you have compiled an ESP-AT project for an ESP series before and want to switch to another series, you must run `rm -rf esp-idf` to remove the old esp/idf and then proceed with the next step.
- If your esp-idf is upgraded, you are recommended to proceed with the next step.

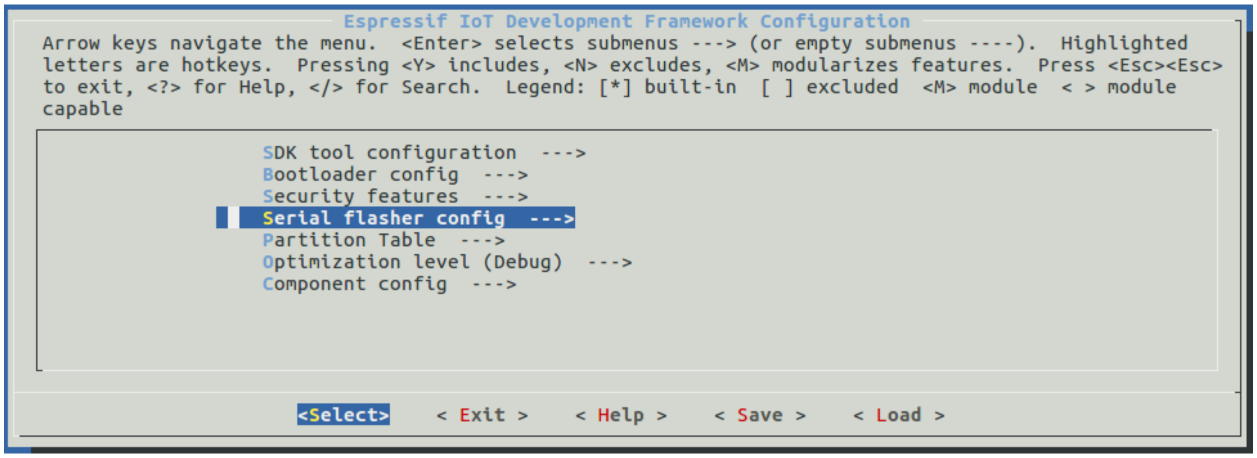
5. Set up the development environment in the `esp-at/esp-idf`.

- Set up the tools in the folder if it is the first time you compile the ESP-AT project. See Step 3 of *ESP-IDF Get Started Guide*.

- Set up environment variables in the folder **every time** you compile an ESP-AT project. See Step 4 of *ESP-IDF Get Started Guide*.
- **Install** `pyyaml` **and** `xlrd` **packages with pip in the folder if you have not done it**.

```
python -m pip install pyyaml xlrd
```

If the previous steps have been done correctly, the following menu appears after you run `./build.py menuconfig`:



```
Espressif IoT Development Framework Configuration
Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted
letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc>
to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > module
capable

                    SDK tool configuration  --->
                    Bootloader config  --->
                    Security features  --->
                    Serial flasher config  --->
                    Partition Table  --->
                    Optimization level (Debug)  --->
                    Component config  --->

          <Select>    < Exit >    < Help >    < Save >    < Load >
```

Fig. 1: Project configuration - Home window

You are using this menu to set up project-specific configuration, e.g. changing *AT port* pins, enabling Classic Bluetooth function, etc. If you made no changes, it will run with the default configuration.

Build the Project

Build the project by running:

```
./build.py build
```

- If Bluetooth feature is enabled, the firmware size will be much larger. Please make sure it does not exceed the ota partition size.
- After compiled, the combined factory bin will be created in `build/factory`. See *How to understand the differences of each type of module* for more information.

Flash onto the Device

Flash the binaries that you just built onto your ESP32 board by running:

```
./build.py -p (PORT) flash
```

- Note that you may need to replace `PORT` with your ESP device's serial port name.
- Or you can follow the printed instructions to flash the bin files into flash. Note that you may also need to replace the `PORT`.
- If the ESP-AT bin fails to boot and prints "ota data partition invalid", you should run `./build.py erase_flash` to erase the entire flash, and then re-flash the AT firmware.

5.1.3 ESP8266 Series

This section describes how to compile an ESP-AT project for ESP8266 series.

Get Started with ESP-IDF

Get started with ESP-IDF before compiling an ESP-AT project, because ESP-AT is developed based on ESP-IDF:

First, set up the development environment according to ESP8266 RTOS SDK Get Started Guide v3.4.

Then, start a simple project of `hello_world` according to the Guide to make sure your environment works and familiarize yourself with the process of starting a new project based on ESP-IDF. It is not a must, but you are strongly recommended to do so.

After setting up the development environment and starting a simple project, you can move onto the following steps that are ESP-AT specific.

Note: Please do not set `IDF_PATH` during the process, otherwise, you would encounter some unexpected issues when compiling ESP-AT projects later.

Get ESP-AT

To compile an ESP-AT project, you need the software libraries provided by Espressif in the ESP-AT repository.

To get ESP-AT, navigate to your installation directory and clone the repository with git clone, following instructions below specific to your operating system.

- Linux or macOS

```
cd ~/esp
git clone --recursive https://github.com/espressif/esp-at.git
```

- Windows

```
cd %userprofile%\esp
git clone --recursive https://github.com/espressif/esp-at.git
```

If you are located in China or have difficulties to access GitHub, you can also use `git clone https://gitee.com/EspressifSystems/esp-at.git` to get ESP-AT, which may be faster.

ESP-AT will be downloaded into `~/esp/esp-at` on Linux or macOS, or `%userprofile%\esp\esp-at` on Windows.

Note: This guide uses the directory `~/esp` on Linux and macOS or `%userprofile%\esp` on Windows as an installation folder for ESP-AT. You can use any directory, but you will need to adjust paths for the commands respectively. Keep in mind that ESP-AT does not support spaces in paths.

Connect Your Device

Connect your device to the PC with a USB cable to download firmware and log output. See *Hardware Connection* for more information. Note that you do not need to set up the "AT command/response" connection if you do not send AT commands and receive AT responses during the compiling process. You can change default port pins referring to *How to Set AT Port Pins*.

Configure

In this step, you will clone the `esp-idf` folder into the `esp-at` folder, set up the development environment in the newly cloned folder, and configure your project.

1. Navigate back to your `~/esp/esp-at` directory.
2. Run the project configuration utility `menuconfig` to configure.

```
./build.py menuconfig
```

3. Select the following configuration options for your ESP device if it is the first time you build the ESP-AT project.

- Select the `Platform name` for your ESP device, i.e., select `PLATFORM_ESP8266` for your ESP8266 device. `Platform name` is defined in factory_param_data.csv .
- Select the `Module name` for your ESP device. For example, select `WROOM-02` for the ESP-WROOM-02D module. `Module name` is defined in factory_param_data.csv .
- Enable or disable `silence mode`. If enabled, it will remove some logs and reduce the firmware size. Generally, it should be disabled.
- The above three option items will not appear if the file `build/module_info.json` exists. So please delete it if you want to reconfigure the module information.

4. Now, the `esp-idf` folder is created in `esp-at` folder. This `esp-idf` is different from that in Step **Get Started with ESP-IDF**.

- If the terminal prompt an error message like the following, please proceed with the next step to set up the develop environment in the `esp-at/esp-idf`.

```
The following Python requirements are not satisfied:
...
Please follow the instructions found in the "Set up the tools" section of ESP-IDF␣
↪Get Started Guide.
```

- If you have compiled an ESP-AT project for an ESP series before and want to switch to another series, you must run `rm -rf esp-idf` to remove the old esp/idf and then proceed with the next step.
- If your esp-idf is upgraded, you are recommended to proceed with the next step.

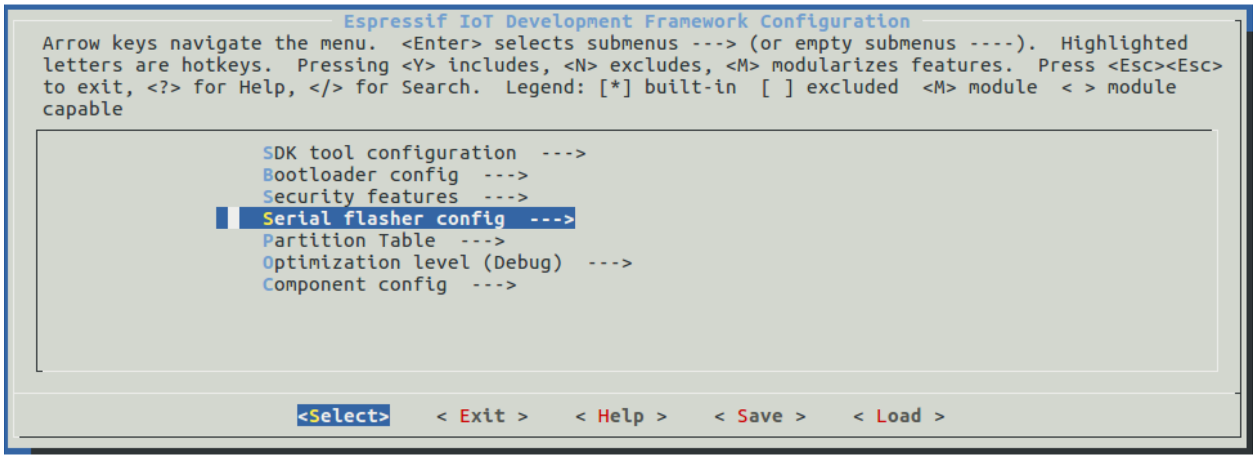
5. Set up the development environment in the `esp-at/esp-idf`.

- Set up the tools in the folder if it is the first time you compile an ESP-AT project. See *ESP8266 RTOS SDK Get Started Guide* v3.4 for more information.
- Set up environment variables in the folder **every time** you compile an ESP-AT project. See *ESP8266 RTOS SDK Get Started Guide* v3.4 for more information.
- **Install `pyyaml` and `xlrd` packages with pip in the folder if you have not done it**.

```
python -m pip install pyyaml xlrd
```

If the previous steps have been done correctly, the following menu appears after you run `./build.py menuconfig`:

You are using this menu to set up project-specific configuration, e.g. changing *AT port* pins, enabling Classic Bluetooth function, etc. If you made no changes, it will run with the default configuration.



```
Espressif IoT Development Framework Configuration
Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted
letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc>
to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > module
capable

                   SDK tool configuration  --->
                   Bootloader config  --->
                   Security features  --->
                   Serial flasher config  --->
                   Partition Table  --->
                   Optimization level (Debug)  --->
                   Component config  --->

            <Select>    < Exit >    < Help >    < Save >    < Load >
```

Fig. 2: Project configuration - Home window

Build the Project

Build the project by running:

```
./build.py build
```

- After compiled, the combined factory bin will be created in `build/factory`. By default, the factory bin is 2 MB flash size, DIO flash mode and 80 MHz flash speed. See *How to understand the differences of each type of module* for more information.

Flash onto the Device

Flash the binaries that you just built onto your ESP8266 board by running:

```
./build.py -p (PORT) flash
```

- Note that you may need to replace `PORT` with your ESP device's serial port name.
- Or you can follow the printed instructions to flash the bin files into flash. Note that you may also need to replace the `PORT`.
- If the ESP-AT bin fails to boot and prints "ota data partition invalid", you should run `./build.py erase_flash` to erase the entire flash, and then re-flash the AT firmware.

5.2 How to Set AT Port Pins

This document introduces how to modify *AT port* pins in the firmware for ESP32, ESP32-S2, ESP32-C3, and ESP8266 series of products. By default, ESP-AT uses two UART interfaces as AT ports: one is to output logs (named as log port below) and the other to send AT commands and receive responses (named as command port below).

To modify the AT port pins of your ESP device, you should:

- *clone the esp-at project*;
- modify the pins either in the menuconfig utility or the factory_param_data.csv table;

- *compile the project* to generate the new bin in `build/customized_partitions/factory_param.bin`;
- *flash the new bin to your device*.

This document focuses on modifying the pins. Click the links above for details of other steps. Below is the document structure:

- *ESP32 Series*
- *ESP32-S2 Series*
- *ESP32-C3 Series*
- *ESP8266 Series*

Note: To use other interfaces as the AT command port, please refer to AT through SDIO , AT through SPI , or AT through socket for more details.

5.2.1 ESP32 Series

The log port and command port pins of ESP32 AT firmware can be user-defined to other pins. Refer to ESP32 Technical Reference Manual for the pins you can use.

Modify Log Port Pins

By default, the ESP32 AT firmware provided by Espressif uses the following UART0 pins to output log:

- TX: GPIO1
- RX: GPIO3

When compiling your esp-at project, you can modify them to other pins with the menuconfig utitlity:

- `./build.py menuconfig -> Component config -> Common ESP-related -> UART for console output`
- `./build.py menuconfig -> Component config -> Common ESP-related -> UART TX on GPIO#`
- `./build.py menuconfig -> Component config -> Common ESP-related -> UART RX on GPIO#`

Modify Command Port Pins

By default, UART1 is used to send AT commands and receive AT responses, and its pins are defined in Column `uart_port`, `uart_tx_pin`, `uart_rx_pin`, `uart_cts_pin`, and `uart_rts_pin` of the factory_param_data.csv .

You can change them directly in your factory_param_data.csv table:

- Open your local factory_param_data.csv file.
- Locate the row of your module.
- Set `uart_port` as needed.
- Set `uart_tx_pin` and `uart_rx_pin` as needed.

- Set `uart_cts_pin` and `uart_rts_pin` to be -1 if you do not use the hardware flow control function.
- Save the table.

5.2.2 ESP32-S2 Series

The log port and command port pins of ESP32-S2 AT firmware can be user-defined to other pins. Refer to ESP32-S2 Technical Reference Manual for the pins you can use.

Modify Log Port Pins

By default, the ESP32-S2 AT firmware provided by Espressif uses the following UART0 pins to output log:

- TX: GPIO43
- RX: GPIO44

When compiling your esp-at project, you can modify them to other pins with the menuconfig utitlity:

- `./build.py menuconfig -> Component config -> Common ESP-related -> UART for console output`
- `./build.py menuconfig -> Component config -> Common ESP-related -> UART TX on GPIO#`
- `./build.py menuconfig -> Component config -> Common ESP-related -> UART RX on GPIO#`

Modify Command Port Pins

By default, UART1 is used to send AT commands and receive AT responses, and its pins are defined in Column `uart_port`, `uart_tx_pin`, `uart_rx_pin`, `uart_cts_pin`, and `uart_rts_pin` of the factory_param_data.csv .

You can change them directly in your factory_param_data.csv table:

- Open your local factory_param_data.csv file.
- Locate the row of your module.
- Set `uart_port` as needed.
- Set `uart_tx_pin` and `uart_rx_pin` as needed.
- Set `uart_cts_pin` and `uart_rts_pin` to be -1 if you do not use the hardware flow control function.
- Save the table.

5.2.3 ESP32-C3 Series

The log port and command port pins of ESP32-C3 AT firmware can be user-defined to other pins. ESP32-C3 Technical Reference Manual for the pins you can use.

Modify Log Port Pins

By default, the ESP32-C3 AT firmware provided by Espressif uses the following UART0 pins to output log:

- TX: GPIO21
- RX: GPIO20

When compiling your esp-at project, you can modify them to other pins with the menuconfig utitlity:

- `./build.py menuconfig -> Component config -> Common ESP-related -> UART for console output`
- `./build.py menuconfig -> Component config -> Common ESP-related -> UART TX on GPIO#`
- `./build.py menuconfig -> Component config -> Common ESP-related -> UART RX on GPIO#`

Modify Command Port Pins

By default, UART1 is used to send AT commands and receive AT responses, and its pins are defined in Column `uart_port`, `uart_tx_pin`, `uart_rx_pin`, `uart_cts_pin`, and `uart_rts_pin` of the factory_param_data.csv .

