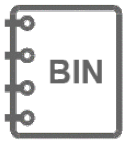
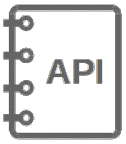
ESP-AT User Guide

Read the Docs

Mar 15, 2022

Contents

This is the documentation for the ESP-AT.

				
Get Started	AT Binary Lists	AT Command Set	AT Command Examples	Compile and Develop

CHAPTER 1

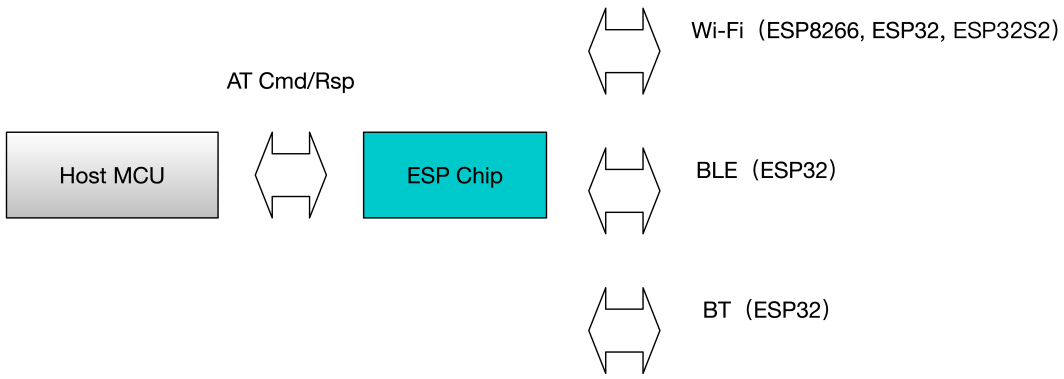
Get Started

[]

1.1 What is ESP-AT?

ESP-AT is an IoT solution developed by Espressif, which can be directly used in mass production. It aims to reduce customers' development costs and quickly form products. Through the ESP-AT command, 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/ESP8266-RTOS-SDK. It makes ESP board work as slave, and needs another MCU to work as host. Host MCU sends AT commands to ESP chip, and receives AT response.ESP-AT provides many AT commands with different functions. For example, Wi-Fi commands, TCPIP commands, BLE commands, BT commands, MQTT commands, HTTP commands, Ethernet commands, etc. In this case, host MCU can implement those functions through ESP board.



avatar

"AT" is short of "Attention". AT command starts with "AT", ends with a new line (CR LF). All commands are executed serially. Only one AT command can be executed at a time. Each command will return `OK` or `ERROR` to indicate the final execution status of the current command. When using the AT command, you should wait for the last command to be executed before sending the next one. If the previous command is not completed and a new command is sent, the related prompt `busy P...` will be returned. More details, please refer to the command descriptions.With the default configuration, host MCU connects to ESP board via UART, and sends/receives AT commands/responses through UART. But you can also change to use other peripheral, such as SDIO, according to your actual use scenario.And also, you can develop your own AT commands based on our ESP-AT project, to implement more features according to your actual use scenario.

1.2 Hardware Connection

1.2.1 ESP32 Series

ESP32 AT uses two UART port, UART0 is to download firmware and print background logs when running, UART1 is to send AT commands and receive AT responses. Different ESP32 modules use different GPIOs as UART1, so please pay attention to the hardware connection, and download the corresponding AT firmware to your specific ESP32 module.

Follow the guide Details of establishing serial connection, to establish your serial connection.

More details of ESP32 modules.

ESP32-WROOM-32 Series

Please refer to the data sheet, to get more details about ESP32-WROOM-32.

avatar

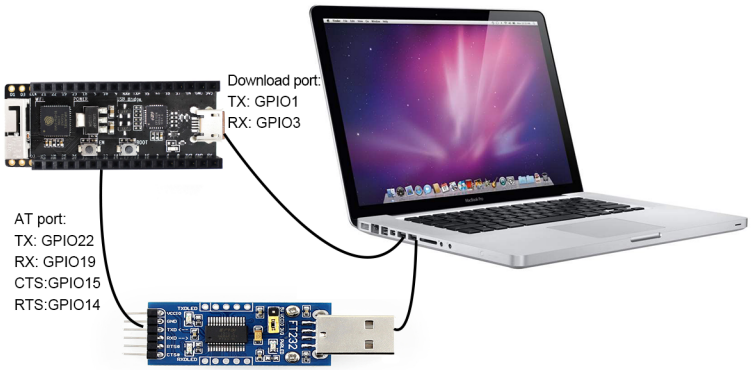
ESP32-WROVER-32 Series

Please refer to the data sheet, to get more details about ESP32-WROVER-32.

avatar

ESP32-PICO Series

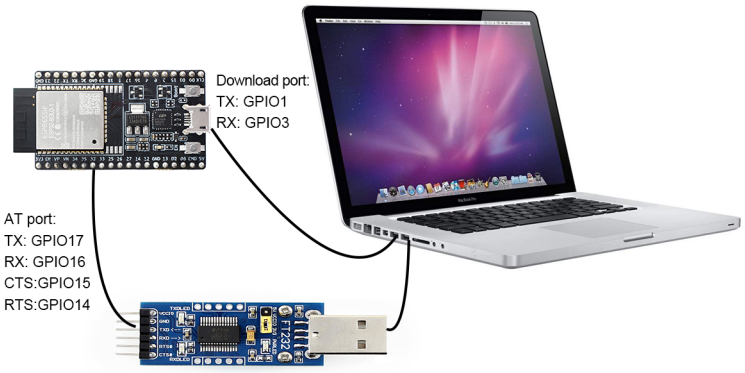
Please refer to the data sheet, to get more details about ESP32-PICO.



avatar

ESP32-SOLO Series

Please refer to the data sheet, to get more details about ESP32-SOLO.



avatar

1.2.2 ESP32S2 Series

ESP32S2 AT uses two UART port, UART0 is to download firmware and print background logs when running, UART1 is to send AT commands and receive AT responses.

Follow the guide Details of establishing serial connection, to establish your serial connection.

More details of ESP32S2 modules.

Please refer to the data sheet, to get more details about ESP32-S2-WROOM.

avatar

1.2.3 ESP8266 Series

ESP8266 AT uses two UART port, UART0 is to download firmware and send AT commands and receive AT responses, UART1 is to print background logs. The official module of ESP8266 is ESP-WROOM-02D

More details about ESP8266 module.

Hardware connection of ESP8266 module. avatar

Please refer to the data sheet, to get more details about ESP8266 module.

1.3 Downloading Guide

This page introduces how to download AT firmware, no matter you use Windows, Linux or MacOS.You can download AT firmware from `AT_Binary_Lists`.Herein, we use `ESP32-WROOM-32_AT_Bin_V2.1` as an example.

```
.
├── 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
```

(continues on next page)

(continued from previous page)

```
│   ├── factory_param_WROOM-32.bin
│   ├── mqtt_ca.bin
│   ├── 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 configuration of downloading is in `download.config`

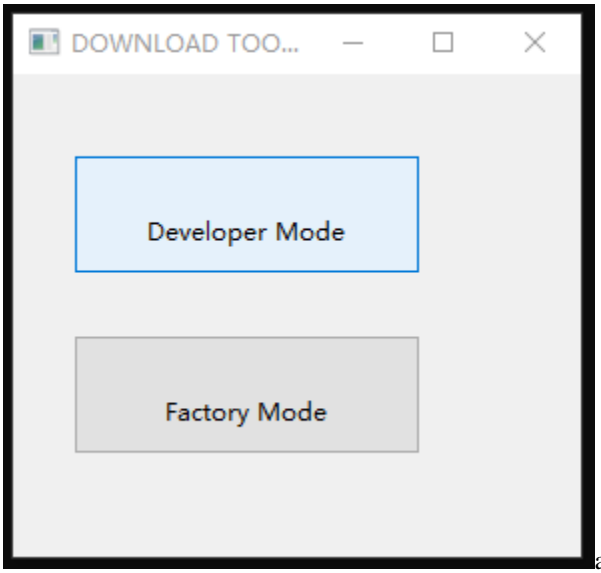
```
--flash_mode dio --flash_freq 40m --flash_size 4MB
0x8000 partition_table/partition-table.bin
0x10000 ota_data_initial.bin
0xf000 phy_init_data.bin
0x1000 bootloader/bootloader.bin
0x100000 esp-at.bin
0x20000 at_customize.bin
0x24000 customized_partitions/server_cert.bin
0x39000 customized_partitions/mqtt_key.bin
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`: the AT firmware is compiled with flash DIO mode
- `--flash_freq 40m`: the AT firmware's flash frequency is 40MHz
- `--flash_size 2MB`: the AT firmware is using flash size 2MB
- `0x10000 ota_data_initial.bin`: to download `ota_data_initial.bin` into address `0x10000`

1.3.1 Windows OS

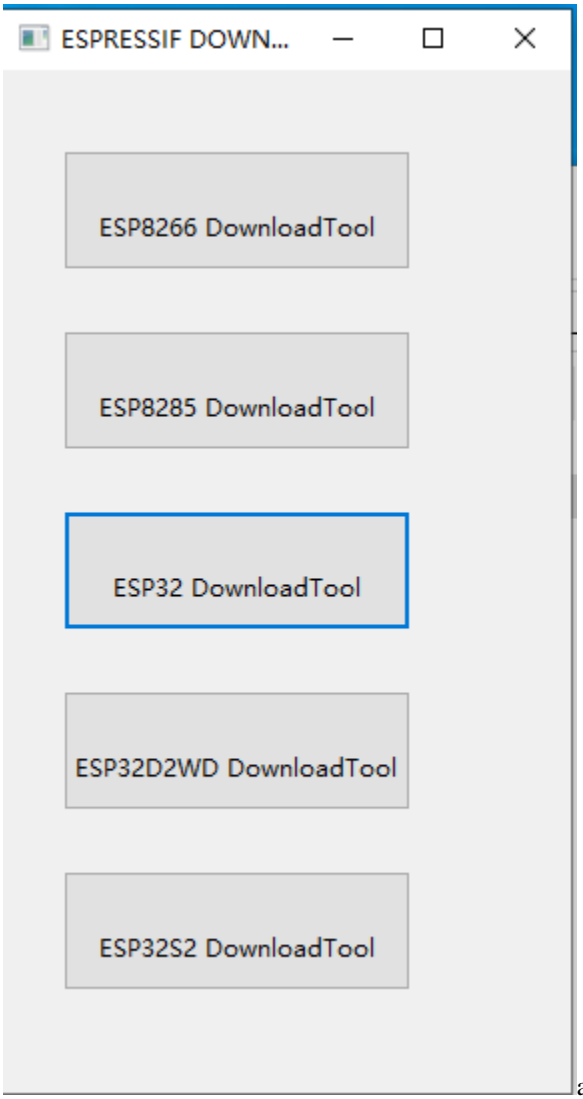
Click to download ESP Flash Download Tool for Windows.Herein, we use `ESP-WROOM-32` module's "Developer Mode" as an example. More details of the tool are in the `readme.pdf` of the `ESP Flash Download Tool`.

- Open the ESP Flash Download Tool.



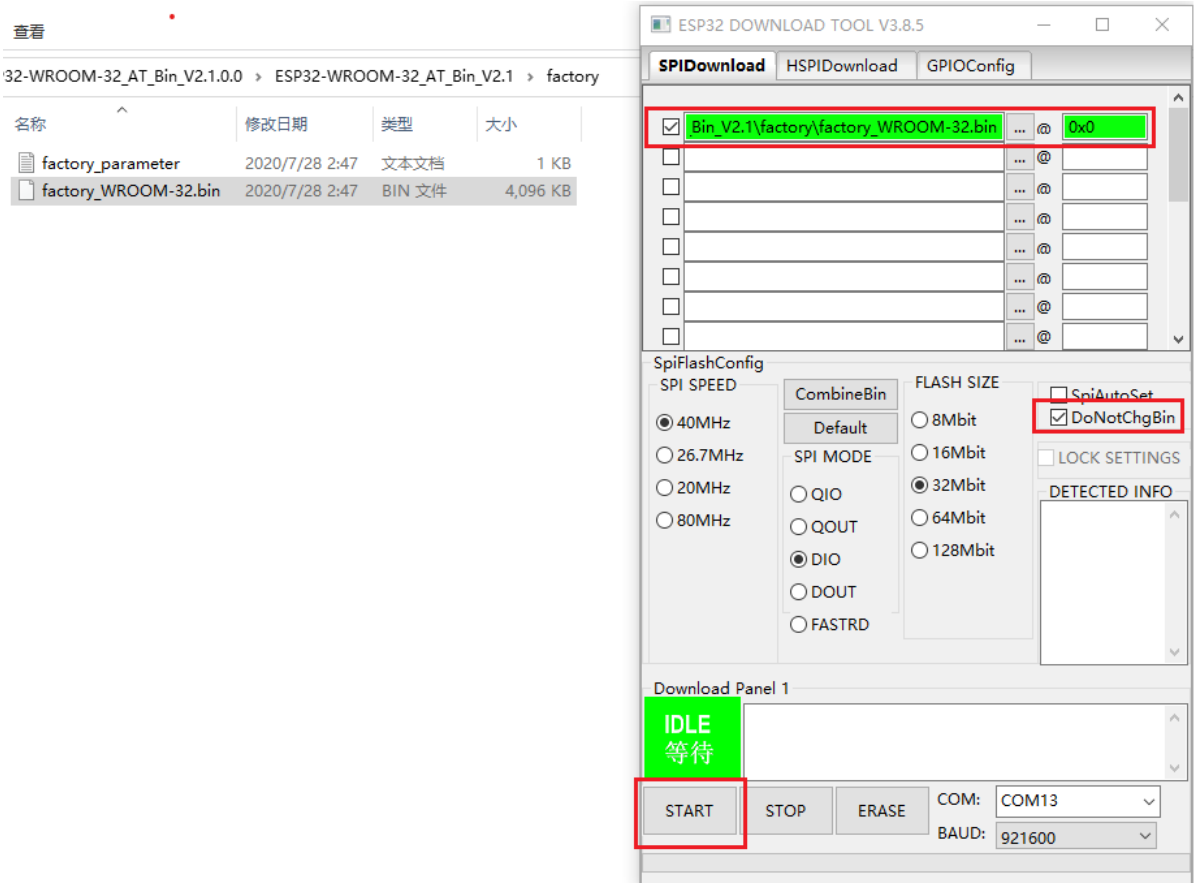
avatar

- Choose the target chip. For example, choose “ESP8266 DownloadTool” for ESP8266 chip; choose “ESP32S2 DownloadTool” for ESP32S2 chip.