You can change them directly in your factory_param_data.csv table:

- Open your local factory_param_data.csv file.
- Locate the row of your module.
- Set `uart_port` as needed.
- Set `uart_tx_pin` and `uart_rx_pin` as needed.
- Set `uart_cts_pin` and `uart_rts_pin` to be -1 if you do not use the hardware flow control function.
- Save the table.

5.2.4 ESP8266 Series

The log port and command port pins of the ESP8266 AT firmware can be user-defined to other pins, but with limited options. Please refer to ESP8266 Technical Reference Manual for more details.

Modify Log Port Pins

By default, the ESP8266 AT firmware provided by Espressif uses UART1 to log output. UART1 only supports GPIO2 as the TX pin, and the pin should not be modified.

However, you could modify the log port from UART1 to UART0 by:

- `./build.py menuconfig -> Component config -> Common ESP-related -> UART for console output`

Modify Command Port Pins

By default, ESP8266 AT firmware uses UART0 to send AT commands and receive AT responses. The UART0 pins are defined in Column `uart_port`, `uart_tx_pin`, `uart_rx_pin`, `uart_cts_pin`, and `uart_rts_pin` of the factory_param_data.csv .

The UART pins can be changed, but there are only two choices: `GPIO15 as TX pin, GPIO13 as RX` or `GPIO1 as TX, GPIO3 as RX`. Below are the detailed steps:

- Open your local factory_param_data.csv file.
- Locate the row of your module.
- Set `uart_port` as needed.
- Set `uart_tx_pin` and `uart_rx_pin` to GPIO15 and GPIO13, or GPIO1 and GPIO3.
- Set `uart_cts_pin` and `uart_rts_pin` to be -1 if you do not use the hardware flow control function.
- Save the table.

For example, if you need to set GPIO1 (TX) and GPIO3 (RX) to be both the log port and command port of ESP-WROOM-02, do as follows:

1. Set log port to UART0: `./build.py menuconfig` -> `Component config` -> `ESP8266-specific` -> `UART for console output` -> `Default: UART0`
2. Open your local factory_param_data.csv.
3. Find the row of `WROOM-02`, set `uart_tx_pin` to 1, `uart_rx_pin` to 3, `uart_cts_pin` to -1, `uart_rts_pin` to -1, and then save the table.

5.3 How to add user-defined AT commands

This document details how to add a user-defined AT command based on the esp-at project. It uses the `AT+TEST` command as an example to show the sample code for each step.

Customizing a basic and well-functioned command requires at least the two steps below:

- *Define AT Commands*
- *Register AT Commands*

This step checks how the newly defined command works out.

- *Give it a try*

The remaining steps are for relatively complex AT commands and are optional depending on your needs.

- *Define Return Values*
- *Access Command Parameters*
- *Omit Command Parameters*
- *Block Command Execution*
- *Access Input Data from AT Command Port*

The source code of AT command set is not open-source, and is provided in the form of library file . It is also the basis to parse user-defined AT commands.

5.3.1 Define AT Commands

Before defining any AT command, you should first decide on the name and type of the AT command you want to define.

Naming rules:

- AT command should start with the + character.
- Alphabetic characters (`A~Z, a~z`), numeric characters (`0~9`), and some other characters (`!`, `%`, `-`, `.`, `/`, `:`, `_`) are supported. See *AT Command Types* for more information.

Command types:

Each AT command can have up to four types: Test Command, Query Command, Set Command, and Execute Command. See *AT Command Types* for more information.

Then, define each type of command. Assuming that `AT+TEST` supports all the four types. Below is the sample code to define each type.

Test Command:

```
uint8_t at_test_cmd_test(uint8_t *cmd_name)
{
    uint8_t buffer[64] = {0};

    snprintf((char *)buffer, 64, "this cmd is test cmd: %s\r\n", cmd_name);

    esp_at_port_write_data(buffer, strlen((char *)buffer));

    return ESP_AT_RESULT_CODE_OK;
}
```

Query Command:

```
uint8_t at_query_cmd_test(uint8_t *cmd_name)
{
    uint8_t buffer[64] = {0};

    snprintf((char *)buffer, 64, "this cmd is query cmd: %s\r\n", cmd_name);

    esp_at_port_write_data(buffer, strlen((char *)buffer));

    return ESP_AT_RESULT_CODE_OK;
}
```

Set Command:

```
uint8_t at_setup_cmd_test(uint8_t para_num)
{
    int32_t para_int_1 = 0;
    uint8_t *para_str_2 = NULL;
    uint8_t num_index = 0;
    uint8_t buffer[64] = {0};

    if (esp_at_get_para_as_digit(num_index++, &para_int_1) != ESP_AT_PARA_PARSE_
↪RESULT_OK) {
        return ESP_AT_RESULT_CODE_ERROR;
    }
```

(continues on next page)

(continued from previous page)

```
    if (esp_at_get_para_as_str(num_index++, &para_str_2) != ESP_AT_PARA_PARSE_RESULT_
↪OK) {
        return ESP_AT_RESULT_CODE_ERROR;
    }

    snprintf((char *)buffer, 64, "this cmd is setup cmd and cmd num is: %u\r\n", para_
↪num);
    esp_at_port_write_data(buffer, strlen((char *)buffer));

    memset(buffer, 0, 64);
    snprintf((char *)buffer, 64, "first parameter is: %d\r\n", para_int_1);
    esp_at_port_write_data(buffer, strlen((char *)buffer));

    memset(buffer, 0, 64);
    snprintf((char *)buffer, 64, "second parameter is: %s\r\n", para_str_2);
    esp_at_port_write_data(buffer, strlen((char *)buffer));

    return ESP_AT_RESULT_CODE_OK;
}
```

Execute Command:

```
uint8_t at_exe_cmd_test(uint8_t *cmd_name)
{
    uint8_t buffer[64] = {0};

    snprintf((char *)buffer, 64, "this cmd is execute cmd: %s\r\n", cmd_name);

    esp_at_port_write_data(buffer, strlen((char *)buffer));

    return ESP_AT_RESULT_CODE_OK;
}
```

Finally, call *esp_at_cmd_struct* to define the name and type(s) that your AT command supports. The sample code below defined the name +TEST (omitting AT) and all the four types.

```
static esp_at_cmd_struct at_custom_cmd[] = {
    {"+TEST", at_test_cmd_test, at_query_cmd_test, at_setup_cmd_test, at_exe_cmd_test}
↪,
};
```

If you do not want to define a particular type, set it to NULL.

5.3.2 Register AT Commands

Call API *esp_at_custom_cmd_array_regist()* to register your AT command. Below is the sample code to register AT+TEST:

```
esp_at_custom_cmd_array_regist(at_custom_cmd, sizeof(at_custom_cmd) / sizeof(at_
↪custom_cmd[0]));
```

Note: esp_at_custom_cmd_array_regist is recommended to be added to the at_custom_init() in app_main().

5.3.3 Give it a try

If you have finished the above two steps, the command should work after you build the esp-at project and flash the firmware to your device. Give it a try!

Below is how `AT+TEST` works out.

Test Command:

```
AT+TEST=?
```

Response:

```
AT+TEST=?
this cmd is test cmd: +TEST

OK
```

Query Command:

```
AT+TEST?
```

Response:

```
AT+TEST?
this cmd is query cmd: +TEST

OK
```

Set Command:

```
AT+TEST=1,"espressif"
```

Response:

```
AT+TEST=1,"espressif"
this cmd is setup cmd and cmd num is: 2
first parameter is: 1
second parameter is: espressif

OK
```

Execute Command:

```
AT+TEST
```

Response:

```
AT+TEST
this cmd is execute cmd: +TEST

OK
```

5.3.4 Define Return Values

`ESP-AT` has defined return values in *`esp_at_result_code_string_index`*. See *AT Messages* for more return values.

In addition to output return values through the return mode, you can also use API *esp_at_response_result()* to output the execution result of the command. *ESP_AT_RESULT_CODE_SEND_OK* and *ESP_AT_RESULT_CODE_SEND_FAIL* can be used with the API in code.

For example, when you send data to the server or MCU with the Execute Command of `AT+TEST`, you can use `esp_at_response_result` to output the sending result, and the return mode to output the command execution result. Below is the sample code:

```
uint8_t at_exe_cmd_test(uint8_t *cmd_name)
{
    uint8_t buffer[64] = {0};

    snprintf((char *)buffer, 64, "this cmd is execute cmd: %s\r\n", cmd_name);

    esp_at_port_write_data(buffer, strlen((char *)buffer));

    // user-defined operation of sending data to server or MCU
    send_data_to_server();

    // output SEND OK
    esp_at_response_result(ESP_AT_RESULT_CODE_SEND_OK);

    return ESP_AT_RESULT_CODE_OK;
}
```

How it works out:

```
AT+TEST
this cmd is execute cmd: +TEST

SEND OK

OK
```

5.3.5 Access Command Parameters

ESP-AT provides two APIs to access command parameters:

- *esp_at_get_para_as_digit()* obtains digital parameters.
- *esp_at_get_para_as_str()* obtains string parameters.

See *Set Command* for an example.

5.3.6 Omit Command Parameters

This section describes how to provide optional command parameters:

- *Omit the First or Middle Parameter*
- *Omit the Last Parameter*

Omit the First or Middle Parameter

Let's say you want to make `<param_2>` and `<param_3>` of `AT+TEST` optional. `<param_2>` is a digital parameter, and `<param_3>` a string parameter.

```
AT+TEST=<param_1>[,<param_2>][,<param_3>],<param_4>
```

Below is the sample code to achieve it:

```
uint8_t at_setup_cmd_test(uint8_t para_num)
{
    int32_t para_int_1 = 0;
    int32_t para_int_2 = 0;
    uint8_t *para_str_3 = NULL;
    uint8_t *para_str_4 = NULL;
    uint8_t num_index = 0;
    uint8_t buffer[64] = {0};
    esp_at_para_parse_result_type parse_result = ESP_AT_PARA_PARSE_RESULT_OK;

    snprintf((char *)buffer, 64, "this cmd is setup cmd and cmd num is: %u\r\n", para_
↪num);
    esp_at_port_write_data(buffer, strlen((char *)buffer));

    parse_result = esp_at_get_para_as_digit(num_index++, &para_int_1);
    if (parse_result != ESP_AT_PARA_PARSE_RESULT_OK) {
        return ESP_AT_RESULT_CODE_ERROR;
    } else {
        memset(buffer, 0, 64);
        snprintf((char *)buffer, 64, "first parameter is: %d\r\n", para_int_1);
        esp_at_port_write_data(buffer, strlen((char *)buffer));
    }

    parse_result = esp_at_get_para_as_digit(num_index++, &para_int_2);
    if (parse_result != ESP_AT_PARA_PARSE_RESULT_OMITTED) {
        if (parse_result != ESP_AT_PARA_PARSE_RESULT_OK) {
            return ESP_AT_RESULT_CODE_ERROR;
        } else {
            // sample code
            // user needs to customize the operation
            memset(buffer, 0, 64);
            snprintf((char *)buffer, 64, "second parameter is: %d\r\n", para_int_2);
            esp_at_port_write_data(buffer, strlen((char *)buffer));
        }
    } else {
        // sample code
        // the second parameter is omitted
        // user needs to customize the operation
        memset(buffer, 0, 64);
        snprintf((char *)buffer, 64, "second parameter is omitted\r\n");
        esp_at_port_write_data(buffer, strlen((char *)buffer));
    }

    parse_result = esp_at_get_para_as_str(num_index++, &para_str_3);
    if (parse_result != ESP_AT_PARA_PARSE_RESULT_OMITTED) {
        if (parse_result != ESP_AT_PARA_PARSE_RESULT_OK) {
            return ESP_AT_RESULT_CODE_ERROR;
        } else {
            // sample code
            // user needs to customize the operation
            memset(buffer, 0, 64);
            snprintf((char *)buffer, 64, "third parameter is: %s\r\n", para_str_3);
            esp_at_port_write_data(buffer, strlen((char *)buffer));
```

(continues on next page)

(continued from previous page)

```
        }
    } else {
        // sample code
        // the third parameter is omitted
        // user needs to customize the operation
        memset(buffer, 0, 64);
        snprintf((char *)buffer, 64, "third parameter is omitted\r\n");
        esp_at_port_write_data(buffer, strlen((char *)buffer));
    }

    parse_result = esp_at_get_para_as_str(num_index++, &para_str_4);
    if (parse_result != ESP_AT_PARA_PARSE_RESULT_OK) {
        return ESP_AT_RESULT_CODE_ERROR;
    } else {
        memset(buffer, 0, 64);
        snprintf((char *)buffer, 64, "fourth parameter is: %s\r\n", para_str_4);
        esp_at_port_write_data(buffer, strlen((char *)buffer));
    }

    return ESP_AT_RESULT_CODE_OK;
}
```

Note: If the string parameter input is `""`, it is not omitted.

5.3.7 Omit the Last Parameter

Let's say you want to make the string parameter `<param_3>` optional, which is also the last parameter.

```
AT+TEST=<param_1>,<param_2>[,<param_3>]
```

There are two cases of omission:

- AT+TEST=<param_1>,<param_2>
- AT+TEST=<param_1>,<param_2>,

Below is the sample code to achieve it:

```
uint8_t at_setup_cmd_test(uint8_t para_num)
{
    int32_t para_int_1 = 0;
    uint8_t *para_str_2 = NULL;
    uint8_t *para_str_3 = NULL;
    uint8_t num_index = 0;
    uint8_t buffer[64] = {0};
    esp_at_para_parse_result_type parse_result = ESP_AT_PARA_PARSE_RESULT_OK;

    snprintf((char *)buffer, 64, "this cmd is setup cmd and cmd num is: %u\r\n", para_
↪num);
    esp_at_port_write_data(buffer, strlen((char *)buffer));

    parse_result = esp_at_get_para_as_digit(num_index++, &para_int_1);
    if (parse_result != ESP_AT_PARA_PARSE_RESULT_OK) {
        return ESP_AT_RESULT_CODE_ERROR;
```

(continues on next page)

(continued from previous page)

```
    } else {
        memset(buffer, 0, 64);
        snprintf((char *)buffer, 64, "first parameter is: %d\r\n", para_int_1);
        esp_at_port_write_data(buffer, strlen((char *)buffer));
    }

    parse_result = esp_at_get_para_as_str(num_index++, &para_str_2);
    if (parse_result != ESP_AT_PARA_PARSE_RESULT_OK) {
        return ESP_AT_RESULT_CODE_ERROR;
    } else {
        memset(buffer, 0, 64);
        snprintf((char *)buffer, 64, "second parameter is: %s\r\n", para_str_2);
        esp_at_port_write_data(buffer, strlen((char *)buffer));
    }

    if (num_index == para_num) {
        memset(buffer, 0, 64);
        snprintf((char *)buffer, 64, "third parameter is omitted\r\n");
        esp_at_port_write_data(buffer, strlen((char *)buffer));
    } else {
        parse_result = esp_at_get_para_as_str(num_index++, &para_str_3);
        if (parse_result != ESP_AT_PARA_PARSE_RESULT_OMITTED) {
            if (parse_result != ESP_AT_PARA_PARSE_RESULT_OK) {
                return ESP_AT_RESULT_CODE_ERROR;
            } else {
                // sample code
                // user needs to customize the operation
                memset(buffer, 0, 64);
                snprintf((char *)buffer, 64, "third parameter is: %s\r\n", para_str_
↪3);
                esp_at_port_write_data(buffer, strlen((char *)buffer));
            }
        } else {
            // sample code
            // the third parameter is omitted
            // user needs to customize the operation
            memset(buffer, 0, 64);
            snprintf((char *)buffer, 64, "third parameter is omitted\r\n");
            esp_at_port_write_data(buffer, strlen((char *)buffer));
        }
    }

    return ESP_AT_RESULT_CODE_OK;
}
```

Note: If the string parameter input is `""`, it is not omitted.

5.3.8 Block Command Execution

Sometimes you want to block the execution of one command to wait for another execution result, and the system may return different values according to the result.

Generally, this kind of command needs to synchronize the results of other tasks.

semaphore is recommended to handle synchronization.

The sample code is as follows:

```
xSemaphoreHandle at_operation_sema = NULL;

uint8_t at_exe_cmd_test(uint8_t *cmd_name)
{
    uint8_t buffer[64] = {0};

    snprintf((char *)buffer, 64, "this cmd is execute cmd: %s\r\n", cmd_name);

    esp_at_port_write_data(buffer, strlen((char *)buffer));

    // sample code
    // users don't have to create semaphores here
    at_operation_sema = xSemaphoreCreateBinary();
    assert(at_operation_sema != NULL);

    // block command execution
    // wait for another execution result
    // other tasks can call xSemaphoreGive to release the semaphore
    xSemaphoreTake(at_operation_sema, portMAX_DELAY);

    return ESP_AT_RESULT_CODE_OK;
}
```

5.3.9 Access Input Data from AT Command Port

ESP-AT supports accessing input data from AT Command port. It provides two APIs for this purpose.

- *esp_at_port_enter_specific()* sets the callback function which will be called by AT port after receiving the input data.
- *esp_at_port_exit_specific()* deletes the callback function set by esp_at_port_enter_specific.

Approaches to access the data vary depending on whether the data length has been specified or not.

5.3.10 Input Data of Specified Length

Assuming that you have specified the data length in <param_1> as follows:

```
AT+TEST=<param_1>
```

Below is the sample to access the input data of <param_1> length from AT Command Port:

```
static xSemaphoreHandle at_sync_sema = NULL;

void wait_data_callback(void)
{
    xSemaphoreGive(at_sync_sema);
}

uint8_t at_setup_cmd_test(uint8_t para_num)
{
```

(continues on next page)

(continued from previous page)

```
    int32_t specified_len = 0;
    int32_t received_len = 0;
    int32_t remain_len = 0;
    uint8_t *buf = NULL;
    uint8_t buffer[64] = {0};

    if (esp_at_get_para_as_digit(0, &specified_len) != ESP_AT_PARA_PARSE_RESULT_OK) {
        return ESP_AT_RESULT_CODE_ERROR;
    }

    buf = (uint8_t *)malloc(specified_len);
    if (buf == NULL) {
        memset(buffer, 0, 64);
        snprintf((char *)buffer, 64, "malloc failed\r\n");
        esp_at_port_write_data(buffer, strlen((char *)buffer));
    }

    // sample code
    // users don't have to create semaphores here
    if (!at_sync_sema) {
        at_sync_sema = xSemaphoreCreateBinary();
        assert(at_sync_sema != NULL);
    }

    // output input prompt ">"
    esp_at_port_write_data((uint8_t *)">", strlen(">"));

    // set the callback function which will be called by AT port after receiving the␣
↪input data
    esp_at_port_enter_specific(wait_data_callback);

    // receie input data
    while(xSemaphoreTake(at_sync_sema, portMAX_DELAY)) {
        received_len += esp_at_port_read_data(buf + received_len, specified_len -␣
↪received_len);

        if (specified_len == received_len) {
            esp_at_port_exit_specific();

            // get the length of the remaining input data
            remain_len = esp_at_port_get_data_length();
            if (remain_len > 0) {
                // sample code
                // if the remaining data length > 0, the actual input data length is␣
↪greater than the specified received data length
                // users can customize the operation to process the remaining data
                // here is just a simple print out of the remaining data
                esp_at_port_recv_data_notify(remain_len, portMAX_DELAY);
            }

            // sample code
            // output received data
            memset(buffer, 0, 64);
            snprintf((char *)buffer, 64, "\r\nreceived data is: ");
            esp_at_port_write_data(buffer, strlen((char *)buffer));

            esp_at_port_write_data(buf, specified_len);
```

(continues on next page)

(continued from previous page)

```
                break;
            }
        }

        free(buf);

        return ESP_AT_RESULT_CODE_OK;
}
```

So, if you set `AT+TEST=5` and the input data is `1234567890`, the `ESP-AT` output is as follows.

```
AT+TEST=5
>67890
received data is: 12345
OK
```

5.3.11 Input Data of Unspecified Length

This scenario is similar to the Wi-Fi *Passthrough Mode*. You do not specify the data length.

```
AT+TEST
```

Assuming that `ESP-AT` ends the execution of the command and returns the execution result, the sample code is as follows:

```
#define BUFFER_LEN (2048)
static xSemaphoreHandle at_sync_sema = NULL;

void wait_data_callback(void)
{
    xSemaphoreGive(at_sync_sema);
}

uint8_t at_exe_cmd_test(uint8_t *cmd_name)
{
    int32_t received_len = 0;
    int32_t remain_len = 0;
    uint8_t *buf = NULL;
    uint8_t buffer[64] = {0};


    buf = (uint8_t *)malloc(BUFFER_LEN);
    if (buf == NULL) {
        memset(buffer, 0, 64);
        snprintf((char *)buffer, 64, "malloc failed\r\n");
        esp_at_port_write_data(buffer, strlen((char *)buffer));
    }

    // sample code
    // users don't have to create semaphores here
    if (!at_sync_sema) {
        at_sync_sema = xSemaphoreCreateBinary();
        assert(at_sync_sema != NULL);
    }
```

(continues on next page)

(continued from previous page)

```
    // output input prompt ">"
    esp_at_port_write_data((uint8_t *)">", strlen(">"));

    // set the callback function which will be called by AT port after receiving the␣
↪input data
    esp_at_port_enter_specific(wait_data_callback);

    // receie input data
    while(xSemaphoreTake(at_sync_sema, portMAX_DELAY)) {
        memset(buf, 0, BUFFER_LEN);

        received_len = esp_at_port_read_data(buf, BUFFER_LEN);
        // check whether to exit the mode
        // the exit condition is the "+++" string received
        if ((received_len == 3) && (strncmp((const char *)buf, "+++", 3)) == 0) {
            esp_at_port_exit_specific();

            // sample code
            // if the remaining data length > 0, it means that there is still data␣
↪left in the buffer to be processed
            // users can customize the operation to process the remaining data
            // here is just a simple print out of the remaining data
            remain_len = esp_at_port_get_data_length();
            if (remain_len > 0) {
                esp_at_port_recv_data_notify(remain_len, portMAX_DELAY);
            }

            break;
        } else if (received_len > 0) {
            // sample code
            // users can customize the operation to process the received data
            // here is just a simple print received data
            memset(buffer, 0, 64);
            snprintf((char *)buffer, 64, "\r\nreceived data is: ");
            esp_at_port_write_data(buffer, strlen((char *)buffer));

            esp_at_port_write_data(buf, strlen((char *)buf));
        }
    }

    free(buf);

    return ESP_AT_RESULT_CODE_OK;
}
```

So, if the first input data is `1234567890`, and the second input data is `+++`, the `ESP-AT` output is as follows:

```
AT+TEST
>
received data is: 1234567890
OK
```

5.4 How To Create Factory Parameter Bin

5.4.1 Overview

In order to adapt the AT firmware to different requirements, for example, different development board, different country code, different RF restriction, we make a table to configure those parameters.

5.4.2 Factory param type

The origin table is `components/customized_partitions/raw_data/factory_param/factory_param_type.csv`, and the factory parameter type is as the following table:

- description:
  - prompt information when build the project
- version:
  - the version of factory param mangement
- reserved1
  - reserved
- tx_max_power
  - Wi-Fi maximum tx power
    - For ESP32 Series chip, range[8, 78], more details please refer to ESP32 tx power setting range
    - For ESP8266 Series chip, range[-128, 127], more details please refer to ESP8266 tx power setting range
    - For ESP32S2 Series chip, range[8, 78], more details please refer to ESP32S2 tx power setting range
    - For ESP32-C3 Series chip, range[8, 78], more details please refer to ESP32-C3 tx power setting range
- start_chanel
  - Wi-Fi start channel
- channel_num
  - the total channel number of Wi-Fi
- country_code
  - Country code
- uart_baudrate
  - uart baudrate
- uart_tx_pin
  - uart tx pin
- uart_rx_pin
  - uart rx pin
- uart_cts_pin
  - uart cts pin, it can be configured -1, if the pin is not used
- uart_rts_pin

  - uart rts pin, it can be configured -1, if the pin is not used

- tx_control_pin
  - for some board, tx pin need to be separated from mcu when power on. It can be configured -1, if the pin is not used
- rx_control_pin
  - for some board, rx pin need to be separated from mcu when power on. It can be configured -1, if the pin is not used
- reserved2
  - reserved
- platform
  - Which platform the current firmware runs on
- module_name
  - Which module the current firmware runs on

5.4.3 Factory param data

The origin table is `components/customized_partitions/raw_data/factory_param/factory_param_data.csv`, and the information each row contains is about one module. The factory parameter data is as the following table:

5.4.4 Add customized module

if you want to add a module named as "MY_MODULE", of which country code is JP, and Wi-Fi channel is from 1 to 14, the table should be as the following one:

Then add module information in `esp_at_module_info` in `at_default_config.c`, like

```
static const esp_at_module_info_t esp_at_module_info[] = {
#if defined(CONFIG_IDF_TARGET_ESP32)
    {"WROOM-32",        CONFIG_ESP_AT_OTA_TOKEN_WROOM32,       CONFIG_ESP_AT_OTA_SSL_
↪TOKEN_WROOM32 },        // default:ESP32-WROOM-32
    {"WROOM-32",        CONFIG_ESP_AT_OTA_TOKEN_WROOM32,       CONFIG_ESP_AT_OTA_SSL_
↪TOKEN_WROOM32 },        // ESP32-WROOM-32
    {"WROVER-32",       CONFIG_ESP_AT_OTA_TOKEN_WROVER32,      CONFIG_ESP_AT_OTA_SSL_
↪TOKEN_WROVER32 },       // ESP32-WROVER
    {"PICO-D4",         CONFIG_ESP_AT_OTA_TOKEN_ESP32_PICO_D4, CONFIG_ESP_AT_OTA_SSL_
↪TOKEN_ESP32_PICO_D4},   // ESP32-PICO-D4
    {"SOLO-1",          CONFIG_ESP_AT_OTA_TOKEN_ESP32_SOLO_1,  CONFIG_ESP_AT_OTA_SSL_
↪TOKEN_ESP32_SOLO_1 },   // ESP32-SOLO-1
    {"MINI-1",          CONFIG_ESP_AT_OTA_TOKEN_ESP32_MINI_1,  CONFIG_ESP_AT_OTA_SSL_
↪TOKEN_ESP32_MINI_1 },   // ESP32-MINI-1
#endif

#if defined(CONFIG_IDF_TARGET_ESP8266)
    {"WROOM-02",        CONFIG_ESP_AT_OTA_TOKEN_WROOM_02,       CONFIG_ESP_AT_OTA_SSL_
↪TOKEN_WROOM_02 },
    {"WROOM-S2",        CONFIG_ESP_AT_OTA_TOKEN_WROOM_S2,       CONFIG_ESP_AT_OTA_SSL_
↪TOKEN_WROOM_S2 },
#endif
```

(continues on next page)

(continued from previous page)

```
#if defined(CONFIG_IDF_TARGET_ESP32S2)
    {"WROOM",         CONFIG_ESP_AT_OTA_TOKEN_ESP32S2_WROOM,       CONFIG_ESP_AT_OTA_
↪SSL_TOKEN_ESP32S2_WROOM },
    {"WROVER",        CONFIG_ESP_AT_OTA_TOKEN_ESP32S2_WROVER,      CONFIG_ESP_AT_OTA_
↪SSL_TOKEN_ESP32S2_WROVER },
    {"SOLO",          CONFIG_ESP_AT_OTA_TOKEN_ESP32S2_SOLO,        CONFIG_ESP_AT_OTA_
↪SSL_TOKEN_ESP32S2_SOLO },
    {"MINI",          CONFIG_ESP_AT_OTA_TOKEN_ESP32S2_MINI,        CONFIG_ESP_AT_OTA_
↪SSL_TOKEN_ESP32S2_MINI },
#endif

#if defined(CONFIG_IDF_TARGET_ESP32C3)
    {"MINI-1",          CONFIG_ESP_AT_OTA_TOKEN_ESP32C3_MINI,        CONFIG_ESP_AT_OTA_
↪SSL_TOKEN_ESP32C3_MINI },
#endif
};
```

5.4.5 Add customized data

If you want to add more parameter, for example, add a string "20181225" as the date, you need to add the type of date in the `factory_param_type.csv`, as the following table.

Edit `factory_param_data.csv` with reference to *Add customized module*, and add the date into the last column, as the following table,

It is important to know that the total size of the AT factory parameter is controlled by the `ESP_AT_FACTORY_PARAMETER_SIZE` in `at_default_config.h`, and can be adjusted as needed

Notes: It's recommended that do not change the first 2048 bytes, which may be used by Espressif.

Then, you can add code to parse `date` in `esp_at_factory_parameter_init` or other api.

5.4.6 Modify Factory param data

If you simply need to modify factory_param on an existing module, the following three methods are recommended:

- method one
  - Premise: you need to have the entire esp-at project.

1. Find the factory_param_data.csv file through the following path: `components/customized_partitions/raw_data/factory_param/factory_param_data.csv`, and modify the parameters.
2. Recompile the `esp-at` project, download the new `factory_param.bin` into flash.

- method two
  - Premise: you need to have the entire esp-at project.

1. Find the factory_param_data.csv file through the following path: `components/customized_partitions/raw_data/factory_param/factory_param_data.csv`, and modify the parameters.
2. Open the terminal in the following path: `esp-at`, execute the following command.

   - Commandline: `python tools/factory_param_generate.py --platform PLATFORM_ESP32S2 --module WROVER --define_file components/customized_partitions/raw_data/factory_param/factory_param_type.csv --module_file components/customized_partitions/raw_data/factory_param/factory_param_data.csv --bin_name factory_param.bin --log_file ./factory_parameter.log`
   - The value of the `-- platform -- module` parameter in the command needs to be changed as the case may be.

3. It will generate factory_param.bin at esp-at folder, download the new `factory_param.bin` into flash.
4. If you want to know how to use the commands in step 2, you can study the factory_param_generate.py file in the 'esp-at/tools".

- method three
   - Premise: you need to have the factory_param.bin file.

1. Open this file directly with a binary tool, and directly modify the parameters in the corresponding position according to the parameters offset in factory_param_type.csv.
2. Download the new `factory_param.bin` into flash.

5.5 How To Customize ble services

5.5.1 Where is the `BLE Services` source file

The path of BLE service source file is `esp-at/components/customized_partitions/raw_data/ble_data/example.csv`. If user wants to customize the BLE services, You need to:

- Modify the `BLE Service` file
- use esp-at/tools/BLEService.py to generate ble_data.bin
- Download generated ble_data.bin to address defined in `module_config/module_esp32_default/partitions_at.csv`(ESP32 modules).

5.5.2 Description of the structure of the BLE service

The BLE Services are defined as a multivariate array of GATT structures, each element of the array always consist of a service, declarations, characteristics and optional descriptors.

User can define more than one services. For example, if you want to define three services(`Server_A`, `Server_B` and `Server_C`), then these three services need to be arranged in order. Since the definition of each service is similar, here we define one service as example, and then you can define others one by one accordingly.

Each service always consist of a service definition and several characteristics. Each characteristic may be followed by some descriptions.

the service definition must be in the first line, it always be a primary service (UUID 0x2800) which determines with its value which service is described(for example, a predefined one such as 0x180A or a self generated one).

- For example, the following line defines a primary service with UUID 0xA002.

Definition of characteristics starts from the second line. It contains at least two lines, one is the characteristic declaration, another is to define the characteristic. UUID 0x2803 means the characteristic declaration, value of this line sets its permission, for example, 02 means both readable and writable, user can keep this configuration. Then the next line defines this characteristic, UUID of this line will be the characteristic's UUID, you can define it as you need, value will be the characteristic's value.