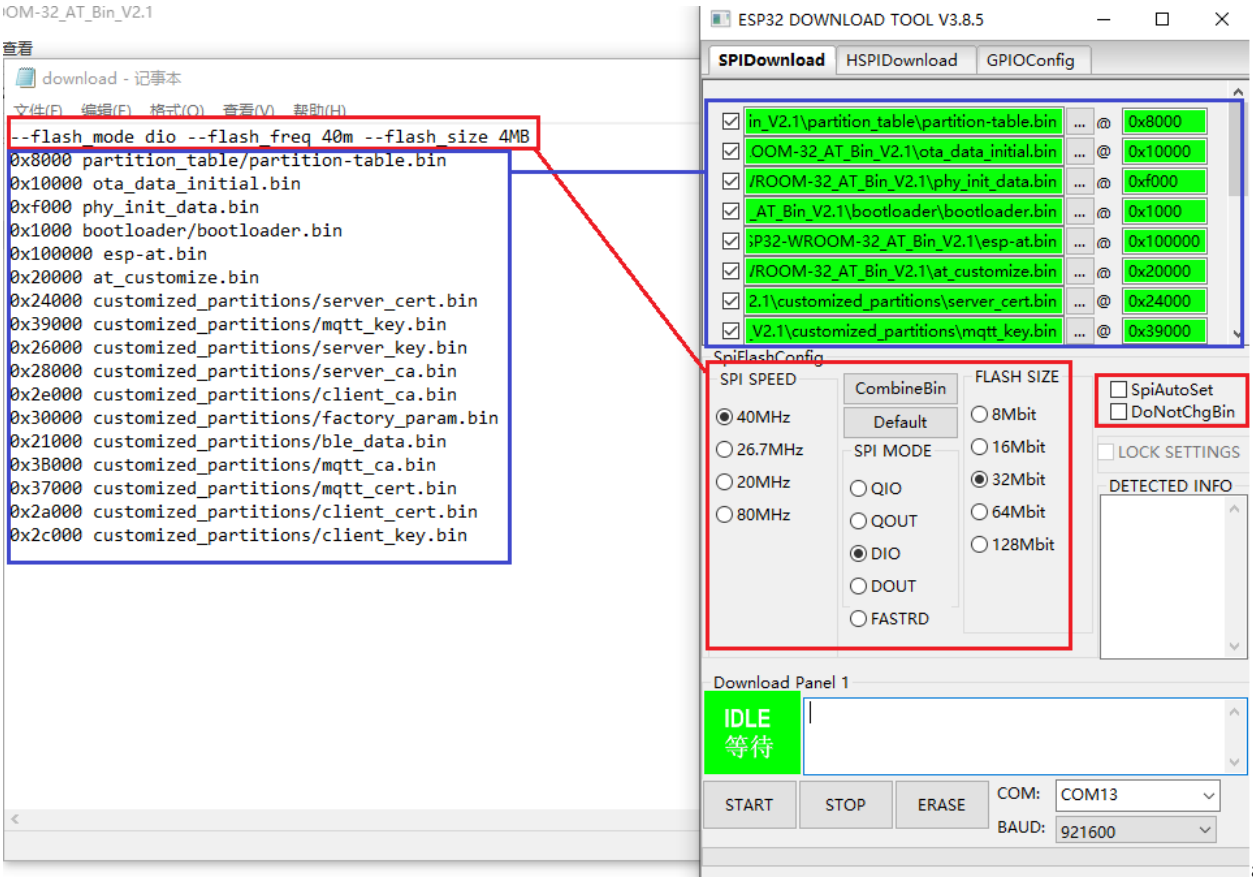
avatar

- You can download one combined factory bin to address 0, or download multiple bins separately to different addresses.
  - Method One: download the combined factory bin to address 0, select "DoNotChgBin" to use the default configuration in the factory bin.



avatar

- Method Two: download multiple bins separately to different addresses, do NOT select “DoNotChgBin”.



avatar

- After downloading, input "AT+GMR" with a new line (CR LF) to make sure the AT works.

```
AT+GMR
AT version:2.1.0.0(883f7f2 - Jul 24 2020 11:50:07)
SDK version:v4.0.1-193-ge7ac221
compile time(0ad6331):Jul 28 2020 02:47:21
Bin version:2.1.0(WROOM-32)

OK
```

avatar

1.3.2 Linux or MacOS

Detailed guide of downloading on Linux/MacOS.

Herein, we use `ESP-WROOM-32` as an example. Open shell on your PC, input the following command for downloading, please notice to set the UART port according to your actual usage.

```
esptool.py --chip auto --port /dev/tty.usbserial-0001 --baud 921600 --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
```

(continues on next page)

(continued from previous page)

Or just download the combined factory bin instead.

```
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
```

After downloading, input "AT+GMR" with a new line (CR LF) to make sure the AT works.

```
AT+GMR
AT version:2.1.0.0(883f7f2 - Jul 24 2020 11:50:07)
SDK version:v4.0.1-193-ge7ac221
compile time(0ad6331):Jul 28 2020 02:47:21
Bin version:2.1.0(WROOM-32)

OK
```

avatar

CHAPTER 2

AT Binary Lists

[]

2.1 Released Firmware

2.1.1 ESP32-WROOM-32 Series

- v2.1.0.0 ESP32-WROOM-32_AT_Bin_V2.1.0.0.zip
- 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
- 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
- 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
- v2.0.0.0 ESP32-SOLO_AT_Bin_V2.0.zip

2.2 Released Firmware

2.2.1 ESP32-S2-WROOM Series

- v2.1.0.0 ESP32-S2-WROOM_AT_Bin_V2.1.0.0.zip

2.2.2 ESP32-S2-WROVER Series

- v2.1.0.0 ESP32-S2-WROVER_AT_Bin_V2.1.0.0.zip

2.2.3 ESP32-S2-SOLO Series

- v2.1.0.0 ESP32-S2-SOLO_AT_Bin_V2.1.0.0.zip

2.2.4 ESP32-S2-MINI Series

- v2.1.0.0 ESP32-S2-MINI_AT_Bin_V2.1.0.0.zip

2.3 Released Firmware

2.3.1 ESP-WROOM-02 Series

- v2.1.0.0 ESP8266-IDF-AT_V2.1.0.0.zip
- 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`

- **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 the AT commands can only work on the ESP32, which is marked as [ESP32 Only]; others can work on both the ESP8266 and ESP32.

3.1 Basic AT Commands

- *AT* : Tests AT startup.
- *AT+RST* : Restarts a module.
- *AT+GMR* : Checks version information.
- *AT+GSLP* : Enters Deep-sleep mode.
- *ATE* : Configures echoing of AT commands.
- *AT+RESTORE* : Restores the factory default settings of the module.
- *AT+UART_CUR* : Current UART configuration.
- *AT+UART_DEF* : Default UART configuration, saved in flash.
- *AT+SLEEP* : Sets the sleep mode.
- *AT+SYSRAM* : Checks current remaining heap size and minimum heap size.
- *AT+SYSMSG* : Set message format.
- *AT+RFPOWER* : Set RF TX Power.
- *AT+SYSFLASH* : Set User Partitions in Flash.
- [ESP32 Only] *AT+FS* : Filesystem Operations.
- *AT+SYSROLLBACK* : Roll back to the previous firmware.
- *AT+SYSTIMESTAMP*: Set local time stamp.
- *AT+SYSLOG* : Enable or disable the AT error code prompt.

- *AT+SLEEPWKCFG* : Config the light-sleep wakeup source and awake GPIO.
- *AT+SYSSTORE* : Config parameter store mode.

3.1.1 AT—Tests AT Startup

Execute Command:

```
AT
```

Response:

```
OK
```

3.1.2 AT+RST—Restarts the Module

Execute Command:

```
AT+RST
```

Response:

```
OK
```

3.1.3 AT+GMR—Checks Version Information

Execute Command:

```
AT+GMR
```

Response:

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

OK
```

Parameters:

- **<AT version info>**: information about the AT version.
- **<SDK version info>**: information about the SDK version.
- **<compile time>**: the duration of time for compiling the BIN.

3.1.4 AT+GSLP—Enters Deep-sleep Mode

Set Command:

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

Response:

```
<time>

OK
```

Parameters:

- **<time>**: the duration of the device's deep sleep. Unit: ms.ESP device will automatically wake up after the deep-sleep for as many milliseconds (ms) as <time> indicates.Upon waking up, the device calls deep sleep wake stub, and then proceeds to load application.

Note:

- On ESP8266 platform, in order to timing wake up, it is necessary to connect GPIO16 to RST pin.
- Moreover, ESP8266 can be waken up from deep sleep externally by directly triggering RST pin low level pulse.

3.1.5 ATE—AT Commands Echoing

Execute Command:

```
ATE
```

Response:

```
OK
```

Parameters:

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

3.1.6 AT+RESTORE—Restores the Factory Default Settings

Execute Command:

```
AT+RESTORE
```

Response:

```
OK
```

Note:

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

3.1.7 AT+UART_CUR—Current UART Configuration, Not Saved in Flash

Query Command:

```
AT+UART_CUR?
```

Response:

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

OK
```

Note:

- Command `AT+UART_CUR?` will return the actual value of UART configuration parameters, which may have allowable errors compared with the set value because of the clock division.

Set Command:

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

Response:

```
OK
```

Parameters:

- **<baudrate>**: UART baud rate
- **<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 NOT be saved in flash.
- The use of flow control requires the support of hardware:
  - IO15 is UART0 CTS
  - IO14 is UART0 RTS
- The range of baud rates supported: 80 ~ 5000000.

Example:

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

3.1.8 AT+UART_DEF—Default UART Configuration, Saved in Flash

Query Command:

```
AT+UART_DEF?
```

Response:

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

OK
```

Set Command:

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

Response:

```
OK
```

Parameters:

- **<baudrate>**: UART baud rate
- **<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.
- The use of flow control requires the support of hardware:
  - IO15 is UART0 CTS
  - IO14 is UART0 RTS
- The range of baud rates supported: 80 ~ 5000000.

Example:

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

3.1.9 AT+SLEEP—Sets the Sleep Mode

Set Command:

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

Response:

```
OK
```

Parameters:

- **<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`.

Example:

```
AT+SLEEP=0
```

Notes:

- Light sleep is not available for ESP32S2 currently.

3.1.10 AT+SYSRAM—Checks current remaining heap size and minimum heap size

Query 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
```

3.1.11 AT+SYSMSG—Control to use new or old information

Query Command:

```
AT+SYSMSG?
Function:
Query the current system message state.
```

Response:

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

Set Command:

```
AT+SYSMSG=<state>
Function:
Control to use new or old information.
```

Response:

```
OK
```

Parameters:

- **<state>**:
  - Bit0: Quit transparent transmission0: Quit transparent transmission no information.1: Quit transparent transmission will supply information.
  - Bit1: Connection information0: Use old connection information.1: Use new connection information.
  - Bit2: conection status information when in Wi-Fi transparent transmission, Ble SPP and BT SPP0: There is no more prompt information but received data.1: It will print some information if Wi-Fi, socket, ble or bt status is changed, the prompt is as following

```
- "CONNECT\r\n" or the message prefixed with "+LINK_CONN:"
- "CLOSED\r\n"
- "WIFI CONNECTED\r\n"
- "WIFI GOT IP\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 set Bit0 to 1 will supply information "+QUITT" when quit transparent transmission.
- If set Bit1 to 1 will impact the infomation 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:

// Use new connection info and quit transparent transmission no information AT+SYSMSG=2

3.1.12 AT+SYSFLASH—Set User Partitions in Flash

Query Command:

```
AT+SYSFLASH?
Function:
Check the user partitions in flash.
```

Response:

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

Set Command:

```
AT+SYSFLASH=<operation>,<partition>,<offset>,<length>
Function:
Read/write the user partitions in flash.
```

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:

- at_customize.bin has to be downloaded, so that the relevant commands can be used. Please refer to the ESP_AT_Customize_Partitions for more details.
- Important things to note when erasing user partitions:
  - When erasing the targeted user partition in its entirety, parameters `<offset>` and `<length>` can be omitted. For example, command `AT+SYSFLASH=0,"ble_data"` can erase the entire "ble_data" user partition.
  - If parameters `<offset>` and `<length>` are not omitted when erasing the user partition, 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 actual data, which length is parameter `<length>`.

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
```

3.1.13 [ESP32 Only] AT+FS—Filesystem Operations

Set 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>**: offset, for writing and reading operations only
- **<length>**: data length, for writing and reading operations only

Notes:

- at_customize.bin has to be downloaded, so that the relevant commands can be used. The definitions of user partitions are in esp-at/at_customize.csv. Please refer to the ESP32_Customize_Partitions for more details.
- 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 actual data, which length is 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,"."
```

3.1.14 AT+RFPOWER-Set RF TX Power

Query Command:

```
AT+RFPOWER?
Function: to query the RF TX Power.
```

Response:

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

Set Command:

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

Response:

```
OK
```

Parameters:

- **<wifi_power>**: the unit is 0.25dBm. for example, if set the value is 78, then the actual RF max power value is 78*0.25dBm = 19.5dBm.after configuring it, please confirm the actual value by AT+RFPOWER?
  - range: [40, 78] on ESP32 platform and ESP32S2 platform, please refer to the notes below for details
  - range: [40, 82] on ESP8266 platform, please refer to the notes below for details
- **<ble_adv_power>**: RF TX Power of BLE advertising, range: [0, 7]
  - 0:7dBm
  - 1:4dBm
  - 2:1dBm
  - 3:-2 dBm

  - 4:-5 dBm
  - 5:-8 dBm
  - 6:-11 dBm
  - 7:-14 dBm
- **<ble_scan_power>**: RF TX Power of BLE scanning, range: [0, 7], the same as **<ble_adv_power>**
- **<ble_conn_power>**: RF TX Power of BLE connecting, range: [0, 7], the same as **<ble_adv_power>**

Notes:

1. Since the RF TX power is actually divided into several levels, and each level has its own value range, so 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 the query value will not be larger than the set one.
2. Relationship between set value and actual value, as following,

ESP32 and ESP32S2 platform:

ESP8266 platform:

3.1.15 AT+SYSROLLBACK-Roll back to the previous firmware

Execute Command:

```
AT+SYSROLLBACK
```

Response:

```
OK
```

Note:

- This command will not upgrade via OTA, only roll back to the firmware which is in the other ota partition.

3.1.16 AT+SYSTIMESTAMP—Set local time stamp.

Query Command:

```
AT+SYSTIMESTAMP?
Function: to query the time stamp.
```

Response:

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

Set Command:

```
AT+SYSTIMESTAMP=<Unix_timestamp>
Function: to set local time stamp. It will be the same as SNTP time when the SNTP␣
↪time updated.
```

Response:

```
OK
```

Parameters:

- **<Unix_timestamp>**: Unix timestamp, the unit is seconds.

Example:

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

3.1.17 AT+SYSLOG : Enable or disable the AT error code prompt.

Query Command:

```
AT+SYSLOG?
Function: to query the AT error code prompt for whether it is enabled or disabled.
```

Response:

```
+SYSLOG:<status>

OK
```

Set Command:

```
AT+SYSLOG=<status>
Function: Enable or disable the AT error code prompt.
```

Response:

```
OK
```

Parameters:

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

Example:If enable AT error code prompt:

```
AT+SYSLOG=1

OK
AT+FAKE
ERR CODE:0x01090000

ERROR
```

If disable AT error code prompt:

```
AT+SYSLOG=0

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

ERROR
```

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

- **category:** stationary value 0x01
- **subcategory:** error type
- **extension:** error extension information, there are different extension for different subcategory, the detail is defined in `components/at/include/esp_at.h`

subcategory is defined as follows:

```
    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 */
```

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

3.1.18 AT+SLEEPWKCFG—Config the light-sleep wakeup source and awake GPIO.

Set Command:

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

Response:

```
OK
```

Parameters:

- **<wakeup source>**:
  - 0: Wakeup by timer.
  - 1: Wakeup by UART. (Only Support ESP32)
  - 2: Wakeup by GPIO.
- **<param1>**:
  - If the wakeup source is timer, this param is time before wakeup, the units is millisecond.
  - If the wakeup source is UART. this param is the Uart number.

  - If the wakeup source is GPIO, the param is the GPIO number.
- **<param2>**:
  - If the wakeup source is GPIO, the param is the wakeup level, 0: Low level, 1: High level.

Example:

```
AT+SLEEPWKCFG=0,1000  // Timer wakeup
AT+SLEEPWKCFG=1,1     // Uart1 wakeup, Only Support ESP32
AT+SLEEPWKCFG=2,12,0  // GPIO12 wakeup, low level.
```

Notes:

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

3.1.19 AT+SYSSTORE— Config parameter store mode

Query Command:

```
AT+SYSSTORE?
Function: to query the AT parameter store mode.
```

Response:

```
+SYSSTORE:<store_mode>

OK
```

Set Command:

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

Response:

```
OK
```

Parameters:

- **<store_mode>**:
  - 0: Do not store command configuration into flash.
  - 1: Store command configuration into flash.

Affected commands:

```
AT+SYSMSG
AT+CWMODE
AT+CWJAP
AT+CWSAP
AT+CIPAP
AT+CIPSTA
AT+CIPAPMAC
AT+CIPSTAMAC
AT+CIPDNS
AT+CIPSSLCCONF
AT+CIPRECONNINTV
AT+CWDHCPS
```

(continues on next page)

(continued from previous page)

```
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
```

Note:

```
* The default value of `store_mode` is 1;
* `AT+SYSSTORE` only effects on setup command, query command is always got from ram.
```

Example:

```
AT+SYSSTORE=0
AT+CWMODE=1  // Do not store into flash
AT+CWJAP="test","1234567890" // Do not store into flash

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

3.2 Wi-Fi AT Commands

- *AT+CWMODE* : Sets the Wi-Fi mode (STA/AP/STA+AP).
- *AT+CWJAP* : Connects to an AP.
- *AT+CWLAPOPT* : Sets the configuration of command AT+CWLAP.
- *AT+CWLAP* : Lists available APs.
- *AT+CWQAP* : Disconnects from the AP.
- *AT+CWSAP* : Sets the configuration of the ESP SoftAP.
- *AT+CWLIF* : Gets the Station IP to which the ESP SoftAP is connected.
- *AT+CWQIF* : Disconnect Station from the ESP SoftAP.
- *AT+CWDHCP* : Enables/disables DHCP.
- *AT+CWDHCPS* : Sets the IP range of the ESP SoftAP DHCP server.
- *AT+CWAUTOCONN* : Connects to the AP automatically on power-up.
- *AT+CWAPPROTO* : Sets the 802.11 b/g/n protocol standard of SoftAP mode.
- *AT+CWSTAPROTO* : Sets the 802.11 b/g/n protocol standard of station mode.

- *AT+CIPSTAMAC* : Sets the MAC address of ESP Station.
- *AT+CIPAPMAC* : Sets the MAC address of ESP SoftAP.
- *AT+CIPSTA* : Sets the IP address of ESP Station.
- *AT+CIPAP* : Sets the IP address of ESP SoftAP.
- *AT+CWSTARTSMART* : Starts SmartConfig.
- *AT+CWSTOPSMART* : Stops SmartConfig.
- *AT+WPS* : Enables the WPS function.
- *AT+MDNS* : Configurates the MDNS function
- [ESP32 Only] *AT+CWJEAP* : Connects to a WPA2 Enterprise AP.
- *AT+CWHOSTNAME* : Configures the Name of ESP Station
- *AT+CWCOUNTRY* : Configures the Wi-Fi Country Code

3.2.1 AT+CWMODE—Sets the Wi-Fi Mode (Station/SoftAP/Station+SoftAP)

Query Command:

```
AT+CWMODE?
Function: to query the Wi-Fi mode of ESP32.
```

Response:

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

Set Command:

```
AT+CWMODE=<mode>[,<auto_connect>]
Function: to set the Wi-Fi mode of ESP32.
```

Response:

```
OK
```

Parameters:

- **<mode>**:
  - 0: Null mode, WiFi RF will be disabled
  - 1: Station mode
  - 2: SoftAP mode
  - 3: SoftAP+Station mode
- **<auto_connect>**:
  - 0: Do not connect to WiFi when WiFi mode change to Station or Station+SoftAP
  - 1: Connect to WiFi when WiFi mode change to Station or Station+SoftAP (default configuration)

Note:

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

Example:

```
AT+CWMODE=3
```

3.2.2 AT+CWJAP—Connects to an AP

Query Command:

```
AT+CWJAP?
Function: to query the AP to which the ESP32 Station is already connected.
```

Response:

```
+CWJAP:<ssid>,<bssid>,<channel>,<rssi>,<pci_en>,<reconn>,<listen_interval>,<scan_mode>
OK
```

Parameters:

- **<ssid>**: a string parameter showing the SSID of the AP.
- **<bssid>**: the AP's MAC address.
- **<channel>**: channel
- **<rssi>**: signal strength
- **[<pci_en>]**: PCI Authentication, which will disable connect OPEN and WEP AP.
- **[<reconn>]**: Wi-Fi reconnection, when beacon timeout, ESP32 will reconnect automatically.
- **[<listen_interval>]**: the interval of listening to the AP's beacon,the range is (0,100]
- **[<scan_mode>]**:
  - 0: Do fast scan, scan will end after find SSID match AP, Wi-Fi will connect the first scanned AP.
  - 1: All channel scan, scan will end after scan all the channel, Wi-Fi will connect the AP with the strongest signal scanned.

Set Command:

```
AT+CWJAP=<ssid>,<pwd>[,<bssid>][,<pci_en>][,<reconn>][,<listen_interval>][,<scan_mode>
↪]
Function: to set the AP to which the ESP32 Station needs to be connected.
```

Response:

```
OK
```

or +CWJAP: ERROR Parameters:

- **<ssid>**: the SSID of the target AP.
  - Escape character syntax is needed if SSID or password contains any special characters, such as , or " or \.
- **<pwd>**: password, MAX: 64-byte ASCII.
- **[<bssid>]**: the target AP's MAC address, used when multiple APs have the same SSID.
- **[<pci_en>]**: enable PCI Authentication, which will disable connect OPEN and WEP AP.
- **[<reconn>]**: enable Wi-Fi reconnection, when beacon timeout, ESP32 will reconnect automatically.

- **[<listen_interval>]**: the interval of listening to the AP's beacon,the range is (0,100], by default, the value is 3.
- **<error code>**: (for reference only)
  - 1: connection timeout.
  - 2: wrong password.
  - 3: cannot find the target AP.
  - 4: connection failed.
  - others: unknown error occurred.
- **[<scan_mode>]**:
  - 0: Do fast scan, scan will end after find SSID match AP, Wi-Fi will connect the first scanned AP.
  - 1: All channel scan, scan will end after scan all the channel, Wi-Fi will connect the AP with the strongest signal scanned.

Note:

- The configuration changes will be saved in the NVS area if `AT+SYSSTORE=1`.
- This command requires Station mode to be active.

Examples:

```
AT+CWJAP="abc","0123456789"
For example, if the target AP's SSID is "ab\,c" and the password is "0123456789"\",␣
↪the command is as follows:
AT+CWJAP="ab\\\,c","0123456789\"\\"
If multiple APs have the same SSID as "abc", the target AP can be found by BSSID:
AT+CWJAP="abc","0123456789","ca:d7:19:d8:a6:44"
```

3.2.3 AT+CWLAPOPT—Sets the Configuration for the Command AT+CWLAP

Set Command:

```
AT+CWLAPOPT=<sort_enable>,<mask>
```

Response:

```
OK
```

Parameters:

- **<sort_enable>**: determines whether the result of command AT+CWLAP will be listed according to RSSI:
  - 0: the result is not ordered according to RSSI.
  - 1: the result is ordered according to RSSI.
- **<mask>**: determines the parameters shown in the result of `AT+CWLAP`;
  - 0 means not showing the parameter corresponding to the bit, and 1 means showing it.
  - bit 0: determines whether <ecn> will be shown in the result of `AT+CWLAP`.
  - bit 1: determines whether <ssid> will be shown in the result of `AT+CWLAP`.
  - bit 2: determines whether <rssi> will be shown in the result of `AT+CWLAP`.
  - bit 3: determines whether <mac> will be shown in the result of `AT+CWLAP`.

  - bit 4: determines whether <channel> will be shown in the result of AT+CWLAP.

Example:

```
AT+CWLAPOPT=1,31
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 <mask>
↪ are set to 1. All parameters will be shown in the result of AT+CWLAP.
```

3.2.4 AT+CWLAP—Lists the Available APs

Set Command:

```
AT+CWLAP=[<ssid>,<mac>,<channel>,<scan_type>,<scan_time_min>,<scan_time_max>]
Function: to query the APs with specific SSID and MAC on a specific channel.
```

Execute Command:

```
AT+CWLAP
Function: to list all available APs.
```

Response:

```
+CWLAP:<ecn>,<ssid>,<rssi>,<mac>,<channel>
OK
```

Parameters:

- **<ecn>**: encryption method.
  - 0: OPEN
  - 1: WEP
  - 2: WPA_PSK
  - 3: WPA2_PSK
  - 4: WPA_WPA2_PSK
  - 5: WPA2_Enterprise (AT can NOT connect to WPA2_Enterprise AP for now.)
- **<ssid>**: string parameter, SSID of the AP.
- **<rssi>**: signal strength.
- **<mac>**: string parameter, MAC address of the AP.
- **<scan_type>**: Wi-Fi scan type, active or passive.
  - 0: active scan
  - 1: passive scan
- **<scan_time_min>**: minimum active scan time per channel, units: millisecond, range [0,1500], if the scan type is passive, this param is invalid.
- **<scan_time_max>**: maximum active scan time per channel, units: millisecond, range [0,1500]. if this param is zero, the firmware will use the default time, active scan type is 120ms , passive scan type is 360ms.

Examples:

```
AT+CWLAP="Wi-Fi","ca:d7:19:d8:a6:44",6,0,400,1000
Or search for APs with a designated SSID:
AT+CWLAP="Wi-Fi"
```

3.2.5 AT+CWQAP—Disconnects from the AP

Execute Command:

```
AT+CWQAP
```

Response:

```
OK
```

3.2.6 AT+CWSAP—Configuration of the ESP32 SoftAP

Query Command:

```
AT+CWSAP?
Function: to obtain the configuration parameters of the ESP32 SoftAP.
```

Response:

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

Set Command:

```
AT+CWSAP=<ssid>,<pwd>,<chl>,<ecn>[,<max conn>][,<ssid hidden>]
Function: to set the configuration of the ESP32 SoftAP.
```

Response:

```
OK
```

Parameters:

- **<ssid>**: string parameter, SSID of AP.
- **<pwd>**: string parameter, length of password: 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>]**(optional parameter): maximum number of Stations to which ESP32 SoftAP can be connected; within the range of [1, 10].
- **[<ssid hidden>]**(optional parameter):

  - 0: SSID is broadcast. (the default setting)
  - 1: SSID is not broadcast.

Note:

- This command is only available when SoftAP is active.
- The configuration changes will be saved in the NVS area if `AT+SYSSTORE=1`.

Example:

```
AT+CWSAP="ESP32","1234567890",5,3
```

3.2.7 AT+CWLIF—IP of Stations to Which the ESP32 SoftAP is Connected

Execute Command:

```
AT+CWLIF
```

Response:

```
<ip addr>,<mac>
OK
```

Parameters:

- **<ip addr>**: IP address of Stations to which ESP32 SoftAP is connected.
- **<mac>**: MAC address of Stations to which ESP32 SoftAP is connected.

Note:

- This command cannot get a static IP. It only works when both DHCPs of the ESP32 SoftAP, and of the Station to which ESP32 is connected, are enabled.

3.2.8 AT+CWQIF—Disconnect Station from the ESP SoftAP

Execute Command:

```
AT+CWQIF
Function: Disconnect all stations that connected to the ESP SoftAP.
```

Response:

```
OK
```

Set Command:

```
AT+CWQIF=<mac>
Function: Disconnect the station whose mac is "<mac>" from the ESP SoftAP.
```

Response:

```
OK
```

Parameters:

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

3.2.9 AT+CWDHCP—Enables/Disables DHCP

Query Command:

```
AT+CWDHCP?
```

Response: state

Set Command:

```
AT+CWDHCP=<operate>,<mode>
Function: to enable/disable DHCP.
```

Response:

```
OK
```

Parameters:

- **<operate>**:
  - 0: disable
  - 1: enable
- **<mode>**:
  - Bit0: Station DHCP
  - Bit1: SoftAP DHCP
- **<state>**:DHCP disabled or enabled now? Bit0: 0: Station DHCP is disabled. 1: Station DHCP is enabled. Bit1: 0: SoftAP DHCP is disabled. 1: SoftAP DHCP is enabled. ***Notes:***
- The configuration changes will be saved in the NVS area if `AT+SYSSTORE=1`.
- This set command interacts with static-IP-related AT commands(AT+CIPSTA-related and AT+CIPAP-related commands):
  - If DHCP is enabled, static IP will be disabled;
  - If static IP is enabled, DHCP will be disabled;
  - Whether it is DHCP or static IP that is enabled depends on the last configuration.