- For example, the following lines define a readable and writable characteristic with UUID 0xC300, whose value is 0x30.

The attribute can be described further by descriptors. A special one is the descriptor "Client Characteristic Configuration" (UUID 0x2902) which should be present if the Notify bit has been activated in the Characteristic Declaration (UUID 0x2803). This descriptor should always be writable and readable.

- For example, the following lines define a readable and writable characteristic with UUID 0xC306, and able to notify.

5.6 The Secondary Partitions Table

5.6.1 Overview

The primary partition table is for system usage, it will generate a "partitions_at.bin" according to the "partitions_at.csv" in compilation. And if the primary partition table goes wrong, the system will fail to startup. So generally, we should not change the "partitions_at.csv".

In this case, we provide a secondary partition table for custom usage, "at_customize.csv". We have already defined some user partitions in it. Custom can add new partitions in the "at_customize.csv", and generate a new "at_customize.bin" according to it. The partition table can be updated by flashing the new "at_customize.bin" into flash, or be revised by command "AT+SYSFLASH".

5.6.2 How to use the secondary partition table

- Enter the project esp32-at, run command `python esp-idf/components/partition_table/gen_esp32part.py -q at_customize.csv at_customize.bin` to generate a "at_customize.bin".
- Download the "at_customize.bin" into flash, the default address is 0x20000.

5.6.3 Notes of revising at_customize.csv

- Users should not change the "name" and "type" of the user partitions which we defined in the "at_customize.csv". But "offset" and "size" can be changed, if necessary.
- If you need to add a new user partition, please check if it has already been defined in the ESP-IDF (esp_partition.h) first.
  - If it is defined in the ESP-IDF, you should keep the "type" value to be the same when adding it into the at_customize.csv.
  - If it is not defined in the ESP-IDF, please set the "type" to be "0x40" when adding it into the at_customize.csv.
- A user partition's name should NOT be longer than 16 bytes.
- The default size of the entire "at_customize" partition is 0xE0000, which is defined in the first partition. Please do NOT over the range when adding new user partitions.

5.6.4 When a "at_customize.bin" is needed

To use below AT commands, the "at_customize.bin" should be downloaded into flash.

- AT+SYSFLASH — Set User Partitions in Flash
- AT+FS — Filesystem Operations
- SSL server relevant commands
- BLE server relevant commands

5.7 How to use ESP-AT Classic Bluetooth

5.7.1 Overview

Classic bluetooth is Disabled by default. If you want to use classic bluetooth commands, you need to enable BT commands in menuconfig.

```
Component config -> AT -> [*] AT bt command support.
```

5.7.2 Command Description

initialization

There are two initialization-related commands. Firstly, initializing the bluetooth protocol stack, and then initializing the profile, such as:

```
AT+BTINIT=1       // init BT statck
AT+BTSPPINIT=2    // init SPP profile, the role is slave
```

Basic parameters setting

After initialization, there are some basic parameter setting commands that may be need to be invoked.

1. device name

The default device name is `esp32`, If use command to set the device name, it will be stored in NVS.

```
AT+BTNAME="EXAMPLE"
```

2. scan mode

Sets whether it can be discovered and connected.

```
AT+BTSCANMODE=2     // both discoverable and connectable
```

3. security parameters

ESP32 supports both Simple pair and Legacy pair by default.

Using this command, you can set the IO capability, PIN type and PIN code of the device.

```
AT+BTSECPARAM=3,1,"9527"  // NO input NO output, fixed PIN code, 9527
```

If the PIN type is variable, the PIN code will be ignored. If use the Simple pair encryption, the PIn code will be ignored.

BT SPP EXAMPLE

1. PC CONNECTS TO ESP32

In this case, generally PC is master and ESP32 is slave. ESP32 needs to do this before the connection is established:

- initialization

```
AT+BTINIT=1     // init BT stack
AT+BTSPPINIT=2  // init spp profile as shave
AT+BTSPPSTART   // if role is client, this command is not required
```

- parameters setting

```
AT+BTNAME="EXAMPLE"       // set device name
AT+BTSCANMODE=2           // discoverable and connectable
AT+BTSECPARAM=3,1,"9527"  // NoInputNoOutput, fixed PIN code
```

At this point, the PC should be able to find the bluetooth device with name "EXAMPLE". If the PC initiates a connection and the connection succeed, ESP32 will print this log:

```
+BTSPPCONN:<conn index>,<remote_addr>
```

ESP32 can also initiate connections, before initiating a connection, please scan the surrounding devices at first:

```
// General inquiry mode, inquiry duration: 10, inquiry response: 10
AT+BTSTARTDISC=0,10,10
```

The scanning results are as follows:

```
+BTSTARTDISC:<bt_addr>,<dev_name>,<major_dev_class>,<minor_dev_class>,<major_srv_
↪class>
```

Initiate the connection using the connection command

```
// conn_index: 0, sec_mode: 0 -> No security, remote_address
AT+BTSPPCONN=0,0,"24:0a:c4:09:34:23"
```

After the connection is established, the data can be sent and received

```
// conn_index: 0, data length: 30
AT+BTSPPSEND=0,30

>
OK
```

If you want to enter passthrough mode:

```
AT+BTSPPSEND
```

If you want to exit passthrough mode, you can input +++.

ESP32 CONNECTS TO ESP32

If you use two ESP32 boards connected to each otherThe process is basically the same as described above, The only difference is the initialization. the client initialization is as follow:

```
AT+BTINIT=1     // init BT stack
AT+BTSPPINIT=1  // init spp profile as master
```

All other steps are the same as described above.

Encryption-related operation

If the IO capability is not NoInputNoOutput, the encryption process will involve the exchange of key and PIN code.

If need to input the PIN code for Legacy Pair:

```
// conn_index, PIN code
AT+BTPINREPLY=0,"0000"
```

If need to input the Simple Pair Key:

```
// conn_index, Key
AT+BTKEYREPLY=0,123456
```

If ESP32 has the ability to output, You need to enter this password on the remote device

```
+BTKEYNTF:0,123456
```

The ESP32 can also choose to accept or reject the encryption request from the remote device:

```
// conn_index, accept or reject
AT+BTSECCFM=0,1
```

There are also two commands to manage bonding devices:

```
//get the bonded devices list
AT+BTENCDEV?
//remove a device from the security database list with a specific index.
AT+BTENCCLEAR=<enc_dev_index>
//remove all devices from the security database
AT+BLEENCCLEAR
```

5.8 How to enable ESP-AT Ethernet

5.8.1 Overview

Initialises the Ethernet interface and enables it, then sends DHCP requests and tries to obtain a DHCP lease. If successful then you will be able to ping the device.

5.8.2 PHY Configuration

Use `./build.py menuconfig` to set the PHY model. These configuration items will vary depending on the hardware configuration you are using.

The default configuration is correct for Espressif's Ethernet board with TP101 PHY. ESP32 AT supports up to four Ethernet PHY: `LAN8720`, `IP101`, `DP83848` and `RTL8201`. `TLK110` PHY is no longer supported because TI stoped production. If you want to use other phy, follow the document to design.

Compiling

1. `./build.py menuconfig` -> `Component config` -> `AT` -> `AT ethernet support` to enable ethernet.
2. `./build.py menuconfig` -> `Component config` -> `AT` -> `Ethernet PHY` to choose ethernet.
3. Recompile the `esp-at` project, download AT bin into flash.

5.9 How to Add a New Platform

Since the esp-at project supports different platform, for example, ESP32 UART AT, ESP32 SDIO AT, even supports ESP8266 AT, when compiling the esp-at project, users can set different configurations to generate AT firmware for different ESP modules. The detailed information are in the `esp-at/module_config` directory. Default configuration is the "PLATFORM_ESP32" for ESP-WROOM-32.Please note that if you use a different bus (for example, SDIO, SPI or other buses) instead of UART, then you cannot use the default `module_espxxxx_default` directly, you need to re-configure it in the `menuconfig`.Herein, we provide an example of the ESP32 SDIO AT to show how to set a new platform for the esp-at project.

5.9.1 1. Create a New Platform

For example, to name the platform as "PLATFORM_ESP32", the module as "WROOM32-SDIO", we need to open the `components/customized_partitions/raw_data/factory_param/factory_param_data.csv` and add a new row of the new platform at the end.

5.9.2 2. Set Makefile

Open the `Makefile` and set to the new platform. Please use capital letters.

```
export ESP_AT_PROJECT_PLATFORM ?= PLATFORM_ESP32
export ESP_AT_MODULE_NAME ?= ESP32-SDIO
```

5.9.3 3. Configure the New Platform

- 3.1. Enter `module_config` folder, copy the `module_esp32_default` to make a new `module_esp32-sdio`.
- 3.2. In this example, we need not to change the partition table and the ESP-IDF version, so the `at_customize.csv`, `IDF_VERSION` and `partitions_at.csv` all need not to be changed.
- 3.3. Revise the `sdkconfig.defaults`
  - Configure to use the partition table in the `module_esp32-sdio` folder, revise the following items

```
CONFIG_PARTITION_TABLE_CUSTOM_FILENAME="module_config/module_esp32-sdio/
↪partitions_at.csv"

CONFIG_PARTITION_TABLE_FILENAME="module_config/module_esp32-sdio/partitions_at.csv
↪"

CONFIG_AT_CUSTOMIZED_PARTITION_TABLE_FILE="module_config/module_esp32-sdio/at_
↪customize.csv"
```

- Since the esp-at project already supports the SDIO configuration, we only need to add it into the `sdkconfig.defaults`.
  - Remove the UART AT configuration in the `sdkconfig.defaults`.

```
CONFIG_AT_BASE_ON_UART=y
CONFIG_AT_UART_PORT=1
CONFIG_AT_UART_PORT_RX_PIN=16
CONFIG_AT_UART_PORT_TX_PIN=17
CONFIG_AT_UART_PORT_RTS_PIN=14
CONFIG_AT_UART_PORT_CTS_PIN=15
```

  And add the following configuration.

```
CONFIG_AT_BASE_ON_SDIO=y
```

5.9.4 4. Revise the at_core lib

Since the ESP32 SDIO AT and the ESP32 UART AT are based on the same platform, ESP32, they will share the same at_core.lib. In this case, we need not to add any new at_core lib.If you need to use a new at_core lib, put the lib into the `components/at/lib`, rename the lib as `libxxxx_at_core.a`, `xxxx` is the platform name. For example, if you set the `ESP_AT_PROJECT_PLATFORM ?= PLATFORM_ESP8848` in the `Makefile` in Step 2, then the lib should be named as `libesp8848_at_core.a`.

5.10 ESP32 SDIO AT Guide

5.10.1 Overview

SDIO AT is based on ESP32 AT, using SDIO, instead of UART, to communicate with host MCU. The ESP32 SDIO AT runs as an SDIO slave.SDIO communication needs at least 4 pins: CMD, CLK, DAT0, DAT1;

- for one line mode, DAT1 runs as the interrupt pin;
- for four lines mode, two more pins (DAT2 and DAT3) are needed.

SDIO SLAVE pins is as below:

- CLK is GPIO14
- CMD is GPIO15
- DAT0 is GPIO2
- DAT1 is GPIO4
- DAT2 is GPIO12 (for four lines mode only)
- DAT3 is GPIO13 (for four lines mode only)

5.10.2 Flashing Firmware

ESP-SDIO-TESTBOARD-V1

1. Turn switch 1, 2, 3, 4, 5 "ON"; and others are "OFF".
2. Flashing firmware to the master. After the flashing completed, the light on ESP32 slave will turn on, it means that the master runs successfully, and power on the slave.
3. Flashing the SDIO AT firmware to the slave.

Note:If you use ESP32-DevKitC or ESP-WROVER-KIT V2 (or earlier versions), please refer to the board-compatibility to set strapping pins, and run the SDIO demo firstly, to make sure the SDIO communication works properly. If it is, then you can start your SDIO AT journey.

5.10.3 SDIO Data Transimission

SDIO HOST

1. Power on the SDIO SLAVE (this step is for ESP-SDIO-TESTBOARD-V1 only)
   - ESP-SDIO-TESTBOARD-V1 contains one master and three slaves (ESP32, ESP8266 and ESP8089).
   - To use ESP32 as SDIO slave, you need to pull down GPIO5, see `slave_power_on`.
2. Intialize SDIO HOST
   - To initialize SDIO, including configure one line or four lines mode, SDIO frequency, initialize SD mode.
3. Negotiate SDIO Communication
   - Negotiate SDIO parameters with the slave according to SDIO spec. Please note that SDIO HOST must wait till the slave startup done, then to start the negotiation. Usually the host needs to delay some time to wait.
4. Receive Data
   - When the SDIO host detects an interrupt from DAT1, if it is the slave sends new packet to the host, the host will read those data by CMD53.
5. Send Data
   - In SDIO AT demo, the data inputs from serial port, when the SDIO host gets it, the host will send it to the slave through SDIO by CMD53.
   - Please note that if the sending time out, there may be something wrong with the slave, then we will re-initiate both SDIO host and slave.
   - Also, after AT+RST or AT+RESTORE, both SDIO host and slave should be initiated again.

SDIO SLAVE

When SDIO slave receives data from SDIO host, the slave will inform the AT core and give the data to the AT core to handle. After the AT core finished the work, the slave will send data to the host as feedback.

1. Initialize SDIO Slave
   - Call `sdio_slave_initialize` to initialize SDIO slave driver
   - Call `sdio_slave_recv_register_buf` to register receiving buffer. To make it receive data faster, you can register multiple buffers.

- Call `sdio_slave_recv_load_buf` to load the receiving buffer, ready to receive data.
- Call `sdio_slave_set_host_intena` to set interrupt for host, which mainly used is the `SDIO_SLAVE_HOSTINT_SEND_NEW_PACKET`, to notify that there is a new packet sent from the host.
- Call `sdio_slave_start` to start SDIO hardware transmission.

2. Send Data
    - Since the SDIO slave data transmission via DMA, you need to copy the data from AT core to the memory which DMA can read firstly.
    - Call `sdio_slave_transmit` to send the data.
3. Receive Data
    - To speed up the data transmission, after receiving data by `sdio_slave_recv`, we use a circular linked list to transmit the received data to the AT core.

5.11 SPI AT Guide

This document mainly introduces the implementation and use of SPI AT, mainly involving the following aspects:

Note: This document mainly introduces the implementation and use of SPI AT for ESP32-C and ESP32-S series. For ESP32 series, please refer to ESP32 SPI AT.

5.11.1 Overview

SPI AT is based on ESP-AT project and uses SPI protocol for data communication. When SPI AT is used, MCU works as SPI master and ESP AT device works as SPI slave. Both sides of communication exchange data based on AT command through SPI protocol.

Why SPI AT?

ESP-AT project uses UART protocol for data communication by default, but UART protocol is not applicable in some application scenarios that need high-speed data transmission. In this case, the SPI protocol which supports a higher transmission rate is a better choice.

How to enable SPI AT?

You can configure and enable SPI AT through the following steps:

1. `./build.py menuconfig` -> `Component config` -> `AT` -> `communicate method for AT command` -> `AT through HSPI` to enable SPI AT.

2. `./build.py menuconfig -> Component config -> AT -> communicate method for AT command` -> `AT SPI Data Transmission Mode` to choose the SPI data transmission mode.
3. `./build.py menuconfig -> Component config -> AT -> communicate method for AT command` -> `AT SPI GPIO settings` to change the default pin assignments for SPI AT.
4. `./build.py menuconfig -> Component config -> AT -> communicate method for AT command` -> `AT SPI driver settings` to choose the SPI slave mode, and config the buffer size for data transmission.
5. Recompile the `esp-at` project(see *Build Your Own ESP-AT Project*), download AT bin into flash.