Examples:

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

3.2.10 AT+CWDHCPS—Sets the IP Address Allocated by ESP32 SoftAP DHCP

Query Command:

```
AT+CWDHCPS?
```

Response:

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

Set Command:

```
AT+CWDHCPS=<enable>,<lease time>,<start IP>,<end IP>
Function: sets the IP address range of the ESP32 SoftAP DHCP server.
```

Response:

```
OK
```

Parameters:

- **<enable>**:
  - 0: Disable the settings and use the default IP range.
  - 1: Enable setting the IP range, and the parameters below have to be set.
- **<lease time>**: lease time, unit: minute, range [1, 2880].
- **<start IP>**: start IP of the IP range that can be obtained from ESP32 SoftAP DHCP server.
- **<end IP>**: end IP of the IP range that can be obtained from ESP32 SoftAP DHCP server.

Notes:

- The configuration changes will be saved in the NVS area if `AT+SYSSTORE=1`.
- This AT command is enabled when ESP8266 runs as SoftAP, and when DHCP is enabled.
- The IP address should be in the same network segment as the IP address of ESP32 SoftAP.

Examples:

```
AT+CWDHCPS=1,3,"192.168.4.10","192.168.4.15"
or
AT+CWDHCPS=0 //Disable the settings and use the default IP range.
```

3.2.11 AT+CWAUTOCONN—Auto-Connects to the AP or Not

Set Command:

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

Response:

```
OK
```

Parameters:

- **<enable>**:
  - 0: does NOT auto-connect to AP on power-up.
  - 1: connects to AP automatically on power-up.

Note:

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

- The ESP32 Station connects to the AP automatically on power-up by default.

Example:

```
AT+CWAUTOCONN=1
```

3.2.12 AT+CWAPPROTO—Sets the 802.11 b/g/n protocol standard of SoftAP mode.

Query Command:

```
AT+CWAPPROTO?
```

Response:

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

Set Command:

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

Response:

```
OK
```

Parameters:

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

Note:

- ESP8266 Currently only support 802.11b or 802.11bg mode
- ESP32 Currently only support 802.11b or 802.11bg or 802.11bgn mode

3.2.13 AT+CWSTAPROTO—Sets the 802.11 b/g/n protocol standard of station mode.

Query Command:

```
AT+CWSTAPROTO?
```

Response:

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

Set 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 we only support 802.11b or 802.11bg or 802.11bgn mode

3.2.14 AT+CIPSTAMAC—Sets the MAC Address of the ESP32 Station

Query Command:

```
AT+CIPSTAMAC?
Function: to obtain the MAC address of the ESP32 Station.
```

Response:

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

Set Command:

```
AT+CIPSTAMAC=<mac>
Function: to set the MAC address of the ESP32 Station.
```

Response:

```
OK
```

Parameters:

- **<mac>**: string parameter, MAC address of the ESP8266 Station.

Notes:

- The configuration changes will be saved in the NVS area if `AT+SYSSTORE=1`.
- The MAC address of ESP32 SoftAP is different from that of the ESP32 Station. Please make sure that you do not set the same MAC address for both of them.
- Bit 0 of the ESP32 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 and cannot be set.

Example:

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

3.2.15 AT+CIPAPMAC—Sets the MAC Address of the ESP32 SoftAP

Query Command:

```
AT+CIPAPMAC?
Function: to obtain the MAC address of the ESP32 SoftAP.
```

Response:

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

Set Command:

```
AT+CIPAPMAC=<mac>
Function: to set the MAC address of the ESP32 SoftAP.
```

Response:

```
OK
```

Parameters:

- **<mac>**: string parameter, MAC address of the ESP8266 SoftAP.

Notes:

- The configuration changes will be saved in the NVS area if `AT+SYSSTORE=1`.
- The MAC address of ESP32 SoftAP is different from that of the ESP32 Station. Please make sure that you do not set the same MAC address for both of them.
- Bit 0 of the ESP32 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"
```

3.2.16 AT+CIPSTA—Sets the IP Address of the ESP32 Station

Query Command:

```
AT+CIPSTA?
Function: to obtain the IP address of the ESP32 Station.
Notice: Only when the ESP32 Station is connected to an AP can its IP address be␣
↪queried.
```

Response:

```
+CIPSTA:<ip>
OK
```

Set Command:

```
AT+CIPSTA=<ip>[,<gateway>,<netmask>]
Function: to set the IP address of the ESP32 Station.
```

Response:

```
OK
```

Parameters:

- **<ip>**: string parameter, the IP address of the ESP32 Station.
- **[<gateway>]**: gateway.
- **[<netmask>]**: netmask.

Notes:

- The configuration changes will be saved in the NVS area if `AT+SYSSTORE=1`.
- The set command interacts with DHCP-related AT commands (AT+CWDHCP-related commands):
  - If static IP is enabled, DHCP will be disabled;
  - If DHCP is enabled, static IP will be disabled;
  - Whether it is DHCP or static IP that is enabled depends on the last configuration.

Example:

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

3.2.17 AT+CIPAP—Sets the IP Address of the ESP32 SoftAP

Query Command:

```
AT+CIPAP?
Function: to obtain the IP address of the ESP32 SoftAP.
```

Response:

```
+CIPAP:<ip>,<gateway>,<netmask>
OK
```

Set Command:

```
AT+CIPAP=<ip>[,<gateway>,<netmask>]
Function: to set the IP address of the ESP32 SoftAP.
```

Response:

```
OK
```

Parameters:

- **<ip>**: string parameter, the IP address of the ESP32 SoftAP.
- **[<gateway>]**: gateway.
- **[<netmask>]**: netmask.

Notes:

- The configuration changes will be saved in the NVS area if `AT+SYSSTORE=1`.
- The set command interacts with DHCP-related AT commands (AT+CWDHCP-related commands):
  - If static IP is enabled, DHCP will be disabled;
  - If DHCP is enabled, static IP will be disabled;
  - Whether it is DHCP or static IP that is enabled depends on the last configuration.

Example:

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

3.2.18 AT+CWSTARTSMART—Starts SmartConfig

Execute Command:

```
AT+CWSTARTSMART
Function: to start SmartConfig. (The type of SmartConfig is ESP-TOUCH + AirKiss.
```

Set Command:

```
AT+CWSTARTSMART=<type>
Function: to start SmartConfig of a designated type.
```

Response:

```
OK
```

Parameters:

- **<type>**:
  - 1: ESP-TOUCH
  - 2: AirKiss
  - 3: ESP-TOUCH+AirKiss

Notes:

- For details on SmartConfig please see ESP-TOUCH User Guide.
- SmartConfig is only available in the ESP32 Station mode.
- The message `Smart get Wi-Fi info` means that SmartConfig has successfully acquired the AP information. ESP32 will try to connect to the target AP.
- Message`Smartconfig connected Wi-Fi` is printed if the connection is successful.
- 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
```

3.2.19 AT+CWSTOPSMART—Stops SmartConfig

Execute Command:

```
AT+CWSTOPSMART
```

Response:

```
OK
```

Note:

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

Example:

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

3.2.20 AT+WPS—Enables the WPS Function

Set Command:

```
AT+WPS=<enable>
```

Response:

```
OK
```

Parameters:

- **<enable>**:
  - 1: enable WPS/Wi-Fi Protected Setup (implemented by PBC/Push Button Configuration).
  - 0: disable WPS (implemented by PBC).

Notes:

- WPS must be used when the ESP32 Station is enabled.
- WPS does not support WEP/Wired-Equivalent Privacy encryption.

Example:

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

3.2.21 AT+MDNS—Configurates the MDNS Function

Set Command:

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

Response:

```
OK
```

Parameters:

- **<enable>**:
  - 1: enables the MDNS function; the following three parameters need to be set.
  - 0: disables the MDNS function; the following three parameters need not 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
```

3.2.22 AT+CWJEAP—Connects to an WPA2 Enterprise AP.

Query Command:

```
AT+CWJEAP?
Function: to query the Enterprise AP to which the ESP32 Station is already connected.
```

Response:

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

Set Command:

```
AT+CWJEAP=<ssid>,<method>,<identity>,<username>,<password>,<security>
Function: to set the Enterprise AP to which the ESP32 Station needs to be connected.
```

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 " 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, must set for EAP-PEAP and EAP-TTLS mode, nor care for EAP-TLS, string limited to 1~32.

- **<password>**: password for phase 2, must set for EAP-PEAP and EAP-TTLS mode, nor care for EAP-TLS, string limited to 1~32.
- **<security>**:
  - Bit0: Client certificate
  - Bit1: Server certificate

Example:

```
1. Connect to EAP-TLS mode enterprise AP, set identity, verify server certificate and load client certificate
AT+CWJEAP="dlink11111",0,"example@espressif.com",,,3
2. 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
    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, make sure enabled.

3.2.23 AT+CWHOSTNAME : Configures the Name of ESP Station

Query Command:

```
AT+CWHOSTNAME?
Function: Checks the host name of ESP Station.
```

Response:

```
+CWHOSTNAME:<hostname>

OK
```

Set Command:

```
AT+CWHOSTNAME=<hostname>
Function: Sets the host name of ESP Station.
```

Response:

```
OK
```

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

```
ERROR
```

Parameters:

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

Note:

- The configuration changes are not saved in the flash.

Example:

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

3.2.24 AT+CWCOUNTRY : Configures the Wi-Fi Country Code

Query Command:

```
AT+CWCOUNTRY?
Function: Query Wi-Fi country code information.
```

Response:

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

OK
```

Set Command:

```
AT+ CWCOUNTRY=<country_policy>,<country_code>,<start_channel>,<total_channel_count>
Function: Sets the Wi-Fi country code information.
```

Response:

```
OK
```

Parameters:

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

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+CIPSTATUS* : Gets the connection status.
- *AT+CIPDOMAIN* : Domain Name Resolution Function.
- *AT+CIPSTART* : Establishes TCP connection, UDP transmission or SSL connection.
- *AT+CIPSTARTEX* : Establishes TCP connection, UDP transmission or SSL connection with automatically assigned ID.
- *AT+CIPSEND* : Sends data.
- *AT+CIPSENDEX* : Sends data when length of data is <length>, or when \0 appears in the data.
- *AT+CIPCLOSE* : Closes TCP/UDP/SSL connection.
- *AT+CIFSR* : Gets the local IP address.
- *AT+CIPMUX* : Configures the multiple connections mode.
- *AT+CIPSERVER* : Deletes/Creates TCP or SSL server.
- *AT+CIPSERVERMAXCONN* : Set the Maximum Connections Allowed by Server.
- *AT+CIPMODE* : Configures the transmission mode.
- *AT+SAVETRANSLINK* : Saves the transparent transmission link in flash.
- *AT+CIPSTO* : Sets timeout when ESP32 runs as a TCP server.
- *AT+CIPSNTPCFG* : Configures the time domain and SNTP server.
- *AT+CIPSNTPTIME* : Queries the SNTP time.

- *AT+CIUPDATE* : Updates the software through Wi-Fi.
- *AT+CIPDINFO* : Shows remote IP and remote port with +IPD.
- *AT+CIPSSLCCONF* : Config SSL client.
- *AT+CIPSSLCCN* : Config SSL client common name.
- *AT+CIPSSLCSNI* : Config SSL client SNI.
- *AT+CIPSSLCALPN* : Config SSL client ALPN.
- *AT+CIPSSLCPSK* : Config SSL client PSK.
- *AT+CIPRECONNINTV*: Set Wi-Fi transparent transmitting auto-connect interval.
- *AT+CIPRECVMODE*: Set Socket Receive Mode.
- *AT+CIPRECVDATA*: Get Socket Data in Passive Receive Mode.
- *AT+CIPRECVLEN*: Get Socket Data Length in Passive Receive Mode.
- *AT+PING*: Ping Packets
- *AT+CIPDNS* : Configures Domain Name System.
- *AT+CIPTCPOPT* : Configurates the socket options.

3.3.1 AT+CIPSTATUS—Gets the Connection Status

Execute Command:

```
AT+CIPSTATUS
```

Response:

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

Parameters:

- **<stat>**: status of the esp device Station interface.
  - 0: The esp device station is inactive.
  - 1: The esp device station is idle.
  - 2: The esp device Station is connected to an AP and its IP is obtained.
  - 3: The esp device Station has created a TCP or SSL transmission.
  - 4: The TCP or SSL transmission of esp device Station is disconnected.
  - 5: The esp device Station does NOT connect to an AP.
- **<link ID>**: ID of the connection (0~4), used for multiple connections.
- **<type>**: string parameter, “TCP” or “UDP”.
- **<remote IP>**: string parameter indicating the remote IP 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.

3.3.2 AT+CIPDOMAIN—Domain Name Resolution Function

Execute Command:

```
AT+CIPDOMAIN=<domain name>
```

Response:

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

Parameter:

- **<domain name>**: the domain name.

Example:

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

3.3.3 AT+CIPSTART—Establishes TCP Connection, UDP Transmission or SSL Connection

Establish TCP Connection

Set Command:

```
Single TCP connection (AT+CIPMUX=0):
AT+CIPSTART=<type>,<remote IP>,<remote port>[,<TCP keep alive>][,<local IP>]
Multiple TCP Connections (AT+CIPMUX=1):
AT+CIPSTART=<link ID>,<type>,<remote IP>,<remote port>[,<TCP keep alive>][,<local IP>]
```

Response:

```
OK
```

Or if the TCP connection is already established, the response is: ALREADY CONNECTTED ERROR Parameters:

- **<link ID>**: ID of network connection (0~4), used for multiple connections.
- **<type>**: string parameter indicating the connection type: “TCP”, “UDP” or “SSL”.
- **<remote IP>**: string parameter indicating the remote IP address.
- **<remote port>**: the remote port number.
- **[<TCP keep alive>]**(optional parameter): detection time interval when TCP is kept alive; this function is disabled by default.
  - 0: disable TCP keep-alive.
  - 1 ~ 7200: detection time interval; unit: second (s).

- **[<local IP>]**(optional parameter): select which IP want to use, this is useful when using both ethernet and WiFi; this parameter is disabled by default. If you want to use this parameter, must be specified firstly, null also is valid.

Notes:

```
If the remote IP over the UDP is a multicast address(224.0.0.0 ~ 239.255.255.255),␣
↪the esp device will send and receive the UDP multicast;
If the remote IP over the UDP is a broadcast address(255.255.255.255), the esp device␣
↪will send and receive the UDP broadcast.
```

Examples:

```
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"
```

Establish UDP Transmission

Set Command:

```
Single connection (AT+CIPMUX=0):
AT+CIPSTART=<type>,<remote IP>,<remote port>[,(<UDP local port>),(<UDP mode>)][,
↪<local IP>]
Multiple connections (AT+CIPMUX=1):
AT+CIPSTART=<link ID>,<type>,<remote IP>,<remote port>[,(<UDP local port>),(<UDP mode>
↪)][,<local IP>]
```

Response:

```
OK
```

Or if the UDP transmission is already established, the response is: ALREADY CONNECTTED ERROR Parameters:

- **<link ID>**: ID of network connection (0~4), used for multiple connections.
- **<type>**: string parameter indicating the connection type: “TCP”, “UDP” or “SSL”.
- **<remote IP>**: string parameter indicating the remote IP address.
- **<remote port>**: remote port number.
- **[<UDP local port>]**(optional parameter): UDP port of ESP32.
- **[<UDP mode>]**(optional parameter): In the UDP transparent transmission, the value of this parameter has to be 0.
  - 0: the destination peer entity of UDP will not change; this is the default setting.
  - 1: the destination peer entity of UDP can change once.
  - 2: the destination peer entity of UDP is allowed to change.
- **[<local IP>]**(optional parameter): select which IP want to use, this is useful when using both ethernet and WiFi; this parameter is disabled by default. If you want to use this parameter, and must be specified firstly, null also is valid.

Notice:

- To use parameter <UDP mode> , parameter <UDP local port> must be set first.

Example:

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

Establish SSL Connection

Set Command:

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

Response:

```
OK
```

Or if the TCP connection is already established, the response is: ALREADY CONNECTTED ERROR Parameters:

- **<link ID>**: ID of network connection (0~4), used for multiple connections.
- **<type>**: string parameter indicating the connection type: "TCP", "UDP" or "SSL".
- **<remote IP>**: string parameter indicating the remote IP address.
- **<remote port>**: the remote port number.
- **[<TCP keep alive>]**(optional parameter): detection time interval when TCP is kept alive; this function is disabled by default.
  - 0: disable the TCP keep-alive function.
  - 1 ~ 7200: detection time interval, unit: second (s).
- **[<local IP>]**(optional parameter): select which IP want to use, this is useful when using both ethernet and WiFi; this parameter is disabled by default. If you want to use this parameter, must be specified firstly, null also is valid.

Notes:

- SSL connection count depends on available memory and maximum connection count.
- SSL connection does not support UART-WiFi passthrough mode (transparent transmission).
- SSL connection needs a large amount of memory; otherwise, it may cause system reboot.
- If the `AT+CIPSTART` is based on a TLS connection, the timeout of each packet is 10s, then the total timeout will be much longer depending on the handshake packets count.

Example:

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

3.3.4 AT+CIPSTARTEX—Establishes TCP connection, UDP transmission or SSL connection with automatically assigned ID

This command is similar to *AT+CIPSTART*, but you need not to assign an ID by yourself when it is the multiple connections mode (AT+CIPMUX=1), the system will assign an ID to the new connection automatically.

3.3.5 AT+CIPSEND—Sends Data

Set Command:

```
Single connection: (+CIPMUX=0)
AT+CIPSEND=<length>
Multiple connections: (+CIPMUX=1)
AT+CIPSEND=<link ID>,<length>
Remote IP and ports can be set in UDP transmission:
AT+CIPSEND=[<link ID>,]<length>[,<remote IP>,<remote port>]
Function: to configure the data length in normal transmission mode.
```

Response:

```
OK
>
```

Begin receiving serial data. When the requirement of data length is met, the transmission of data starts. If the connection cannot be established or gets disrupted during data transmission, the system returns:

```
ERROR
```

If data is transmitted successfully, the system returns:

```
SEND OK
```

Execute Command:

```
AT+CIPSEND
Function: to start sending data in transparent transmission mode.
```

Response:

```
OK
>
```

Enter transparent transmission, with a 20-ms interval between each packet, and a maximum of 2048 bytes per packet. When a single packet containing +++ is received, ESP32 returns to normal command mode. Please wait for at least one second before sending the next AT command. This command can only be used in transparent transmission mode which requires single connection. For UDP transparent transmission, the value of has to be 0 when using AT+CIPSTART.

Or

```
ERROR
```

Parameters:

- **<link ID>**: ID of the connection (0~4), for multiple connections.
- **<length>**: data length, MAX: 2048 bytes.
- **[<remote IP>]**(optional parameter): remote IP can be set in UDP transmission.
- **[<remote port>]**(optional parameter): remote port can be set in UDP transmission.

3.3.6 AT+CIPSENDEX—Sends Data

Set Command:

```
Single connection: (+CIPMUX=0)
AT+CIPSENDEX=<length>
Multiple connections: (+CIPMUX=1)
AT+CIPSENDEX=<link ID>,<length>
Remote IP and ports can be set in UDP transmission:
AT+CIPSENDEX=[<link ID>,]<length>[,<remote IP>,<remote port>]
Function: to configure the data length in normal transmission mode.
```

Response:

```
OK
>
```

Send data of designated length. Wrap return > after the set command. Begin receiving serial data. When the requirement of data length, determined by , is met, or when \0 appears in the data, the transmission starts. If connection cannot be established or gets disconnected during transmission, the system returns:

```
ERROR
```

If data are successfully transmitted, the system returns: SEND OK Parameters:

- **<link ID>**: ID of the connection (0~4), for multiple connections.
- **<length>**: data length, MAX: 2048 bytes.
  - When the requirement of data length, determined by <length>, is met, or when string \0 appears, the transmission of data starts. Go back to the normal command mode and wait for the next AT command.
  - When sending \0, please send it as \\0.

3.3.7 AT+CIPCLOSE—Closes TCP/UDP/SSL Connection

Set Command (for multiple connections):

```
AT+CIPCLOSE=<link ID>
Function: to close TCP/UDP connection.
```

Parameters:

- **<link ID>**: ID number of connections to be closed; when ID=5, all connections will be closed.

Execute Command (for single connection):

```
AT+CIPCLOSE
```

Response:

```
OK
```

3.3.8 AT+CIFSR—Gets the Local IP Address

Execute Command:

```
AT+CIFSR
```

Response:

```
+CIFSR:<SoftAP IP address>
+CIFSR:<Station IP address>
OK
```

Parameters:

- **<IP address>**:
  - IP address of the ESP32 SoftAP;
  - IP address of the ESP32 Station.

Notes:

- Only when the ESP32 Station is connected to an AP can the Station IP can be queried.

3.3.9 AT+CIPMUX—Enables/Disables Multiple Connections

Query Command:

```
AT+CIPMUX?
Function: to query the connection type.
```

Response:

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

Set Command:

```
AT+CIPMUX=<mode>
Function: to set the connection type.
```

Response:

```
OK
```

Parameters:

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

Notes:

- The default mode is single connection mode.
- Multiple connections can only be set when transparent transmission is disabled (`AT+CIPMODE=0`).
- This mode can only be changed after all connections are disconnected.
- If the TCP server is running, it must be deleted (`AT+CIPSERVER=0`) before the single connection mode is activated.

Example:

```
AT+CIPMUX=1
```

3.3.10 AT+CIPSERVER—Deletes/Creates TCP or SSL Server

Set Command:

```
AT+CIPSERVER=<mode>[,<port>][,<SSL>,<SSL CA enable>]
```

Response:

```
OK
```

Parameters:

- **<mode>**:
  - 0: delete server.
  - 1: create server.
- **<port>**: port number; 333 by default.
- **[ESP32 Only] [<SSL>]**(optional parameter): string "SSL", to set a SSL server
- **[ESP32 Only] [<SSL CA enable>]**(optional parameter):
  - 0: disable CA.
  - 1: enable CA.

Notes:

- A TCP server can only be created when multiple connections are activated (`AT+CIPMUX=1`).
- A server monitor will automatically be created when the TCP server is created. And only one server is allowed.
- When a client is connected to the server, it will take up one connection and be assigned an ID.

Example:

```
// To create a TCP server
AT+CIPMUX=1
AT+CIPSERVER=1,80
// To create a SSL server
AT+CIPMUX=1
AT+CIPSERVER=1,443,"SSL",1
```

3.3.11 AT+CIPSERVERMAXCONN—Set the Maximum Connections Allowed by Server

Query Command:

```
AT+CIPSERVERMAXCONN?
Function: obtain the maximum number of clients allowed to connect to the TCP or SSL␣
↪server.
```

Response:

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

Set Command:

```
AT+CIPSERVERMAXCONN=<num>
Function: set the maximum number of clients allowed to connect to the TCP or SSL␣
↪server.
```

Response:

```
OK
```

Parameters:

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

Notes:

- To set this configuration, you should call the command `AT+CIPSERVERMAXCONN=<num>` before creating a server.

Example:

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

3.3.12 AT+CIPMODE—Configures the Transmission Mode

Query Command:

```
AT+CIPMODE?
Function: to obtain information about transmission mode.
```

Response:

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

Set Command:

```
AT+CIPMODE=<mode>
Function: to set the transmission mode.
```

Response:

```
OK
```

Parameters:

- **<mode>**:
  - 0: normal transmission mode.
  - 1: UART-Wi-Fi passthrough mode (transparent transmission), which can only be enabled in TCP single connection mode or in UDP mode when the remote IP and port do not change.

Notes:

- The configuration changes will NOT be saved in flash.
- During the UART-WiFi passthrough transmission, if the TCP connection breaks, ESP32 will keep trying to reconnect until +++ is input to exit the transmission.

- If it is a normal TCP transmission and the TCP connection breaks, ESP32 will give a prompt and will not attempt to reconnect.

Example:

```
AT+CIPMODE=1
```

3.3.13 AT+SAVETRANSLINK—Saves the Transparent Transmission Link in Flash

Save TCP Single Connection in Flash

Set Command:

```
AT+SAVETRANSLINK=<mode>,<remote IP or domain name>,<remote port>[,<type>,<TCP keep␣
↪alive>]
```

Response:

```
OK
```

Parameters:

- **<mode>**:
  - 0: normal mode, ESP32 will NOT enter UART-WiFi passthrough mode on power-up.
  - 1: ESP32 will enter UART-WiFi passthrough mode on power-up.
- **<remote IP>**: remote IP or domain name.
- **<remote port>**: remote port.
- **[<type>]**(optional parameter): TCP or UDP, TCP by default.
- **[<TCP keep alive>]**(optional parameter): TCP is kept alive. This function is disabled by default.
  - 0: disables the TCP keep-alive function.
  - 1 ~ 7200: keep-alive detection time interval; unit: second (s).

Notes:

- This command will save the UART-WiFi passthrough mode and its link in the NVS area. ESP32 will enter the UART-WiFi passthrough mode on any subsequent power cycles.
- As long as the remote IP (or domain name) and port are valid, the configuration will be saved in flash.

Example:

```
AT+SAVETRANSLINK=1,"192.168.6.110",1002,"TCP"
```

Save UDP Transmission in Flash

Set Command:

```
AT+SAVETRANSLINK=<mode>,<remote IP>,<remote port>,<type>[,<UDP local port>]
```

Response:

```
OK
```

Parameters:

- **<mode>**:
  - 0: normal mode; ESP32 will NOT enter UART-WiFi passthrough mode on power-up.
  - 1: ESP32 enters UART-WiFi passthrough mode on power-up.
- **<remote IP>**: remote IP or domain name.
- **<remote port>**: remote port.
- **[<type>]**(optional parameter): UDP, TCP by default.
- **[<UDP local port>]**(optional parameter): local port when UDP transparent transmission is enabled on power-up.

Notes:

- This command will save the UART-WiFi passthrough mode and its link in the NVS area. ESP32 will enter the UART-WiFi passthrough mode on any subsequent power cycles.
- As long as the remote IP (or domain name) and port are valid, the configuration will be saved in flash.

Example:

```
AT+SAVETRANSLINK=1,"192.168.6.110",1002,"UDP",1005
```

3.3.14 AT+CIPSTO—Sets the TCP Server Timeout

Query Command:

```
AT+CIPSTO?
Function: to check the TCP server timeout.
```

Response:

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

Set Command:

```
AT+CIPSTO=<time>
Function: to set the TCP server timeout.
```

Response:

```
OK
```

Parameter:

- **<time>**: TCP server timeout within the range of 0 ~ 7200s.

Notes:

- ESP32 configured as a TCP server will disconnect from the TCP client that does not communicate with it until timeout.
- If `AT+CIPSTO=0`, the connection will never time out. This configuration is not recommended.

Example:

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

3.3.15 AT+CIPSNTPCFG — Set the Time Zone and the SNTP Server

Query Command:

```
AT+CIPSNTPCFG?
```

Response:

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

Execute Command:

```
AT+CIPSNTPCFG
Function: to clear the SNTP server information.
```

Response:

```
OK
```

Set Command:

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

Response:

```
OK
```

Parameters:

- **<enable>**:
  - 1: the SNTP server is configured.
  - 0: the SNTP server is not configured.
- **<timezone>**: time zone, range: [-11,13].
- **<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, the following default configuration is used: “cn.ntp.org.cn”, “ntp.sjtu.edu.cn”, “us.pool.ntp.org”.

Example:

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

3.3.16 AT+CIPSNTPTIME—Queries the SNTP Time

Query Command:

```
AT+CIPSNTPTIME?
```

Response:

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

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
```

Note:

- asctime style time is defined at asctime man page

3.3.17 AT+CIUPDATE—Updates the Software Through Wi-Fi

Execute Command:

```
AT+CIUPDATE
Function: OTA the lastest version via TCP from server.
```

Response:

```
+CIPUPDATE:<n>
OK
```

Execute Command:

```
AT+CIUPDATE=<ota mode>[,version]
Function: OTA the specified version from server.
```

Response:

```
+CIPUPDATE:<n>
OK
```

Parameters:

- **<ota mode>**:
  - 0: OTA via TCP
  - 1: OTA via SSL, please ensure `make menuconfig` > `Component config` > `AT` > `OTA based upon ssl` is enabled.
- **<version>**: AT version, for example, `v1.2.0.0`, `v1.1.3.0`,`v1.1.2.0`
- **<n>**:
  - 1: find the server.

  - 2: connect to server.
  - 3: get the software version.
  - 4: start updating.

Example:

```
AT+CIUPDATE
```

Or AT+CIUPDATE=1,"v1.2.0.0"

Notes:

- The speed of the upgrade is susceptible to the connectivity of the network.
- ERROR will be returned if the upgrade fails due to unfavourable network conditions. Please wait for some time before retrying.

Notice:

- If using Espressif's AT BIN, `AT+CIUPDATE` will download a new AT BIN from the Espressif Cloud.
- If using a user-compiled AT BIN, users need to implement their own AT+CIUPDATE FOTA function. esp-at project provides an example of FOTA.
- It is suggested that users call `AT+RESTORE` to restore the factory default settings after upgrading the AT firmware.

3.3.18 AT+CIPDINFO—Shows the Remote IP and Port with "+IPD"

Set Command:

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

Response:

```
OK
```

Parameters:

- **<mode>**:
  - 0: does not show the remote IP and port with "+IPD" and "+CIPRECVDATA".
  - 1: shows the remote IP and port with "+IPD" and "+CIPRECVDATA".

Example:

```
AT+CIPDINFO=1
```

3.3.19 +IPD—Receives Network Data

Command:

```
Single connection:
(+CIPMUX=0)+IPD,<len>[,<remote IP>,<remote port>]:<data>
multiple connections:
(+CIPMUX=1)+IPD,<link ID>,<len>[,<remote IP>,<remote port>]:<data>
```

Parameters:

- **[<remote IP>]**: remote IP string, enabled by command `AT+CIPDINFO=1`.
- **[<remote port>]**: remote port, enabled by command `AT+CIPDINFO=1`.
- **<link ID>**: ID number of connection.
- **<len>**: data length.
- **<data>**: data received.

Note:

- The command is valid in normal command mode. When the module receives network data, it will send the data through the serial port using the `+IPD` command.

3.3.20 AT+CIPSSLCCONF—Config SSL client

Query Command:

```
AT+CIPSSLCCONF?
Function: to get the configuration of each link that running as SSL client.
```

Response:

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

Set Command:

```
Single connection: (+CIPMUX=0)
AT+CIPSSLCCONF=<auth_mode>[,<pki_number>][,<ca_number>]
Multiple connections: (+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 authorization. In this case, <pki_number> and <ca_number> are not required.
  - 1: load cert and private key for server authorization.
  - 2: load CA for client authorize server cert and private key.
  - 3: both authorization.
- **<pki_number>**: the index of cert and private key, if only one cert and private key, the value should be 0.
- **<ca_number>**: the index of CA, if only one CA, the value should be 0.

Notes:

- Send this command before establish SSL connection if you want configuration take effect immediately.

- The configuration changes will be saved in the NVS area. If you use AT+SAVETRANSLINK to set SSL passthrough mode, the ESP will establish an SSL connection based on this configuration after next power on.

3.3.21 AT+CIPSSLCCN—Config SSL client common name

Query Command:

```
AT+CIPSSLCCN?
Function: to get the common name configuration of each link that running as SSL␣
↪client.
```

Response:

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

Set Command:

```
Single connection: (+CIPMUX=0)
AT+CIPSSLCCN=<"common name">
Multiple connections: (+CIPMUX=1)
AT+CIPSSLCCN=<link ID>,<"common name">
```

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.
- **<"common name">**: common name is used to verify the commonName in the certificate sent by the server.

Notes:

- Send this command before establish SSL connection if you want configuration take effect immediately.

3.3.22 AT+CIPSSLCSNI—Config SSL client server name indication

Query Command:

```
AT+CIPSSLCSNI?
Function: to get the SNI configuration of each link that running as SSL client.
```

Response:

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

Set Command:

```
Single connection: (+CIPMUX=0)
AT+CIPSSLCSNI=<"sni">
Multiple connections: (+CIPMUX=1)
AT+CIPSSLCSNI=<link ID>,<"sni">
```

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.
- **<"sni">**: Set TLS extension servername (SNI) in ClientHello

Notes:

- Send this command before establish SSL connection if you want configuration take effect immediately.

3.3.23 AT+CIPSSLCALPN—Config SSL client application layer protocol negotiation(ALPN)

Query Command:

```
AT+CIPSSLCALPN?
Function: to get the ALPN configuration of each link that running as SSL client.
```

Response:

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

Set Command:

```
Single connection: (+CIPMUX=0)
AT+CIPSSLCALPN=<counts>,<"alpn">[,<"alpn">[,<"alpn">]]
Multiple connections: (+CIPMUX=1)
AT+CIPSSLCALPN=<link ID>,<counts>,<"alpn">[,<"alpn">[,<"alpn">]]
```

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.
- **<counts>**: ALPN counts
- **<"alpn">**: Set ALPN in ClientHello

Notes:

- Send this command before establish SSL connection if you want configuration take effect immediately.

3.3.24 AT+CIPSSLCPSK—Config SSL client pre-shared key(PSK)

Query Command:

```
AT+CIPSSLCPSK?
Function: to get the PSK configuration of each link that running as SSL client.
```

Response:

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

Set Command:

```
Single connection: (+CIPMUX=0)
AT+CIPSSLCPSK=<"psk">,<"hint">
Multiple connections: (+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, maxlen is 32.
- **<"hint">**: PSK hint, maxlen is 32.

Notes:

- Send this command before establish SSL connection if you want configuration take effect immediately.
- This command is not currently supported on ESP32.

3.3.25 AT+CIPRECONNINTV—Set Wi-Fi transparent transmitting auto-connect interval

Query Command:

```
AT+CIPRECONNINTV?
Function: to get Wi-Fi transparent transmitting auto-connect interval .
```

Response:

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

Set Command:

```
AT+CIPRECONNINTV=<interval>
Function: to set the interval of auto reconnecting when the TCP/UDP/SSL transmission␣
↪broke in UART-WiFi transparent mode.
```

Parameters:

- **<interval>**: Time interval for automatic reconnection, default is 1, range is 1~36000, unit is 100ms.

Example:

```
AT+CIPRECONNINTV=10
```

Notes:

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

3.3.26 +IPD—Receives Network Data

Command:

```
Single connection:
(+CIPMUX=0)+IPD,<len>[,<remote IP>,<remote port>]:<data>
multiple connections:
(+CIPMUX=1)+IPD,<link ID>,<len>[,<remote IP>,<remote port>]:<data>
```

Parameters:

- **[<remote IP>]**: remote IP, enabled by command `AT+CIPDINFO=1`.
- **[<remote port>]**: remote port, enabled by command `AT+CIPDINFO=1`.
- **<link ID>**: ID number of connection.
- **<len>**: data length.
- **<data>**: data received.

Note:

- The command is valid in normal command mode. When the module receives network data, it will send the data through the serial port using the `+IPD` command.

3.3.27 AT+CIPRECVMODE—Set Socket Receive Mode

Query Command:

```
AT+CIPRECVMODE?
Function: to query socket receive mode.
```

Response:

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

Set Command:

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

Response:

```
OK
```

Parameters:

- **<mode>**: the receive mode of socket data is active mode by default.
  - 0: active mode - ESP-AT will send all the received socket data instantly to host MCU through UART with header "+IPD".
  - 1: passive mode - ESP-AT will keep the received socket data in an internal buffer (default is 5744 bytes), and wait for host MCU to read the data. If the buffer is full, the socket transmission will be blocked.

Example:

```
AT+CIPRECVMODE=1
```

Notes:

- The configuration is for TCP and SSL transmission only, and can not be used on WiFi-UART passthrough mode. If it is a UDP transmission in passive modedata will be missed when buffer full.
- If the passive mode is enabled, when ESP-AT receives socket data, it will prompt the following message in different scenarios:
  - for multiple connection 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 buffer
  - User should read data by command `AT+CIPRECVDATA` once there is a `+IPD` reported. Otherwise, the next `+IPD` will not be reported to the host MCU, until the previous `+IPD` be read by command `AT+CIPRECVDATA`.
  - In a case of disconnection, the buffered socket data will still be there and can be read by MCU until you send `AT+CIPCLOSE`. Specially, 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`.

3.3.28 AT+CIPRECVDATA—Get Socket Data in Passive Receive Mode

Set Command:

```
Single connection: (+CIPMUX=0)
AT+CIPRECVDATA=<len>
Multiple connections: (+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 connection mode.
- **<len>**: the max value is up to 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 get
- **<data>**: the data you get
- **[<remote IP>]**: remote IP string, enabled by command `AT+CIPDINFO=1`.
- **[<remote port>]**: remote port, enabled by command `AT+CIPDINFO=1`.

Example:

```
AT+CIPRECVMODE=1
For example, if host MCU gets a message of receiving 100 bytes data in connection
↪with No.0, the message will be as following: +IPD,0,100
then you can read those 100 bytes by using the command below
AT+CIPRECVDATA=0,100
```

3.3.29 AT+CIPRECVLEN—Get Socket Data Length in Passive Receive Mode

Query Command:

```
AT+CIPRECVLEN?
Function: query the length of the entire data buffered for the link.
```

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 link

Example:

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

Notes:

- For ssl link, it will return the length of encrypted data, so the returned length will be more than the real data length.

3.3.30 AT+PING: Ping Packets

Set Command:

```
AT+PING=<IP>
Function: Ping packets.
```

Response:

```
+PING:<time>

OK
```

or

```
+timeout

ERROR
```

Parameters:

- **<IP>**: string; host IP or domain name

- **<time>**: the response time of ping, unit: millisecond.

Example:

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

3.3.31 AT+CIPDNS : Configures Domain Name System.

Query Command:

```
AT+CIPDNS?
Function: to obtain current Domain Name System information.
```

Response:

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

Set Command:

```
AT+CIPDNS=<enable>[,<"DNS IP1">,<"DNS IP2">,<"DNS IP3">]
Function: Configures Domain Name System.
```

Response:

```
OK
```

or

```
ERROR
```

Parameters:

- **<enable>**:
  - 0: Enable automatic DNS settings from DHCP, the DNS will be restore to `222.222.67.208`, only when DHCP is updated will it take effect.
  - 1: Enable manual DNS settings, if not set `DNS IP`, It will use `222.222.67.208` by default.
- **<DNS IP1>**: the first DNS IP. For set command, only for manual DNS settings; for query command, it is current DNS setting.
- **<DNS IP2>**: the second DNS IP. For set command, only for manual DNS settings; for query command, it is current DNS setting.
- **<DNS IP3>**: the third DNS IP. For set command, only for manual DNS settings; for query command, it is current DNS setting.

Example:

```
AT+CIPDNS=0
AT+CIPDNS=1,"222.222.67.208","114.114.114.114","8.8.8.8"
```

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.

- The DNS server may change according to the configuration of the router which the ESP chip connected to.

3.3.32 AT+CIPTCPOPT : Configurates the socket options.

Query Command:

```
AT+CIPTCPOPT?
Function: to obtain current socket options information.
```

Response:

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

Set 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>**: configurate the SO_LINGER options for socket, in second, default: -1.
  - = -1: off
  - = 0: on, linger time = 0
  - 0: on, linger time = <so_linger>
- **<tcp_nodelay>**: configurate the TCP_NODELAY options for socket.
  - 0: disable TCP_NODELAY, default
  - 1: enable TCP_NODELAY
- **<so_sndtimeo>**: configurate the SO_SNDTIMEO options for socket, in millisecond, default: 0.

3.4 [ESP32 Only] BLE AT Commands

Download BLE Spec (ESP32 supports Core Version 4.2)

- [ESP32 Only] *AT+BLEINIT* : Bluetooth Low Energy (BLE) initialization
- [ESP32 Only] *AT+BLEADDR* : Sets BLE device's address
- [ESP32 Only] *AT+BLENAME* : Sets BLE device's name

- [ESP32 Only] *AT+BLESCANPARAM* : Sets parameters of BLE scanning
- [ESP32 Only] *AT+BLESCAN* : Enables BLE scanning
- [ESP32 Only] *AT+BLESCANRSPDATA* : Sets BLE scan response
- [ESP32 Only] *AT+BLEADVPARAM* : Sets parameters of BLE advertising
- [ESP32 Only] *AT+BLEADVDATA* : Sets BLE advertising data
- [ESP32 Only] *AT+BLEADVDATAEX* : Auto sets BLE advertising data
- [ESP32 Only] *AT+BLEADVSTART* : Starts BLE advertising
- [ESP32 Only] *AT+BLEADVSTOP* : Stops BLE advertising
- [ESP32 Only] *AT+BLECONN* : Establishes BLE connection
- [ESP32 Only] *AT+BLECONNPARAM* : Updates parameters of BLE connection
- [ESP32 Only] *AT+BLEDISCONN* : Ends BLE connection
- [ESP32 Only] *AT+BLEDATALEN* : Sets BLE data length
- [ESP32 Only] *AT+BLECFGMTU* : Sets BLE 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* : GATTS notifies of characteristics
- [ESP32 Only] *AT+BLEGATTSIND* : GATTS indicates characteristics
- [ESP32 Only] *AT+BLEGATTSSETATTR* : GATTS sets attributes
- [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* : Sets BLE spp parameters
- [ESP32 Only] *AT+BLESPP* : Enter BLE spp mode
- [ESP32 Only] *AT+BLESECPARAM* : Set BLE encryption parameters
- [ESP32 Only] *AT+BLEENC* : Initiate BLE encryption request
- [ESP32 Only] *AT+BLEENCRSP* : Grant security request access.
- [ESP32 Only] *AT+BLEKEYREPLY* : Reply the key value to the peer device in the lagecy connection stage.
- [ESP32 Only] *AT+BLECONFREPLY* : Reply the comfirm value to the peer device in the lagecy connection stage.
- [ESP32 Only] *AT+BLEENCDEV* : Query BLE encryption device list
- [ESP32 Only] *AT+BLEENCCLEAR* : Clear BLE encryption device list

- [ESP32 Only] *AT+BLESETKEY* : Set BLE static pair key
- [ESP32 Only] *AT+BLEHIDINIT* : BLE HID device profile initialization
- [ESP32 Only] *AT+BLEHIDKB* : Send BLE HID Keyboard information
- [ESP32 Only] *AT+BLEHIDMUS* : Send BLE HID mouse information
- [ESP32 Only] *AT+BLEHIDCONSUMER* : Send BLE HID consumer information
- [ESP32 Only] *AT+BLUFI* : Start or Stop BLUFI
- [ESP32 Only] *AT+BLUFINAME* : Set BLUFI device name

3.4.1 [ESP32 Only] AT+BLEINIT—BLE Initialization

Query Command:

```
AT+BLEINIT?
Function: to check the initialization status of BLE.
```

Response:

If BLE is not initialized, it will return

```
+BLEINIT:0
OK
```

If BLE is initialized, it will return

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

Set Command:

```
AT+BLEINIT=<init>
Function: to initialize the role of BLE.
```

Response:

```
OK
```

Parameter:

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

Notes:

- at_customize.bin has to be downloaded, so that the relevant commands can be used. Please refer to the ESP32_Customize_Partitions for more details.
- Before using BLE AT commands, this command has to be called first to trigger the initialization process.
- After being initialized, the BLE role cannot be changed. User needs to call AT+RST to restart the system first, then re-init the BLE role.
- If using ESP32 as a BLE 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/partitions_at.csv.

Example:

```
AT+BLEINIT=1
```

3.4.2 [ESP32 Only] AT+BLEADDR—Sets BLE Device's Address

Query Command:

```
AT+BLEADDR?
Function: to get the BLE public address.
```

Response:

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

Set Command:

```
AT+BLEADDR=<addr_type>[,<random_addr>]
Function: to set the BLE address type.
```

Response:

```
OK
```

Parameter:

- **<addr_type>**:
  - 0: public address
  - 1: random address

Notes:

- A static address shall meet the following requirements:
  - The two most significant bits of the address shall be equal to 1
  - At least one bit of the random part of the address shall be 0
  - At least one bit of the random part of the address shall 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—Sets BLE Device's Name

Query Command:

```
AT+BLENAME?
Function: to get the BLE device name.
```

Response:

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

Set Command:

```
AT+BLENAME=<device_name>
Function: to set the BLE device name, The maximum length is 32.
```

Response:

```
OK
```

Parameter:

- **<device_name>**: the BLE device name

Notes:

- The configuration changes will be saved in the NVS area if `AT+SYSSTORE=1`.
- The default BLE device name is "BLE_AT".

Example:

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

3.4.4 [ESP32 Only] AT+BLESCANPARAM—Sets Parameters of BLE Scanning

Query Command:

```
AT+BLESCANPARAM?
Function: to get the parameters of BLE scanning.
```

Response:

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

Set Command:

```
AT+BLESCANPARAM=<scan_type>,<own_addr_type>,<filter_policy>,<scan_interval>,<scan_
↪window>
Function: to set the parameters of BLE scanning.
```

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

Notes:

- <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—Enables BLE Scanning

Set Command:

```
AT+BLESCAN=<enable>[,<interval>][,<filter_type>,<filter_param>]
Function: to enable/disable scanning.
```

Response:

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

Parameters:

- **<enable>**:
  - 0: disable continuous scanning
  - 1: enable continuous scanning
- **[<interval>]**: optional parameter, unit: second
  - When disabling the scanning, this parameter should be omitted
  - When enabling the scanning, and the `<interval>` is 0, it means that scanning is continuous
  - When enabling the scanning, and the `<interval>` is NOT 0, for example, command `AT+BLESCAN=1,3`, it means that scanning should last for 3 seconds and then stop automatically, so that the scanning results be returned.
- **<filter_type>**: Filtering option, 1:MAC or 2:"NAME".
- **<filter_param>**: Filtering param, remote evice mac address or remote device name.

- **<addr>**: BLE address
- **<rssi>**: signal strength
- **<adv_data>**: advertising data
- **<scan_rsp_data>**: scan response data
- **<addr_type>**: the address type of broadcasters

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—Sets BLE Scan Response

Set Command:

```
AT+BLESCANRSPDATA=<scan_rsp_data>
Function: to set scan response.
```

Response:

```
OK
```

Parameter:

- **<scan_rsp_data>**: scan response data is a HEX string.
  - For example, to set the response data as "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—Sets Parameters of Advertising

Query Command:

```
AT+BLEADVPARAM?
Function: to query the parameters of advertising.
```

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:

```
AT+BLEADVPARAM=<adv_int_min>,<adv_int_max>, <adv_type>,<own_addr_type>,<channel_map>[,
↪<adv_filter_policy>][,<peer_addr_type>] [,<peer_addr>]
Function: to set the parameters of advertising.
```

Response:

```
OK
```

Parameters:

- **<adv_int_min>**: minimum value of advertising interval; range: 0x0020 ~ 0x4000
- **<adv_int_max>**: maximum value of advertising interval; range: 0x0020 ~ 0x4000
- **<adv_type>**:
  - 0ADV_TYPE_IND
  - 2ADV_TYPE_SCAN_IND
  - 3ADV_TYPE_NONCONN_IND
- **<own_addr_type>**own BLE address type
  - 0BLE_ADDR_TYPE_PUBLIC
  - 1BLE_ADDR_TYPE_RANDOM
- **<channel_map>**channel of advertising
  - 1ADV_CHNL_37
  - 2ADV_CHNL_38
  - 4ADV_CHNL_39
  - 7ADV_CHNL_ALL
- **[<adv_filter_policy>]**(optional parameter)filter policy of advertising
  - 0ADV_FILTER_ALLOW_SCAN_ANY_CON_ANY
  - 1ADV_FILTER_ALLOW_SCAN_WLST_CON_ANY
  - 2ADV_FILTER_ALLOW_SCAN_ANY_CON_WLST
  - 3ADV_FILTER_ALLOW_SCAN_WLST_CON_WLST
- **[<peer_addr_type>]**(optional parameter)remote BLE address type
  - 0PUBLIC
  - 1RANDOM
- **[<peer_addr>]**(optional parameter)remote BLE 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—Sets Advertising Data

Set Command:

```
AT+BLEADVDATA=<adv_data>
Function: to set advertising data.
```

Response:

```
OK
```

Parameters:

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

Notes:

- If advertising data is preset by command `AT+BLEADVDATAEX=<dev_name>,<uuid>,<manufacturer_data>,<include_power>`, it will be over write by `AT+BLEADVDATA=<adv_data>`.

Example:

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

3.4.9 [ESP32 Only] AT+BLEADVDATAEX—Auto sets BLE advertising data

Query Command:

```
AT+BLEADVDATAEX?
Function: to query the parameters of advertising data.
```

Response:

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

OK
```

Set Command:

```
AT+BLEADVSTART=<dev_name>,<uuid>,<manufacturer_data>,<include_power>
Function: configurate the adv data and start advertising.
```

Response:

```
OK
```

Parameters:

- **<dev_name>**: device name; this is a string. For example:
  - to set the device name "just-test", the command should be `AT+BLEADVSTART="just-test",<uuid>,<manufacturer_data>,<include_power>`
- **<uuid>**: this is a string. For example:
  - to set the UUID "0xA002", the command should be `AT+BLEADVSTART=<dev_name>,"A002",<manufacturer_data>,<include_power>`.

- **<manufacturer_data>**: manufacturer data; this is a HEX string, For example:
  - to set the manufacturer data as "0x11 0x22 0x33 0x44 0x55", the command should be `AT+BLEADVSTART=<dev_name>,<uuid>,"1122334455",<include_power>`.
- **<include_power>**: if User need include the tx power in the advertising data, this param should be set `1`, if not, this param should be set `0`

Notes:

- If advertising data is preset by command `AT+BLEADVDATA=<adv_data>`, it will be over write by `AT+BLEADVDATAEX=<dev_name>,<uuid>,<manufacturer_data>,<include_power>`

Example:

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

- [ESP32 Only] *AT+BLEADVDATAEX* : Auto sets BLE advertising data

3.4.10 [ESP32 Only] AT+BLEADVSTART—Starts Advertising

Execute Command:

```
AT+BLEADVSTART
Function: to start advertising.
```

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 all zeros data will be sent.
- If advertising data is preset by command `AT+BLEADVDATA=<adv_data>`, it will be over write 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—Stops Advertising

Execute Command:

```
AT+BLEADVSTOP
Function: to stop advertising.
```

Response:

```
OK
```

Notes:

- After having started advertising, if the BLE 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—Establishes BLE connection

Query Command:

```
AT+BLECONN?
Function: to query the BLE connection.
```

Response:

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

If the connection has not been established, there will NOT be <conn_index> and <remote_address> Set Command:

```
AT+BLECONN=<conn_index>,<remote_address>[,<addr_type>,<timeout>]
Function: to establish the BLE connection, the address_type is an optional parameter.
```

Response:

```
OK
```

It will prompt the message below, if the connection is established successfully:

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

It will prompt the message below, if NOT:

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

Parameters:

- **<conn_index>**: index of BLE connection, range [0~2].
- **<remote_address>**remote BLE address
- **<addr_type>**: the address type of broadcasters
- **<timeout>**: the timeout for the connection command, range is [3,30] second.

Example:

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

Notes:

- It is recommended to scan devices by "AT+BLESCAN" before initiating a new connection to ensure that the target device is in broadcast state.

- The maximum timeout for connection is 30 seconds.

3.4.13 [ESP32 Only] AT+BLECONNPARAM—Updates parameters of BLE connection

Query Command:

```
AT+BLECONNPARAM?
Function: to query the parameters of BLE connection.
```

Response:

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

Set Command:

```
AT+BLECONNPARAM=<conn_index>,<min_interval>,<max_interval>,<latency>,<timeout>
Function: to update the parameters of BLE connection.
```

Response:

```
OK  // command received
```

If the setting failed, it will prompt message below:

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

Parameters:

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

Notes:

- This commands supports the client only 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—Ends BLE connection

Execute Command:

```
AT+BLEDISCONN=<conn_index>
Function: to end the BLE connection.
```

Response:

```
OK  // the AT+BLEDISCONN command is received
If the command is successful, it will prompt + BLEDISCONN:<conn_index>,<remote_
↪address>
```

Parameter:

- **<conn_index>**: index of BLE connection, range [0~2].
- **<remote_address>**: remote BLE address

Notes:

- Only client can call this command to break 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—Sets BLE Data Packet Length

Set Command:

```
AT+BLEDATALEN=<conn_index>,<pkt_data_len>
Function: to set the length of BLE data packet.
```

Response:

```
OK
```

Parameter:

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

Notes:

- The BLE 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—Sets BLE MTU Length

Query Command:

```
AT+BLECFGMTU?
Function: to query the length of the maximum transmission unit (MTU).
```

Response:

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

Set Command:

```
AT+BLECFGMTU=<conn_index>,<mtu_size>
Function: to set the length of the maximum transmission unit (MTU).
```

Response:

```
OK  // the command is received
```

Parameter:

- **<conn_index>**: index of BLE connection, range [0~2].
- **<mtu_size>**: MTU length

Notes:

- Only the client can call this command to set the length of MTU. However, the BLE connection has to be established first.
- The actual length of MTU needs to be negotiated. The "OK" response only means that the MTU length must be set. So, the user should use command `AT+BLECFGMTU?` to query the actual MTU 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:

```
AT+BLEGATTSSRVCRE
Function: The Generic Attributes Server (GATTS) creates BLE services.
```

Response:

```
OK
```

Notes:

- If using ESP32 as a BLE 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/partitions_at.csv.
- This command should be called immediately to create services, right after the BLE server is initialized.
- If a BLE connection is established first, the service creation will fail.

Example:

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

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

Execute Command:

```
AT+BLEGATTSSTART
Function: GATTS starts all services.
```

Set Command:

```
AT+BLEGATTSSRVSTART=<srv_index>
Function: GATTS starts a specific service.
```

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:

```
AT+BLEGATTSSTOP
Function: GATTS stops all services.
```

Set Command:

```
AT+BLEGATTSSRVSTOP=<srv_index>
Function: GATTS stops a specific service.
```

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:

```
AT+BLEGATTSSRV?
Function: GATTS services discovery.
```

Response:

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

Parameters:

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

Example:

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

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

Query Command:

```
AT+BLEGATTSCHAR?
Function: GATTS characteristics discovery.
```

Response:

When showing a characteristic, it will be as:

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

When showing a descriptor, it will be as:

```
+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—GATTS Notifies of Characteristics

Set Command:

```
AT+BLEGATTSNTFY=<conn_index>,<srv_index>,<char_index>,<length>
Function: GATTS to notify of its characteristics.
```

Response:

```
>
```

Begin receiving serial data. When the requirement of data length, determined by , is met, the notification starts. If the data transmission is successful, the system returns: OK Parameters:

- **<conn_index>**: index of BLE 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 // starts advertising. After the client is connected, it must be␣
↪configured to receive notifications.
AT+BLEGATTSCHAR?  // check which characteristic the client will be notified of
// for example, to notify of 4 bytes of data using the 6th characteristic in the 3rd␣
↪service, use the following command:
AT+BLEGATTSNTFY=0,3,6,4
// after > shows, inputs 4 bytes of data, such as "1234"; then, the data will be␣
↪transmitted automatically
```

3.4.23 [ESP32 Only] AT+BLEGATTSIND—GATTS Indicates Characteristics

Set Command:

```
AT+BLEGATTSIND=<conn_index>,<srv_index>,<char_index>,<length>
Function: GATTS indicates its characteristics.
```

Response:

```
>
```

Begin receiving serial data. When the requirement of data length, determined by , is met, the indication starts. If the data transmission is successful, the system returns: OK Parameters:

- **<conn_index>**: index of BLE 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 // starts advertising. After the client is connected, it must be␣
↪configured to receive indications.
AT+BLEGATTSCHAR?  // check for which characteristic 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 > shows, inputs 4 bytes of data, such as "1234"; then, the data will be␣
↪transmitted automatically
```

3.4.24 [ESP32 Only] AT+BLEGATTSSETATTR—GATTS Sets Characteristic

Set Command:

```
AT+BLEGATTSSETATTR=<srv_index>,<char_index>[,<desc_index>],<length>
Function: GATTS to set its characteristic (descriptor).
```

Response:

```
>
```

Begin receiving serial data. When the requirement of data length, determined by , is met, the setting starts. If the setting is successful, the system 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>]**(Optional parameter): descriptor's index.
  - If it is set, this command is used to set the value of the descriptor; if it is not, this command is used to set the value of the characteristic.
- **<length>**: data length

Note:

- If the <value> length is larger than the maximum length allowed, the setting will fail.

Example:

```
AT+BLEINIT=2   // role: server
AT+BLEGATTSSRVCRE
AT+BLEGATTSSRVSTART
AT+BLEGATTSCHAR?
// for example, to set 4 bytes of data of the 1st characteristic in the 1st service,␣
↪use the following command:
```

(continues on next page)

(continued from previous page)

```
AT+BLEGATTSSETATTR=1,1,,4
// after > shows, inputs 4 bytes of data, such as "1234"; then, the setting starts
```

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

Query Command:

```
AT+BLEGATTCPRIMSRV=<conn_index>
Function: GATTC to discover primary services.
```

Response:

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

Parameters:

- **<conn_index>**: index of BLE connection, range [0~2].
- **<srv_index>**: service's index starting from 1
- **<srv_uuid>**: service's UUID
- **<srv_type>**: service's type
  - 0is not a primary service
  - 1is a primary service

Note:

- The BLE 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:

```
AT+BLEGATTCINCLSRV=<conn_index>,<srv_index>
Function: GATTC to discover included services.
```

Response:

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

Parameters:

- **<conn_index>**: index of BLE connection; only 0 is supported for the single connection right now, but multiple BLE connections will be supported in the future.

- **<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
  - 0is not a primary service
  - 1is a primary service
- **<included_srv_uuid>**: included service's UUID
- **<included_srv_type>**: included service's type
  - 0is not a primary service
  - 1is a primary service

Note:

- The BLE 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:

```
AT+BLEGATTCCHAR=<conn_index>,<srv_index>
Function: GATTC to discover characteristics.
```

Response:

When showing a characteristic, it will be as:

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

When showing a descriptor, it will be as:

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

Parameters:

- **<conn_index>**: index of BLE 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 BLE 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
```

3.4.28 [ESP32 Only] AT+BLEGATTCRD—GATTC Reads a Characteristic

Set Command:

```
AT+BLEGATTCRD=<conn_index>,<srv_index>,<char_index>[,<desc_index>]
Function: GATTC to read a characteristic or descriptor.
```

Response

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

Parameters

- **<conn_index>**: index of BLE 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>]**(Optional parameter): 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".