The Default Pin Assignment

The following pin assignments are used by default:

Table 2: The Default Pins for SPI AT

Signal	GPIO Number (ESP32-C3)	GPIO Number (ESP32-S2)
SCLK	6	12
MISO	2	13
MOSI	7	11
CS	10	10
HANDSHAKE	3	5
GND	GND	GND
QUADWP (qio/qout):sup:*1*	8	16
QUADHD (qio/qout):sup:*1*	9	17

Note 1: QUADWP and QUADHD signals are only used for 4-bit (qio/qout) transactions.

You can change the default pin assignments by `./build.py menuconfig` > `Component config` > `AT` > `communicate method for AT command` > `AT through HSPI` > `AT SPI GPIO settings` and compile the project (see *Build Your Own ESP-AT Project*).

5.11.2 How to Use SPI AT?

When SPI AT is usedESP device works in SPI half-duplex mode as a slave. Usually, when SPI protocol works in half-duplex mode, it is the SPI master that starts the read/write operation. However, when AT command is used for data interaction, ESP device (slave) is required to actively report some information. Therefore, we add a handshake line between SPI master and slave to realize the purpose. The specific methods of using handshake line are as follows:

- When master sends AT commands to slave via SPI, the workflow with an extra handshake line is as follows:
  1. The master sends a request for data transmission to the slave, and then waits for the signal sent by the slave to the handshake line to allow data transmission.
  2. After the master detects the permission signal sent by the slave on the handshake line, it starts to send data.
  3. After sending the data, the master notifies slave that the data transmission is finished.
- When slave sends AT responses to master via SPI, the workflow with an extra handshake line is as follows:
  1. The slave sends a signal through the handshake line to inform the master to start receiving data.
  2. The master receives data and notifies slave that the data transmission is finished after receiving all data.

Communication Formats

The communication format used by SPI AT is CMD+ADDR+DUMMY+DATA (Read or Write). When using SPI AT, some communication messages used by SPI master are described as follows:

- The message used by the master to send data to slave:

Table 3: Master write data to slave

CMD (1 byte)	ADDR (1 byte)	DUMMY (1 byte)	DATA (Up to 4092 bytes)
0x3	0x0	0x0	data_buffer

- The message used by the master to inform the slave all data has been sent:

Table 4: Master write data done

CMD (1 byte)	ADDR (1 byte)	DUMMY (1 byte)	DATA
0x7	0x0	0x0	null

- The message used by the master to receive data from the slave:

Table 5: Master read data from slave

CMD (1 byte)	ADDR (1 byte)	DUMMY (1 byte)	DATA (Up to 4092 bytes)
0x4	0x0	0x0	data_buffer

- The message used by the master to inform the slave all data has been received:

Table 6: Master read data done

CMD (1 byte)	ADDR (1 byte)	DUMMY (1 byte)	DATA
0x8	0x0	0x0	null

- The message used by the master to send a request to send data:

Table 7: Master request to send data

CMD (1 byte)	ADDR (1 byte)	DUMMY (1 byte)	DATA (4 bytes)
0x1	0x0	0x0	data_info

The 4-byte length `data_info` contains the information about the packet to be sent. The specific format is as follows:

1. 0~15 bits: the length of the data that the master want to send to the slave.
2. 16~23 bits: the sequence number of the packet sent by the master to the slave.
3. 24~31 bits: the magic num, and default value is 0xFE.

- After receiving the signal from the handshake line, the master can send the message to query the read or write status of the slave:

Table 8: Master query the read/write status of the slave

CMD (1 byte)	ADDR (1 byte)	DUMMY (1 byte)	DATA (4 bytes)
0x2	0x4	0x0	slave_status

After sending the query request, the slave's status returned will be stored in the 4-byte length `slave_status`, the specific format is as follows:

1. 0~15 bits: the length of the data the slave want to sent to the master. This field is valid only when the slave is readable.
2. 16~23 bits: the sequence number of the packet to been sent. The maximum sequence number is 0xFF, and if the number is reached, the sequence number is incremented by 1 from 0.
3. 24~31 bits: the slave status(readable/writable).

SPI AT Workflow

The workflows mainly includes two aspects:

- When master sends AT commands to slave, the workflow is as follows:

```
| SPI master |                            | SPI slave |
       |                                       |
       |   -------step 1: request to send---->  |
       |                                       |
       |   <------step 2: GPIO interrupt------  |
       |                                       |
       |   -------step 3: read slave status-->  |
       |                                       |
       |   -------step 4: send data---------->  |
       |                                       |
       |   -------step 5: send done---------->  |
```

The specific description of each step is as follows:

step 1. The master sends a request for data transmission to the slave. step 2. The slave receives the request from the master. If the master is allowed to send data, the slave sets the status register, and then triggers the GPIO interrupt on the master through the handshake line. step 3. After receiving the interrupt, master will query the status register of slave. If the query result shows that the slave is in the writable state, the master starts to send data. step 4. The master send data to the slave. step 5. After sending the data, the master notifies slave that the data transmission is finished.

- When master receives AT responses from slave, the workflow is as follows:

```
| SPI master |                            | SPI slave |
       |                                       |
       |   <------step 1: GPIO interrupt------  |
       |                                       |
       |   -------step 2: read slave status-->  |
       |                                       |
       |   <------step 3: send data-----------  |
       |                                       |
       |   -------step 4: receive done------->  |
```

The specific description of each step is as follows:

step 1. The slave sets the status register, and then triggers the GPIO interrupt on the master through the handshake line. step 2. After receiving the interrupt, master will query the status register of slave. If the query result shows that the slave is in the readable state, the master starts to receive data. step 3. The master receives the data send by the slave. step 4. After receiving all data, the master notifies the slave that the data transmission is finished.

Sample Code of SPI AT Master

A code example of SPI AT master can be found under the directory AT ESP32 SPI Master Example.

5.11.3 SPI AT Throughput

Introduction of the Test

- An ESP32-DevKitC development board is been used as SPI master. The application runs in the board can be found under the directory at_spi-master/spi/esp32_c_series of the ESP-AT project. Some related configurations are described below:

1. Hardware configuration: The frequency of CPU is 240 MHzflash SPI mode is in QIO mode with 40 MHz.
2. Software configuration: The ESP-IDF version is v4.3. The size of streambuffer is 8192 bytes.

- An ESP32-C3-DevKitC development board is been used as SPI slave. Please refer to *Build Your Own ESP-AT Project* to build your own ESP-AT project and flash the generated binary files into the board. The board works in the TCP passthrough mode, and some related configurations are described below:

1. Hardware configuration: The frequency of CPU is 160 MHz.
2. Software configuration: The size of streambuffer is 8192 bytes, the sdkconfig is sdkconfig.defaults.esp32c3.

Reference Results

The table below shows the throughput results we got in a shield box.

Table 9: SPI AT Wi-Fi TCP Throughput

Clock	SPI mode	master->slave	slave->master
10 M	Standard	0.95 MByte/s	1.00 MByte/s
10 M	Dual	1.37 MByte/s	1.29 MByte/s
10 M	Quad	1.43 MByte/s	1.31 MByte/s
20 M	Standard	1.41 MByte/s	1.30 MByte/s
20 M	Dual	1.39 MByte/s	1.30 MByte/s
20 M	Quad	1.39 MByte/s	1.30 MByte/s
40 M	Standard	1.37 MByte/s	1.30 MByte/s
40 M	Dual	1.40 MByte/s	1.31 MByte/s
40 M	Quad	1.48 MByte/s	1.31 MByte/s

Note 1: When SPI clock frequency is high, due to the limitation of upper network components, the communication rate of Dual or Quad mode is not significantly improved compared with Standard mode.

Note 2: For more information about SPI communication, please refer to the Technical Reference Manuals.

5.12 How to implement OTA update

The following steps guide the users in creating a device on iot.espressif.cn and updating the OTA BIN on it.

1. Open the website http://iot.espressif.cn. If using SSL OTA, it should be https://iot.espressif.cn.
2. Click "Join" in the upper right corner of the webpage, and enter your name, email address, and password.
3. Click on "Device" in the upper right corner of the webpage, and click on "Create" to create a device.

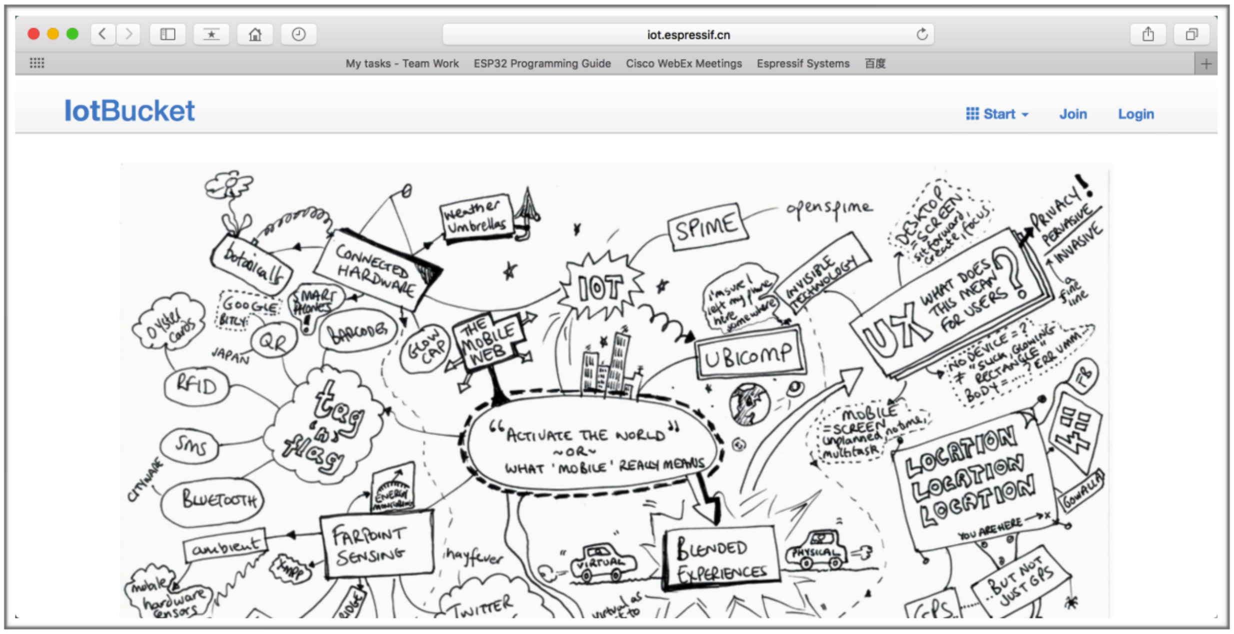
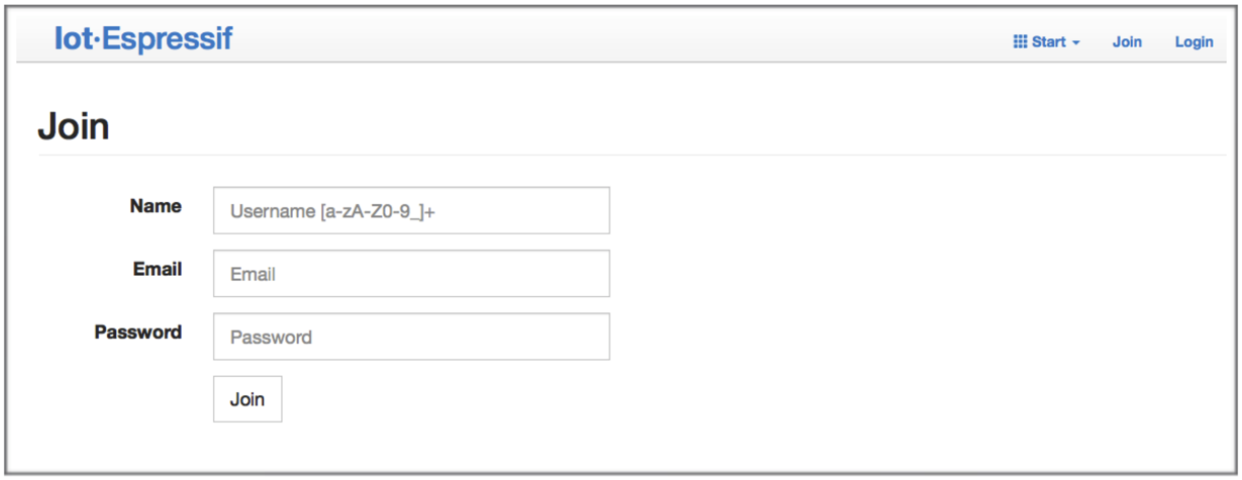


Fig. 3: Open iot.espressif.cn website



Join

Name	Username [a-zA-Z0-9_]+
Email	Email
Password	Password
	Join

Fig. 4: Join iot.espressif.cn website

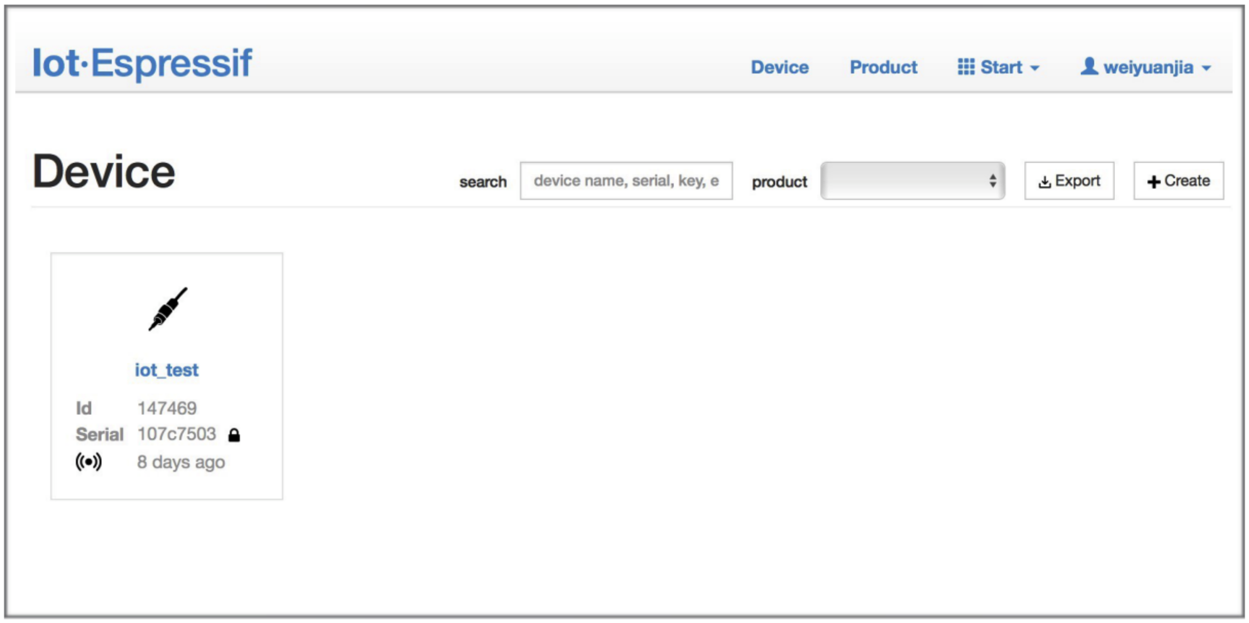


Fig. 5: Click on "Device"

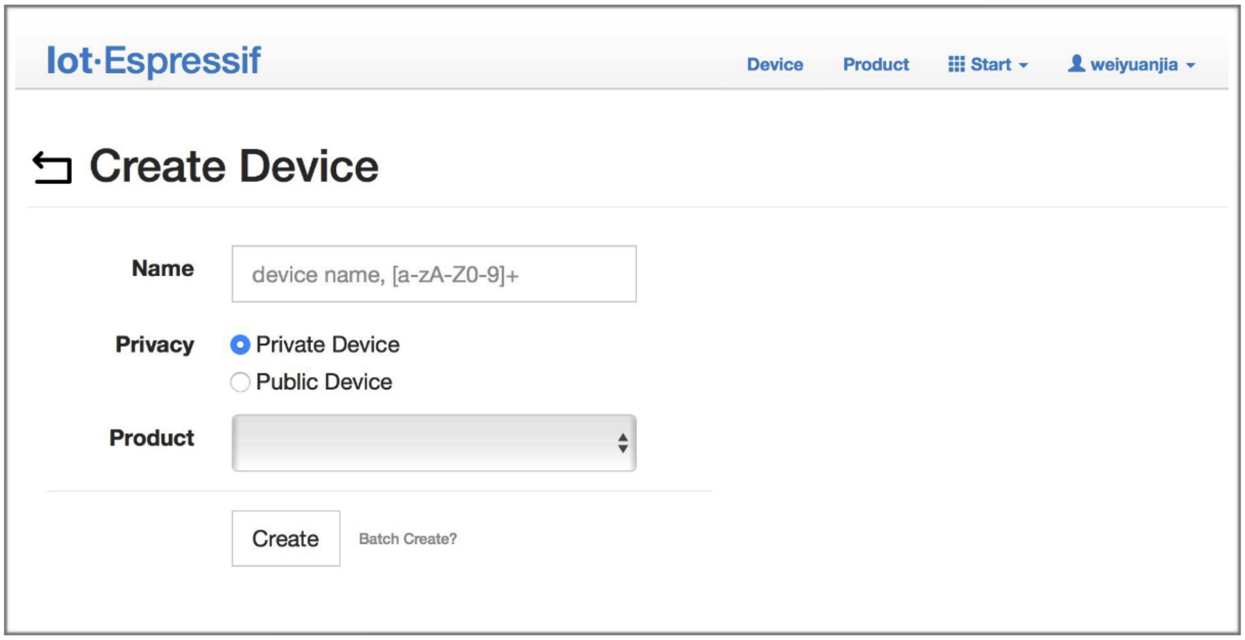
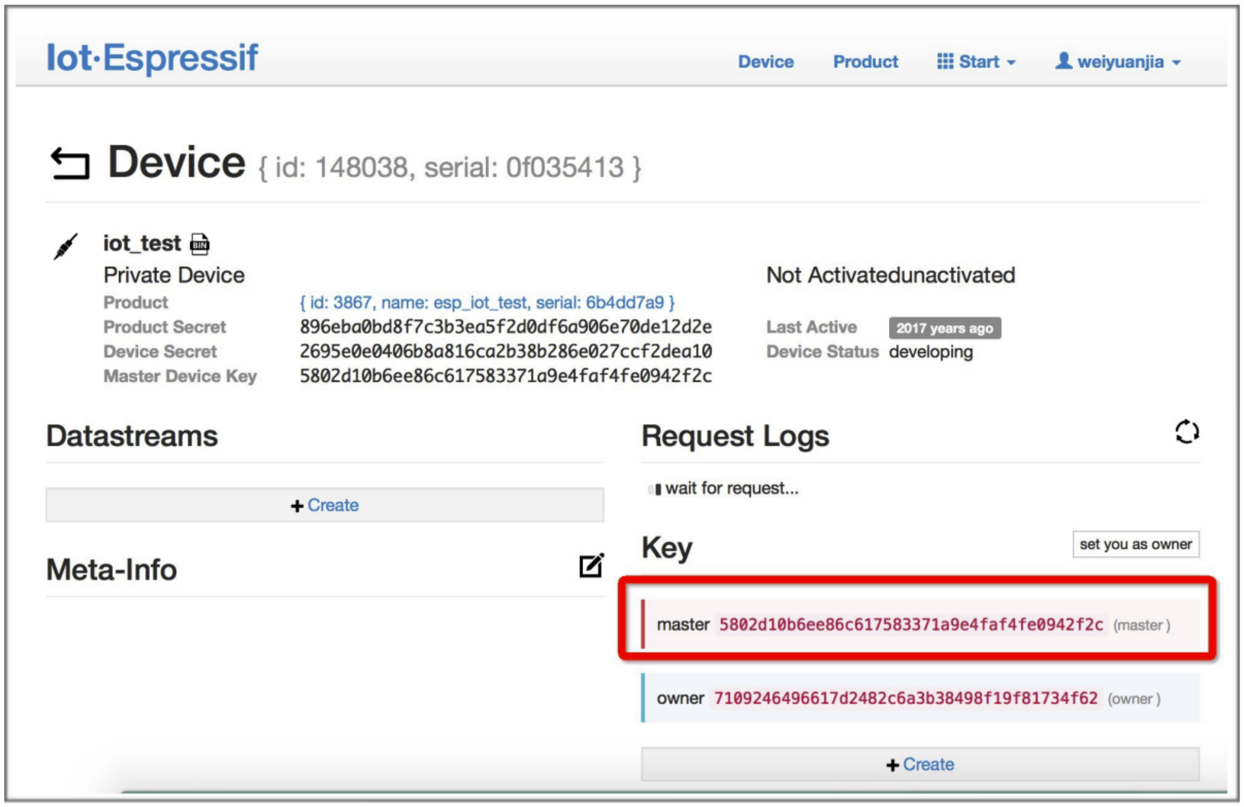


Fig. 6: Click on "Create"

4. A key is generated when the device is successfully created, as the figure below shows.



Device { id: 148038, serial: 0f035413 }

iot_test
Private Device

Product	{ id: 3867, name: esp_iot_test, serial: 6b4dd7a9 }
Product Secret	896eba0bd8f7c3b3ea5f2d0df6a906e70de12d2e
Device Secret	2695e0e0406b8a816ca2b38b286e027ccf2dea10
Master Device Key	5802d10b6ee86c617583371a9e4faf4fe0942f2c

Not Activatedunactivated

Last Active	2017 years ago
Device Status	developing

Key

master 5802d10b6ee86c617583371a9e4faf4fe0942f2c (master)

owner 7109246496617d2482c6a3b38498f19f81734f62 (owner)

Fig. 7: A key has been generated

5. Use the key to compile your own OTA BIN. The process of configuring the AT OTA token key is as follows:

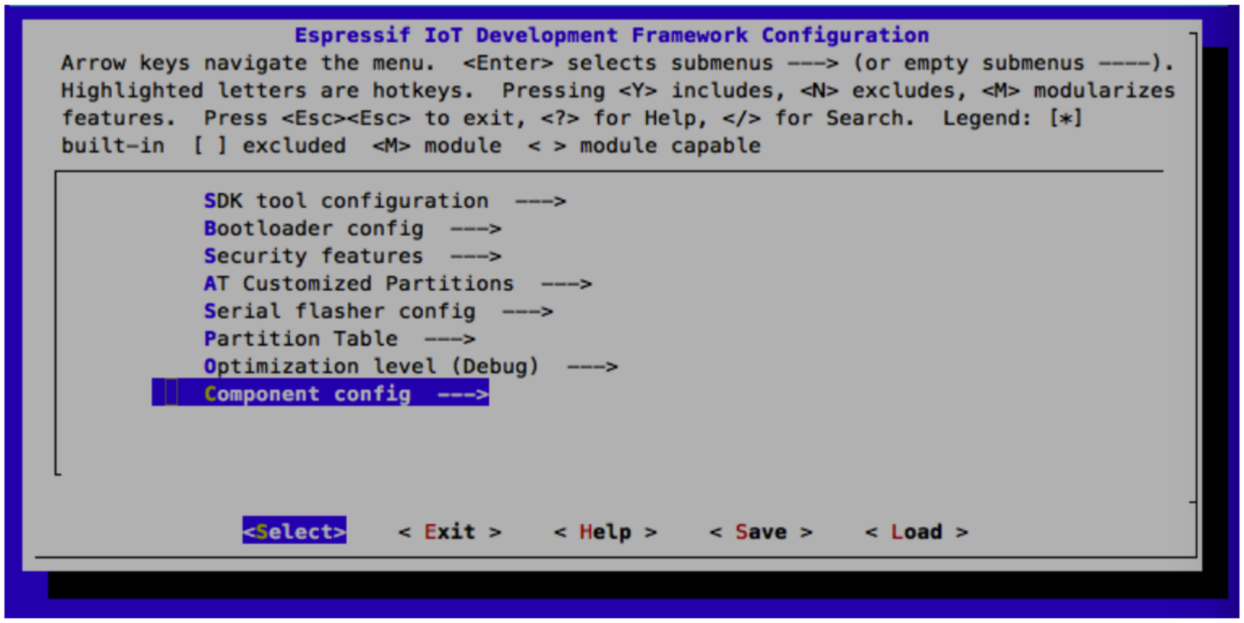
Note: If using SSL OTA, the option "OTA based upon ssl" should be selected.

6. Click on "Product" to enter the webpage, as shown below. Click on the device created. Enter version and corename under "ROM Deploy". Rename the BIN compiled in Step 5 as "ota.bin" and save the configuration.
7. Click on the ota.bin to save it as the current version.
8. Run the command AT+CIUPDATE on the ESP device. If the network is connected, OTA update will be done.

5.13 How to update the esp-idf version

The esp-at project supports two platforms by default, ESP32 UART AT and ESP8266 UART AT. Each of the platform has a set of configurations, you can configure its directory by setting the `ESP_AT_MODULE_CONFIG_DIR` in the `Makefile`. The default directory of ESP32 UART AT is `module_config/module_esp32_default`, and the ESP8266 UART AT is `module_config/module_esp8266_default`, version information is in the file `IDF_VERSION`. For example, the version info of `module_esp32_default` is:

```
branch:master
commit:7fa98593bc179ea50a1bc8244d5b94bac59c9a10
repository:https://github.com/espressif/esp-idf.git
```



```
Espressif IoT Development Framework Configuration
Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*]
built-in  [ ] excluded  <M> module  < > module capable

          SDK tool configuration  --->
          Bootloader config  --->
          Security features  --->
          AT Customized Partitions  --->
          Serial flasher config  --->
          Partition Table  --->
          Optimization level (Debug)  --->
          Component config  --->

       <Select>    < Exit >    < Help >    < Save >    < Load >
```

Fig. 8: Configuring the AT OTA token key - Step 1

branch: branch of the idfcommit: commit id of the idfrepository: url of the idf

To update the idf/SDK, you need to change the branch, commit and repository to be the ones that you want to update to.

It is suggested that you can delete the original idf/SDK after updating the file `IDF_VERSION`. In this case, the esp-at will clone the new idf/SDK according to the file `IDF_VERSION` firstly in next compilation.

Notice: If you changed the repository url in the file `IDF_VERSION`, then you have to delete the original idf/SDK in the project. Otherwise, the update may fail.

5.14 How to understand the differences of each type of module

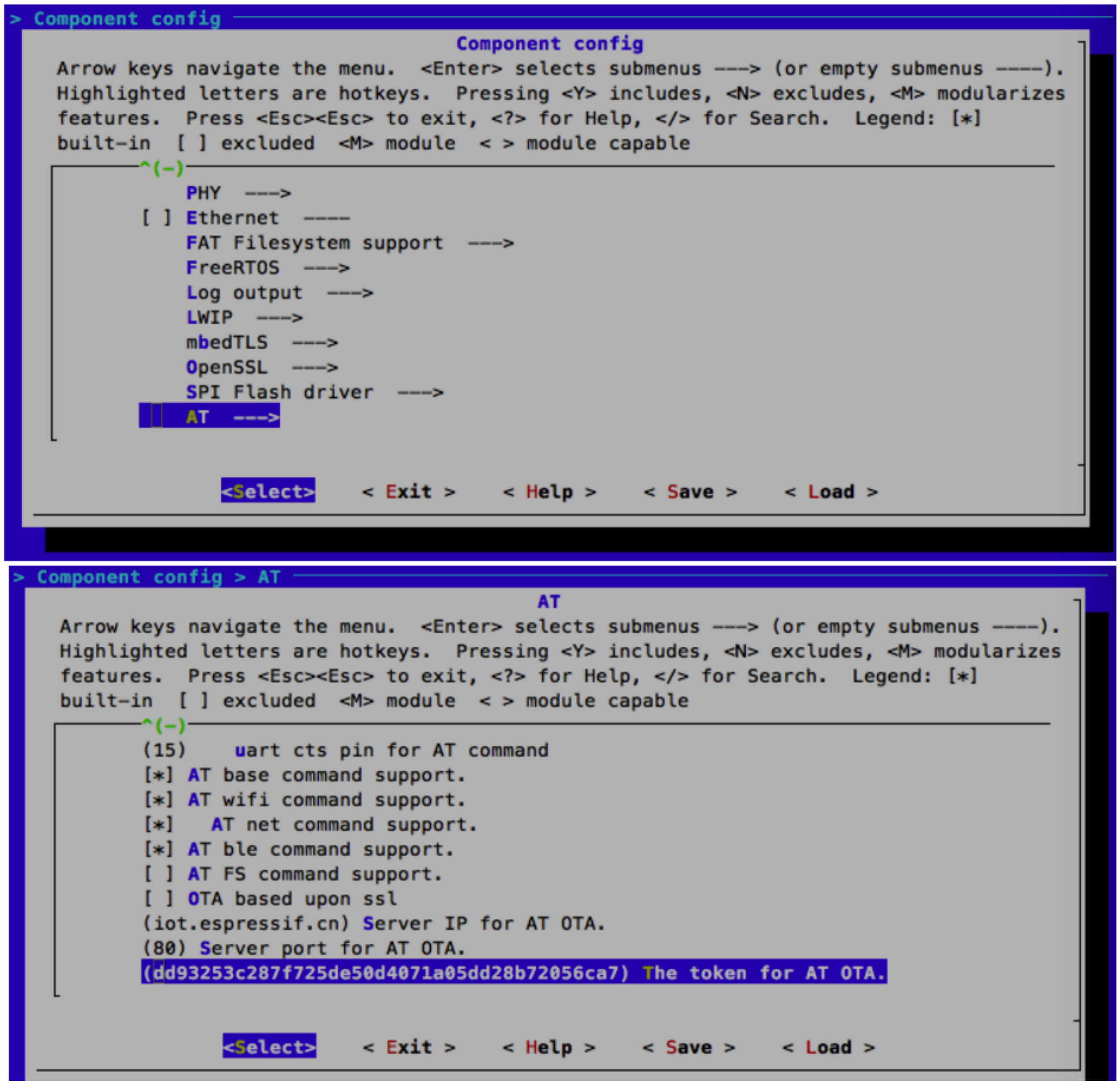
Since the esp-at project supports different module, for example, WROOM-32, WROVER-32, ESP8266_1MB, even supports WROOM-5V2L, when compiling the esp-at project, users can set different configurations to generate AT firmware for different ESP modules. The detailed information are in the `esp-at/module_config` directory.In order to facilitate customers to understand the differences between different modules of the same platform, they will be summarized in the following list.

5.14.1 1. ESP32 Platform

1. Command differences supported

2. Hardware differences

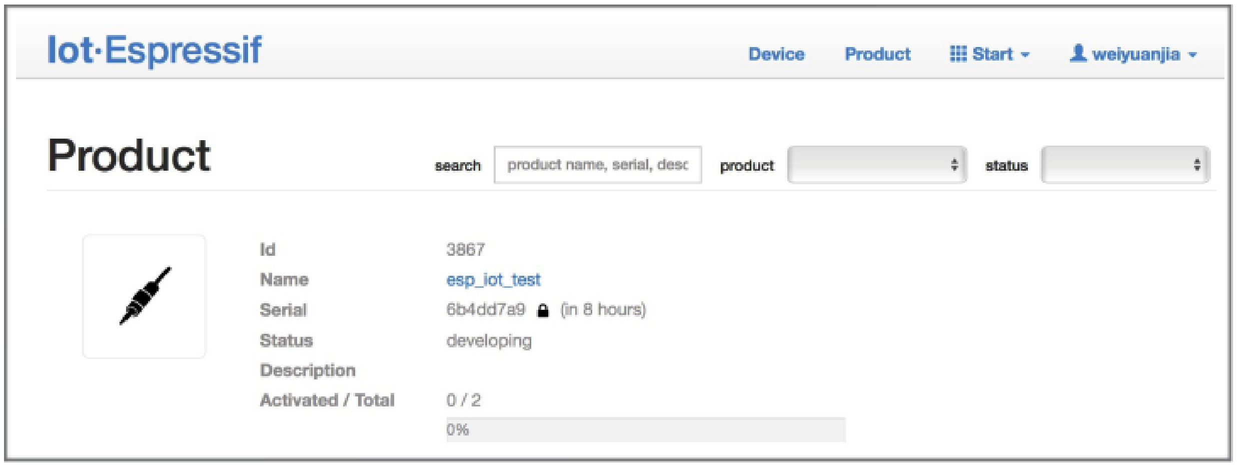
For factory param, see: `components/customized_partitions/raw_data/factory_param/factory_param_data.csv`



```
> Component config
                              Component config
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
  features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*]
  built-in  [ ] excluded  <M> module  < > module capable
  ┌───────^(-)────────────────────────────────────────────────────────────────┐
  │           PHY  --->
  │       [ ] Ethernet  ----
  │           FAT Filesystem support  --->
  │           FreeRTOS  --->
  │           Log output  --->
  │           LWIP  --->
  │           mbedTLS  --->
  │           OpenSSL  --->
  │           SPI Flash driver  --->
  │           AT  --->
  └
          <Select>    < Exit >    < Help >    < Save >    < Load >
```

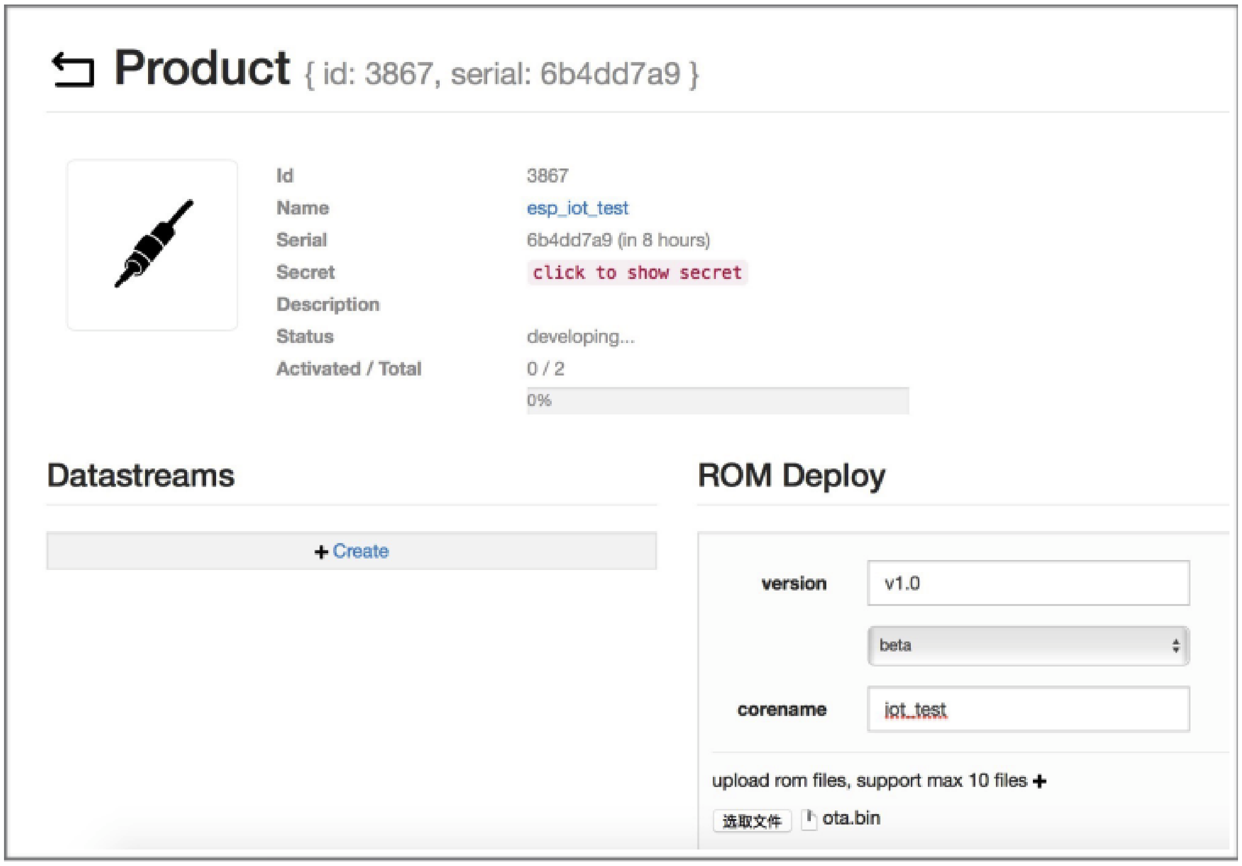
```
> Component config > AT
                                   AT
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes
  features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*]
  built-in  [ ] excluded  <M> module  < > module capable
  ┌───────^(-)────────────────────────────────────────────────────────────────┐
  │       (15)    uart cts pin for AT command
  │       [*] AT base command support.
  │       [*] AT wifi command support.
  │       [*]   AT net command support.
  │       [*] AT ble command support.
  │       [ ] AT FS command support.
  │       [ ] OTA based upon ssl
  │       (iot.espressif.cn) Server IP for AT OTA.
  │       (80) Server port for AT OTA.
  │       (dd93253c287f725de50d4071a05dd28b72056ca7) The token for AT OTA.
  └
          <Select>    < Exit >    < Help >    < Save >    < Load >
```

Fig. 9: Configuring the AT OTA token key - Step 2 and 3



Product

Id	3867
Name	esp_iot_test
Serial	6b4dd7a9 (in 8 hours)
Status	developing
Description	
Activated / Total	0 / 2
	0%



Product { id: 3867, serial: 6b4dd7a9 }

Id	3867
Name	esp_iot_test
Serial	6b4dd7a9 (in 8 hours)
Secret	click to show secret
Description	
Status	developing...
Activated / Total	0 / 2
	0%

version: v1.0

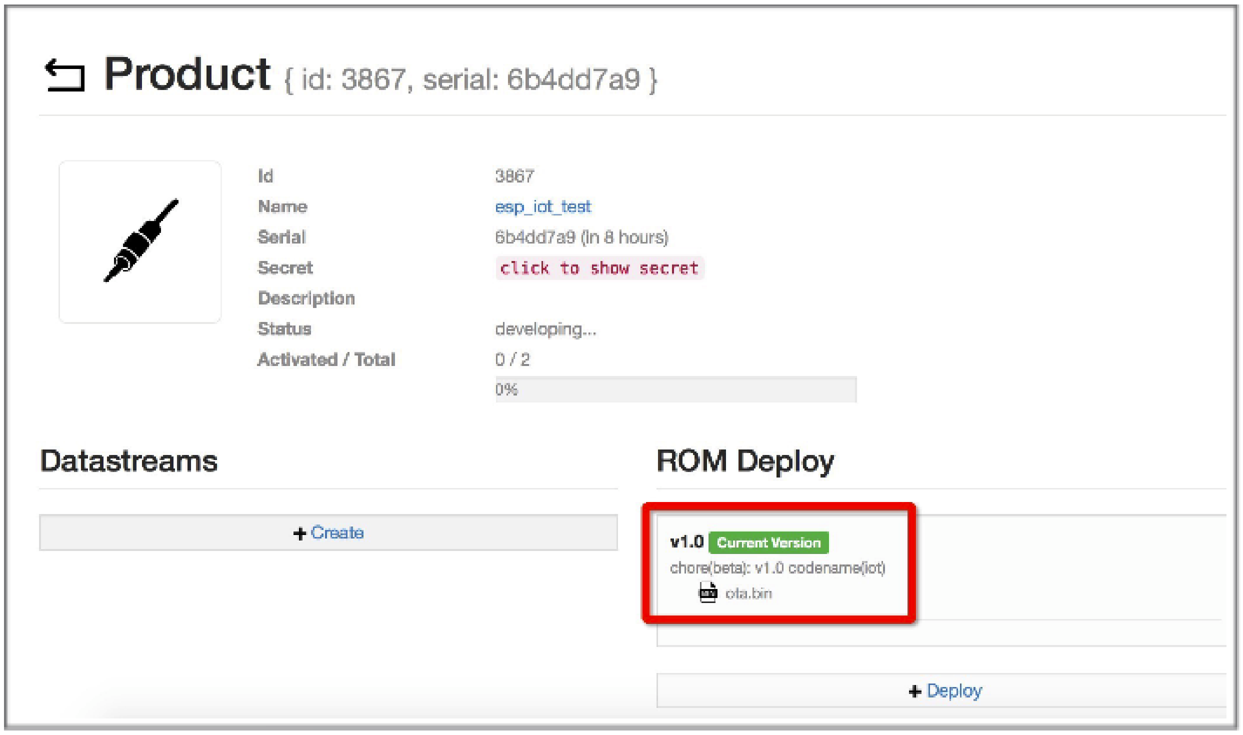
beta

corename: iot_test

upload rom files, support max 10 files +

ota.bin

Fig. 10: Enter version and corename


Product { id: 3867, serial: 6b4dd7a9 }

Id	3867
Name	esp_iot_test
Serial	6b4dd7a9 (In 8 hours)
Secret	click to show secret
Description	
Status	developing...
Activated / Total	0 / 2
	0%

Datastreams

+ Create

ROM Deploy

v1.0 Current Version

chore(beta): v1.0 codename(iot)

ota.bin

+ Deploy

Fig. 11: Save the current version of ota.bin

3. AT pin

See: https://docs.espressif.com/projects/esp-at/en/latest/AT_Binary_Lists/index.html

4. Firmware supported modules

5.14.2 2. ESP8266 Platform

1. Command differences supported

2. Hardware differences

For factory param, see: `components/customized_partitions/raw_data/factory_param/factory_param_data.csv`

3. AT pin

See: https://docs.espressif.com/projects/esp-at/en/latest/AT_Binary_Lists/index.html

4. Firmware supported modules

5.14.3 3. ESP32S2 Platform

1. Command differences supported

2. Hardware differences

For factory param, see: `components/customized_partitions/raw_data/factory_param/factory_param_data.csv`

3. AT pin

See: https://docs.espressif.com/projects/esp-at/en/latest/AT_Binary_Lists/index.html

4. Firmware supported modules

5.14.4 4. ESP32-C3 Platform

1. Command differences supported

2. Hardware differences

For factory param, see: `components/customized_partitions/raw_data/factory_param/factory_param_data.csv`

3. AT pin

See: https://docs.espressif.com/projects/esp-at/en/latest/AT_Binary_Lists/index.html

4. Firmware supported modules

5.15 AT API Reference

5.15.1 Header File

- at/include/esp_at_core.h

5.15.2 Functions

void **`esp_at_module_init`** (uint32_t *netconn_max*, **`const`** uint8_t **custom_version*)

This function should be called only once, before any other AT functions are called.

Parameters

- `netconn_max`: the maximum number of the link in the at module
- `custom_version`: version information by custom

esp_at_para_parse_result_type **`esp_at_get_para_as_digit`** (int32_t *para_index*, int32_t **value*)

Parse digit parameter from command string.

Return

- ESP_AT_PARA_PARSE_RESULT_OK : succeed
- ESP_AT_PARA_PARSE_RESULT_FAIL : fail
- ESP_AT_PARA_PARSE_RESULT_OMITTED : this parameter is OMITTED

Parameters

- `para_index`: the index of parameter
- `value`: the value parsed

esp_at_para_parse_result_type **`esp_at_get_para_as_str`**(int32_t *para_index*, uint8_t ***result*)

Parse string parameter from command string.

Return

- ESP_AT_PARA_PARSE_RESULT_OK : succeed
- ESP_AT_PARA_PARSE_RESULT_FAIL : fail
- ESP_AT_PARA_PARSE_RESULT_OMITTED : this parameter is OMITTED

Parameters

- `para_index`: the index of parameter
- `result`: the pointer that point to the result.

void **`esp_at_port_recv_data_notify_from_isr`**(int32_t *len*)

Calling the esp_at_port_recv_data_notify_from_isr to notify at module that at port received data. When received this notice,at task will get data by calling get_data_length and read_data in esp_at_device_ops. This function MUST be used in isr.

Parameters

- `len`: data length

bool **`esp_at_port_recv_data_notify`**(int32_t *len*, uint32_t *msec*)

Calling the esp_at_port_recv_data_notify to notify at module that at port received data. When received this notice,at task will get data by calling get_data_length and read_data in esp_at_device_ops. This function MUST NOT be used in isr.

Return

- true : succeed
- false : fail

Parameters

- `len`: data length
- `msec`: timeout time,The unit is millisecond. It waits forever,if msec is portMAX_DELAY.

void **`esp_at_transmit_terminal_from_isr`**(void)

terminal transparent transmit mode,This function MUST be used in isr.

void **`esp_at_transmit_terminal`**(void)

terminal transparent transmit mode,This function MUST NOT be used in isr.

bool **`esp_at_custom_cmd_array_regist`**(**const** *esp_at_cmd_struct* **custom_at_cmd_array*, uint32_t *cmd_num*)

regist at command set, which defined by custom,

Parameters

- `custom_at_cmd_array`: at command set
- `cmd_num`: command number

void **`esp_at_device_ops_regist`** (*esp_at_device_ops_struct *ops*)
regist device operate functions set,

Parameters

- `ops`: device operate functions set

bool **`esp_at_custom_net_ops_regist`** (int32_t *link_id*, *esp_at_custom_net_ops_struct *ops*)

bool **`esp_at_custom_ble_ops_regist`** (int32_t *conn_index*, *esp_at_custom_ble_ops_struct *ops*)

void **`esp_at_custom_ops_regist`** (*esp_at_custom_ops_struct *ops*)
regist custom operate functions set interacting with AT,

Parameters

- `ops`: custom operate functions set

uint32_t **`esp_at_get_version`** (void)
get at module version number,

Return at version bit31~bit24: at main version bit23~bit16: at sub version bit15~bit8 : at test version bit7~bit0 : at custom version

void **`esp_at_response_result`** (uint8_t *result_code*)
response AT process result,

Parameters

- `result_code`: see esp_at_result_code_string_index

int32_t **`esp_at_port_write_data`** (uint8_t **data*, int32_t *len*)
write data into device,

Return

- >= 0 : the real length of the data written
- others : fail.

Parameters

- `data`: data buffer to be written
- `len`: data length

int32_t **`esp_at_port_read_data`** (uint8_t **data*, int32_t *len*)
read data from device,

Return

- >= 0 : the real length of the data read from device
- others : fail

Parameters

- `data`: data buffer
- `len`: data length

bool **`esp_at_port_wait_write_complete`**(int32_t *timeout_msec*)
wait for transmitting data completely to peer device,

Return

- true : succeed,transmit data completely
- false : fail

Parameters

- `timeout_msec`: timeout time,The unit is millisecond.

int32_t **`esp_at_port_get_data_length`**(void)
get the length of the data received,

Return

- >= 0 : the length of the data received
- others : fail

bool **`esp_at_base_cmd_regist`**(void)
regist at base command set. If not,you can not use AT base command

bool **`esp_at_user_cmd_regist`**(void)
regist at user command set. If not,you can not use AT user command

bool **`esp_at_wifi_cmd_regist`**(void)
regist at wifi command set. If not,you can not use AT wifi command

bool **`esp_at_net_cmd_regist`**(void)
regist at net command set. If not,you can not use AT net command

bool **`esp_at_mdns_cmd_regist`**(void)
regist at mdns command set. If not,you can not use AT mdns command

bool **`esp_at_driver_cmd_regist`**(void)
regist at driver command set. If not,you can not use AT driver command

bool **`esp_at_wps_cmd_regist`**(void)
regist at wps command set. If not,you can not use AT wps command

bool **`esp_at_smartconfig_cmd_regist`**(void)
regist at smartconfig command set. If not,you can not use AT smartconfig command

bool **`esp_at_ping_cmd_regist`**(void)
regist at ping command set. If not,you can not use AT ping command

bool **`esp_at_http_cmd_regist`**(void)
regist at http command set. If not,you can not use AT http command

bool **`esp_at_mqtt_cmd_regist`**(void)
regist at mqtt command set. If not,you can not use AT mqtt command

bool **`esp_at_ble_cmd_regist`**(void)
regist at ble command set. If not,you can not use AT ble command

bool **`esp_at_ble_hid_cmd_regist`**(void)
regist at ble hid command set. If not,you can not use AT ble hid command

bool **`esp_at_blufi_cmd_regist`**(void)
regist at blufi command set. If not,you can not use AT blufi command

bool **`esp_at_bt_cmd_regist`**(void)
regist at bt command set. If not,you can not use AT bt command

bool **`esp_at_bt_spp_cmd_regist`**(void)
regist at bt spp command set. If not,you can not use AT bt spp command

bool **`esp_at_bt_a2dp_cmd_regist`**(void)
regist at bt a2dp command set. If not,you can not use AT bt a2dp command

bool **`esp_at_fs_cmd_regist`**(void)
regist at fs command set. If not,you can not use AT fs command

bool **`esp_at_eap_cmd_regist`**(void)
regist at WPA2 Enterprise AP command set. If not,you can not use AT EAP command

bool **`esp_at_eth_cmd_regist`**(void)
regist at ethernet command set. If not,you can not use AT ethernet command

bool **`esp_at_custom_cmd_line_terminator_set`**(uint8_t **terminator*)
Set AT command terminator, by default, the terminator is "\r\n" You can change it by calling this function, but it just supports one character now.

Return

- true : succeed,transmit data completely
- false : fail

Parameters

- `terminator`: the line terminator

uint8_t ***`esp_at_custom_cmd_line_terminator_get`**(void)
Get AT command line terminator,by default, the return string is "\r\n".

Return the command line terminator

`const` esp_partition_t ***`esp_at_custom_partition_find`**(esp_partition_t ype_t *type*, esp_partition_subtype_t *subtype*, **`const`** char **label*)
Find the partition which is defined in at_customize.csv.

Return pointer to esp_partition_t structure, or NULL if no partition is found. This pointer is valid for the lifetime of the application

Parameters

- `type`: the type of the partition
- `subtype`: the subtype of the partition
- `label`: Partition label

void **`esp_at_port_enter_specific`**(*esp_at_port_specific_callback_t callback*)
Set AT core as specific status, it will call callback if receiving data. for example:

```
static void wait_data_callback (void)
{
    xSemaphoreGive(sync_sema);
}

void process_task(void* para)
{
    vSemaphoreCreateBinary(sync_sema);
    xSemaphoreTake(sync_sema,portMAX_DELAY);
    esp_at_port_write_data((uint8_t *)">",strlen(">"));
    esp_at_port_enter_specific(wait_data_callback);
    while(xSemaphoreTake(sync_sema,portMAX_DELAY)) {
        len = esp_at_port_read_data(data, data_len);
        // TODO:
    }
}
```

Parameters

- `callback`: which will be called when received data from AT port

void **`esp_at_port_exit_specific`** (void)
Exit AT core as specific status.

`const` uint8_t ***`esp_at_get_current_cmd_name`** (void)
Get current AT command name.

esp_err_t **`esp_at_wifi_event_handler`** (void **ctx*, system_event_t **event*)
Wi-Fi event handler callback, which used in AT core.

Return

- ESP_OK: succeed
- others: fail

Parameters

- `ctx`: reserved for user
- `event`: event type defined in this file

5.15.3 Structures

`struct esp_at_cmd_struct`
esp_at_cmd_struct used for define at command

Public Members

char ***`at_cmdName`**
at command name

uint8_t (***`at_testCmd`**) (uint8_t *cmd_name)
Test Command function pointer

uint8_t (***`at_queryCmd`**) (uint8_t *cmd_name)
Query Command function pointer

uint8_t (***at_setupCmd**) (uint8_t para_num)
: Setup Command function pointer

uint8_t (***at_exeCmd**) (uint8_t *cmd_name)
: Execute Command function pointer

struct esp_at_device_ops_struct
esp_at_device_ops_struct device operate functions struct for AT

Public Members

int32_t (***read_data**) (uint8_t *data, int32_t len)
: read data from device

int32_t (***write_data**) (uint8_t *data, int32_t len)
: write data into device

int32_t (***get_data_length**) (void)
: get the length of data received

bool (***wait_write_complete**) (int32_t timeout_msec)
: wait write finish

struct esp_at_custom_net_ops_struct
esp_at_custom_net_ops_struct custom socket callback for AT

Public Members

int32_t (***recv_data**) (uint8_t *data, int32_t len)
: callback when socket received data

void (***connect_cb**) (void)
: callback when socket connection is built

void (***disconnect_cb**) (void)
: callback when socket connection is disconnected

struct esp_at_custom_ble_ops_struct
esp_at_custom_ble_ops_struct custom ble callback for AT

Public Members

int32_t (***recv_data**) (uint8_t *data, int32_t len)
: callback when ble received data

void (***connect_cb**) (void)
: callback when ble connection is built

void (***disconnect_cb**) (void)
: callback when ble connection is disconnected

struct esp_at_custom_ops_struct
esp_at_ops_struct some custom function interacting with AT

Public Members

void (***status_callback**)(*esp_at_status_type* status)
callback when AT status changes

void (***pre_deepsleep_callback**)(void)
callback before enter deep sleep

void (***pre_restart_callback**)(void)
callback before restart

5.15.4 Macros

ESP_AT_ERROR_NO(subcategory, extension)

ESP_AT_CMD_ERROR_OK
No Error

ESP_AT_CMD_ERROR_NON_FINISH
terminator character not found ("\r\n" expected)

ESP_AT_CMD_ERROR_NOT_FOUND_AT
Starting "AT" not found (or at, At or aT entered)

ESP_AT_CMD_ERROR_PARA_LENGTH(which_para)
parameter length mismatch

ESP_AT_CMD_ERROR_PARA_TYPE(which_para)
parameter type mismatch

ESP_AT_CMD_ERROR_PARA_NUM(need, given)
parameter number mismatch

ESP_AT_CMD_ERROR_PARA_INVALID(which_para)
the parameter is invalid

ESP_AT_CMD_ERROR_PARA_PARSE_FAIL(which_para)
parse parameter fail

ESP_AT_CMD_ERROR_CMD_UNSUPPORT
the command is not supported

ESP_AT_CMD_ERROR_CMD_EXEC_FAIL(result)
the command execution failed

ESP_AT_CMD_ERROR_CMD_PROCESSING
processing of previous command is in progress

ESP_AT_CMD_ERROR_CMD_OP_ERROR
the command operation type is error

5.15.5 Type Definitions

typedef void (***esp_at_port_specific_callback_t**)(void)
AT specific callback type.

5.15.6 Enumerations

enum esp_at_status_type

esp_at_status some custom function interacting with AT

Values:

ESP_AT_STATUS_NORMAL = 0x0
Normal mode.Now mcu can send AT command

ESP_AT_STATUS_TRANSMIT
Transparent Transmition mode

enum esp_at_module

module number,Now just AT module

Values:

ESP_AT_MODULE_NUM = 0x01
AT module

enum esp_at_error_code

subcategory number

Values:

ESP_AT_SUB_OK = 0x00
OK

ESP_AT_SUB_COMMON_ERROR = 0x01
reserved

ESP_AT_SUB_NO_TERMINATOR = 0x02
terminator character not found ("\r\n" expected)

ESP_AT_SUB_NO_AT = 0x03
Starting "AT" not found (or at, At or aT entered)

ESP_AT_SUB_PARA_LENGTH_MISMATCH = 0x04
parameter length mismatch

ESP_AT_SUB_PARA_TYPE_MISMATCH = 0x05
parameter type mismatch

ESP_AT_SUB_PARA_NUM_MISMATCH = 0x06
parameter number mismatch

ESP_AT_SUB_PARA_INVALID = 0x07
the parameter is invalid

ESP_AT_SUB_PARA_PARSE_FAIL = 0x08
parse parameter fail

ESP_AT_SUB_UNSUPPORT_CMD = 0x09
the command is not supported

ESP_AT_SUB_CMD_EXEC_FAIL = 0x0A
the command execution failed

ESP_AT_SUB_CMD_PROCESSING = 0x0B
processing of previous command is in progress

ESP_AT_SUB_CMD_OP_ERROR = 0x0C
the command operation type is error

enum esp_at_para_parse_result_type
the result of AT parse

Values:

ESP_AT_PARA_PARSE_RESULT_FAIL = -1
parse fail,Maybe the type of parameter is mismatched,or out of range

ESP_AT_PARA_PARSE_RESULT_OK = 0
Successful

ESP_AT_PARA_PARSE_RESULT_OMITTED
the parameter is OMITTED.

enum esp_at_result_code_string_index
the result code of AT command processing

Values:

ESP_AT_RESULT_CODE_OK = 0x00
"OK"

ESP_AT_RESULT_CODE_ERROR = 0x01
"ERROR"

ESP_AT_RESULT_CODE_FAIL = 0x02
"ERROR"

ESP_AT_RESULT_CODE_SEND_OK = 0x03
"SEND OK"

ESP_AT_RESULT_CODE_SEND_FAIL = 0x04
"SEND FAIL"

ESP_AT_RESULT_CODE_IGNORE = 0x05
response nothing, just change internal status

ESP_AT_RESULT_CODE_PROCESS_DONE = 0x06
response nothing, just change internal status

ESP_AT_RESULT_CODE_MAX

5.15.7 Header File

- at/include/esp_at.h

5.15.8 Functions

const char ***esp_at_get_current_module_name** (void)
get current module name

const char ***esp_at_get_module_name_by_id** (uint32_t *id*)
get module name by index

uint32_t **esp_at_get_module_id** (void)
get current module id

void **esp_at_board_init** (void)
init peripheral and default parameters in factory_param.bin

bool **`esp_at_web_server_cmd_regist`**(void)
: regist WiFi config via web command. If not,you can not use web server to config wifi connect

5.15.9 Macros

`ESP_AT_PORT_TX_WAIT_MS_MAX`

`ESP_AT_FACTORY_PARAMETER_SIZE`

CHAPTER 6

Customized AT Commands and Firmware

6.1 Tencent Cloud IoT AT Commands and Firmware

6.1.1 Tencent Cloud IoT AT Command Set

Please refer to [Chinese version]. The English version is not provided since the firmware and commands are applicable to the Chinese market (Tencent).

6.1.2 Tencent Cloud IoT AT Firmware

ESP8266

- ESP8266-WROOM-02 Series
  - QCloud_AT_ESP8266_v2.2.0
  - QCloud_AT_ESP8266_v2.1.1

CHAPTER 7

Index of Abbreviations

A2DP Advanced Audio Distribution Profile

ADC Analog-to-Digital Converter

ALPN Application Layer Protocol Negotiation

AT port AT port is the general name of AT log port (that is used to output log) and AT command port (that is used to send AT commands and receive responses). Please refer to *Hardware Connection* for default AT port pins and *How to Set AT Port Pins* for how to customize them.

AT AT AT AT *Hardware Connection* AT *How to Set AT Port Pins* AT

Bluetooth LE Bluetooth Low Energy

BluFi Wi-Fi network configuration function via Bluetooth channel

BluFi Wi-Fi

DHCP Dynamic Host Configuration Protocol

DNS Domain Name System

DTIM Delivery Traffic Indication Map

GATTC Generic Attributes client

GATT

GATTS Generic Attributes server

GATT

HID Human Interface Device

I2C Inter-Integrated Circuit

ICMP Intemet Control Message Protocol

LWT Last Will and Testament

MAC Media Access Control

MAC

mDNS Multicast Domain Name System

DNS

MSB Most Significant Bit

MTU maximum transmission unit

NVS Non-Volatile Storage

Normal Transmission Mode Default Transmission Mode

In normal transmission mode, users can send AT commands. For examples, users can send MCU data received by AT command port to the opposite end of transmission by *AT+CIPSEND*; and the data received from the opposite end of transmission will also be returned to MCU through AT command port with additional prompt: *+IPD*.

During a normal transmission, if the connection breaks, ESP devices will give a prompt and will not attempt to reconnect.

More details are in *Transmission Mode Shift Diagram*.

AT *AT+CIPSEND* AT MCU AT MCU *+IPD*

ESP

Transmission Mode Shift Diagram

Passthrough Mode Also called as "Passthrough Sending & Receiving Mode".

In passthrough mode, users cannot send AT commands except special +++ command. All MCU data received by AT command port will be sent to the opposite end of transmission without any modification; and the data received from the opposite end of transmission will also be returned to MCU through AT command port without any modification.

During the Wi-Fi passthrough transmission, if the connection breaks, ESP devices will keep trying to reconnect until +++ is input to exit the passthrough transmission.

More details are in *Transmission Mode Shift Diagram*.

""

AT +++ AT MCU AT MCU

Wi-Fi ESP +++

Transmission Mode Shift Diagram

Transmission Mode Shift Diagram

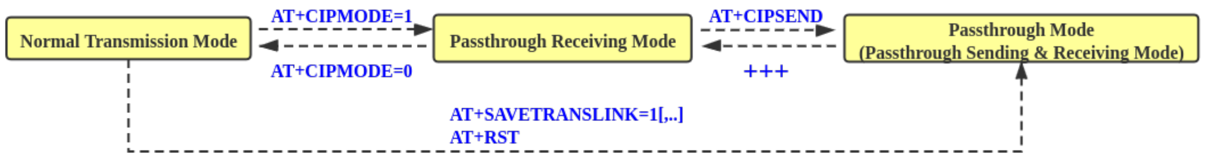


Fig. 1: Transmission Mode Shift Diagram

More details are in the following introduction.

- *Normal Transmission Mode* ()
- *Passthrough Receiving Mode* ()
- *Passthrough Mode* ()
- *AT+CIPMODE*
- *AT+CIPSEND*
- *+++*
- *AT+SAVETRANSLINK*

Passthrough Receiving Mode The temporary mode between *Normal Transmission Mode* and *Passthrough Mode*.

In passthrough receiving mode, AT cannot send any data to the opposite end of transmission; but the data received from the opposite end of transmission can be returned to MCU through AT command port without any modification. More details are in *Transmission Mode Shift Diagram*.

AT AT AT MCU *Transmission Mode Shift Diagram*

PBC Push Button Configuration

PCI Authentication Payment Card Industry Authentication. In ESP-AT project, it refers to all Wi-Fi authentication modes except OPEN and WEP.

PCI ESP-AT OPEN WEP Wi-Fi

PLCP Physical Layer Convergence Procedure

PLCP

PMF protected management frame

PSK Pre-shared Key

PWM Pulse-Width Modulation

QoS Quality of Service

RTC Real Time Controller. A group of circuits in SoC that keeps working in any chip mode and at any time.

SoC

SMP Security Manager Protocol

SNI Server Name Indication

SNTP Simple Network Time Protocol

SPI Serial Peripheral Interface

SPP Serial Port Profile

SPP

SSL Secure Sockets Layer

SSL

TLS Transport Layer Security

TLS

URC Unsolicited Result Code

MCU

UTC Coordinated Universal Time

UUID universally unique identifier

WEP Wired-Equivalent Privacy

WEP

WPA Wi-Fi Protected Access

Wi-Fi

WPA2 Wi-Fi Protected Access II

Wi-Fi II

WPS Wi-Fi Protected Setup

Wi-Fi

- genindex

Index