Note:

- The BLE 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 Characteristic

Set Command:

```
AT+BLEGATTCWR=<conn_index>,<srv_index>,<char_index>[,<desc_index>],<length>
Function: GATTC to write characteristics or descriptor.
```

Response

```
>
```

Begin receiving serial data. When the requirement of data length, determined by , is met, the writting starts. If the setting is successful, the system returns: OKParameters

- **<conn_index>**: index of BLE 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>]**(Optional parameter): 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

Note:

- The BLE 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 > shows, inputs 6 bytes of data, such as "123456"; then, the writing starts
```

3.4.30 [ESP32 Only] AT+BLESPPCFG—Sets BLE spp parameters

Query Command:

```
AT+BLESPPCFG?
Function: to get the parameters of BLE spp.
```

Response:

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

Set Command:

```
AT+BLESCANPARAM=<option>[,<tx_service_index>,<tx_char_index>,<rx_service_index>,<rx_
↪char_index>]
Function: to set or reset the parameters of BLE spp.
```

Response:

```
OK
```

Parameters:

- **<option>**: if the option is 0, it means all the spp parametersthe will be reset, and the next four parameters don't need input. if the option is 1, the user must input all the parameters.
- **<tx_service_index>**: tx service's index; it can be fetched with command `AT+BLEGATTCPRIMSRV=<conn_index>` and `AT+BLEGATTSSRVCRE?`
- **<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+BLEGATTSSRVCRE?`
- **<rx_char_index>**: rx characteristic's index; it can be fetched with command `AT+BLEGATTCCHAR=<conn_index>,<srv_index>` and `AT+BLEGATTSCHAR?`

Note:

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

Example:

```
AT+BLESPPCFG=0          // reset ble spp parameters
AT+BLESPPCFG=1,3,5,3,7  // set ble spp parameters
AT+BLESPPCFG?           // query ble spp parameters
```

3.4.31 [ESP32 Only] AT+BLESPP—Enter BLE spp mode

Execute Command:

```
AT+BLESPP
Function: Enter BLE spp mode.
```

Response:

```
>
```

Note:

- If the ble spp parameters is illegal, this command will return ERROR.

Example:

```
AT+BLESPP   // enter ble spp mode
```

3.4.32 [ESP32 Only] AT+BLESECPARAM—Set BLE encryption parameters

Query Command:

```
AT+BLESECPARAM?
Function: to get the parameters of BLE smp.
```

Response:

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

Set Command:

```
AT+BLESECPARAM=<auth_req>,<iocap>,<key_size>,<init_key>,<rsp_key>[,<auth_option>]
Function: to set the parameters of BLE smp.
```

Response:

```
OK
```

Parameters:

- **<auth_req>**:
  - 0 : NO_BOND
  - 1 : BOND
  - 4 : MITM
  - 8 : SC_ONLY
  - 9 : SC_BOND
  - 12 : SC_MITM
  - 13 : SC_MITM_BOND
- **<iocap>**:
  - 0 : DisplayOnly
  - 1 : DisplayYesNo
  - 2 : KeyboardOnly
  - 3 : NoInputNoOutput
  - 4 : Keyboard displa
- **<key_size>**: the key size should be 7~16 bytes.

- **<init_key>**: combination of the bit pattern.
- **<rsp_key>**: combination of the bit pattern.
- **<auth_option>**: auth option of security.
  - 0 : Select the security level automaticly.
  - 1 : If cannot follow the preset security level, the connection will disconnect.

Note:

- The bit pattern for init_key&rsp_key is:
  - (1<<0) Used to exchange the encrytyption key in the init key & response key
  - (1<<1) Used to exchange the IRK key in the init key & response key
  - (1<<2) Used to exchange the CSRK key in the init key & response key
  - (1<<3) Used to exchange the link key(this key just used in the BLE & 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 BLE encryption request

Set Command:

```
AT+BLEENC=<conn_index>,<sec_act>
Function: to start a pairing request
```

Response:

```
OK
```

Parameters:

- **<conn_index>**: index of BLE connection, range [0~2].
- **<sec_act>**:
  - 0 : SEC_NONE
  - 1 : SEC_ENCRYPT
  - 2 : SEC_ENCRYPT_NO_MITM
  - 3 : SEC_ENCRYPT_MITM

Note:

- Before ipput this command, user must set the security paramsters and connection with remote device.

Example:

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

3.4.34 [ESP32 Only] AT+BLEENCRSP—Grant security request access

Set Command:

```
AT+BLEENCRSP=<conn_index>,<accept>
Function: to set a pairing response.
```

Response:

```
OK
```

Parameters:

- **<conn_index>**: index of BLE connection, range [0~2].
- **<accept>**:
  - 0 : reject
  - 1 : accept;

Example:

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

3.4.35 [ESP32 Only] AT+BLEKEYREPLY—Reply the key value to the peer device in the lagecy connection stage

Set Command:

```
AT+BLEKEYREPLY=<conn_index>,<key>
Function: to reply a pairing key.
```

Response:

```
OK
```

Parameters:

- **<conn_index>**: index of BLE connection, range [0~2].
- **<key>**: pairing key

Example:

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

3.4.36 [ESP32 Only] AT+BLECONFREPLY—Reply the comfirm value to the peer device in the lagecy connection stage

Set Command:

```
AT+BLECONFREPLY=<conn_index>,<confirm>
Function: to reply to a pairing result.
```

Response:

```
OK
```

Parameters:

- **<conn_index>**: index of BLE connection, range [0~2].
- **<confirm>**:
  - 0 : NO
  - 1 : Yes

Example:

```
AT+BLECONFREPLY=0,1
```

3.4.37 [ESP32 Only] AT+BLEENCDEV—Query BLE encryption device list

Query Command:

```
AT+BLEENCDEV?
Function: to get the bounded devices.
```

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 BLE encryption device list

Set Command:

```
AT+BLEENCCLEAR=<enc_dev_index>
Function: remove a device from the security database list with a specific index.
```

Response:

```
OK
```

Execute Command:

```
AT+BLEENCCLEAR
Function: remove all devices from the security database.
```

Response:

```
OK
```

Parameters:

- **<enc_dev_index>**: index of the bonded devices.

Example:

```
AT+BLEENCCLEAR
```

3.4.39 [ESP32 Only]AT+BLESETKEY—Set BLE static pair key

Query Command:

```
AT+BLESETKEY?
Function: to query the ble static pair key, If it's not set, it will returns -1.
```

Response:

```
+BLESETKEY:<static_key>
OK
```

Set Command:

```
AT+BLESETKEY=<static_key>
Function: to set a BLE static pair key for all BLE connections.
```

Response:

```
OK
```

Parameters:

- **<static_key>**: static BLE pair key.

Example:

```
AT+BLESETKEY=123456
```

3.4.40 [ESP32 Only]AT+BLEHIDINIT—BLE HID device profile initialization

Query Command:

```
AT+BLEHIDINIT?
Function: to check the initialization status of BLE HID profile.
```

Response:

If BLE HID device profile is not initialized, it will return:

```
+BLEHIDINIT:0
OK
```

If BLE HID device profile is initialized, it will return:

```
+BLEHIDINIT:1
OK
```

Set Command:

```
AT+BLEHIDINIT=<init>
Function: to initialize the BLE HID device profile.
```

Response:

```
OK
```

Parameter:

- **<init>**:
  - 0: deinit ble hid device profile
  - 1: init ble hid device profile

Notes:

- The BLE 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 BLE HID Keyboard information

Set Command:

```
AT+BLEHIDKB=<Modifier_keys>,<key_1>,<key_2>,<key_3>,<key_4>,<key_5>,<key_6>
Function: to send keyboard information.
```

Response:

```
OK
```

Parameter:

- **<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

Example:

```
AT+BLEHIDKB=0,4,0,0,0,0,0   // input a
```

3.4.42 [ESP32 Only]AT+BLEHIDMUS—Send BLE HID mouse information

Set Command:

```
AT+BLEHIDMUS=<buttons>,<X_displacement>,<Y_displacement>,<wheel>
Function: to send mouse information.
```

Response:

```
OK
```

Parameter:

- **<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 BLE HID consumer information

Set Command:

```
AT+BLEHIDCONSUMER=<consumer_usage_id>
Function: to send consumer information.
```

Response:

```
OK
```

Parameter:

- **<consumer_usage_id>**: consumer id, such as power, reset, help, volume and so on.

Example:

```
AT+BLEHIDCONSUMER=233   // volume up
```

3.4.44 [ESP32 Only] AT+BLUFI—Start or Stop BLUFI

Query Command:

```
AT+BLUFI?
Function: to check the status of 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:

```
AT+BLUFI=<option>
Function: start or stop blufi.
```

Response:

```
OK
```

Parameter:

- **<option>**:
  - 0: stop blufi
  - 1: start blufi

Example:

```
AT+BLUFI=1
```

[ESP32 Only] *AT+BLUFINAME* : Set BLUFI device name

3.4.45 [ESP32 Only] AT+BLUFINAME—Set BLUFI device name

Query Command:

```
AT+BLUFINAME?
Function: to query the BLUFI name.
```

Response:

```
+BLUFINAME:<device_name>
OK
```

Set Command:

```
AT+BLUFINAME=<device_name>
Function: set the BLUFI 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 max length of BLUFI name is 29 bytes.

Example:

```
AT+BLUFINAME="BLUFI_DEV"
AT+BLUFINAME?
```

3.5 [ESP32 Only] BT AT Commands

Download BlueTooth Spec (ESP32 supports Core Version 4.2)

- [ESP32 Only] *AT+BTINIT* : Classic Bluetooth initialization
- [ESP32 Only] *AT+BTNAME* : Sets BT device's name
- [ESP32 Only] *AT+BTSCANMODE* : Sets BT SCAN mode
- [ESP32 Only] *AT+BTSTARTDISC* : Start BT discovery
- [ESP32 Only] *AT+BTSPPINIT* : Classic Bluetooth SPP profile initialization
- [ESP32 Only] *AT+BTSPPCONN* : Establishes SPP connection
- [ESP32 Only] *AT+BTSPPDISCONN* : Ends SPP connection
- [ESP32 Only] *AT+BTSPPSTART* : Start Classic Bluetooth SPP profile
- [ESP32 Only] *AT+BTSPPSEND* : Sends data to remote bt spp device
- [ESP32 Only] *AT+BTA2DPINIT* : Classic Bluetooth A2DP profile initialization
- [ESP32 Only] *AT+BTA2DPCONN* : Establishes A2DP connection
- [ESP32 Only] *AT+BTA2DPDISCONN* : Ends A2DP connection
- [ESP32 Only] *AT+BTA2DPSRC* : Set or query the audio file URL
- [ESP32 Only] *AT+BTA2DPCTRL* : control the audio play
- [ESP32 Only] *AT+BTSECPARAM* :Set and query 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 BT encryption device list
- [ESP32 Only] *AT+BTENCCLEAR* : Clear BT encryption device list
- [ESP32 Only] *AT+BTCOD* : Set class of device

3.5.1 [ESP32 Only] AT+BTINIT—Classic Bluetooth initialization

Query Command:

```
AT+BTINIT?
Function: to check the initialization status of classic bluetooth.
```

Response:

If classic bluetooth is not initialized, it will return:

```
+BTINIT:0
OK
```

If classic bluetooth is initialized, it will return:

```
+BTINIT:1
OK
```

Set Command:

```
AT+BTINIT=<init>
Function: to init or deinit classic bluetooth.
```

Response:

```
OK
```

Parameter:

- **<init>**:
  - 0: deinit classic bluetooth
  - 1: init classic bluetooth

Example:

```
AT+BTINIT=1
```

3.5.2 [ESP32 Only] AT+BTNAME—Sets BT device's name

Query Command:

```
AT+BTNAME?
Function: to get the classic bluetooth device name.
```

Response:

```
+BTNAME:<device_name>
OK
```

Set Command:

```
AT+BTNAME=<device_name>
Function: to set the classic bluetooth device name, The maximum length is 32.
```

Response:

```
OK
```

Parameter:

- **<device_name>**: the classic bluetooth device name

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—Sets BT SCAN mode

Set Command:

```
AT+BTSCANMODE=<scan_mode>
Function: to set the scan mode of classic bluetooth.
```

Response:

```
OK
```

Parameters:

- **<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 BT discovery

Set Command:

```
AT+BTSTARTDISC=<inq_mode>,<inq_len>,<inq_num_rsps>
Function: to set the scan mode of classic bluetooth.
```

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, ranging from 0x01 to 0x30
- **<inq_num_rsps>**: number of inquiry responses that can be received, value 0 indicates an unlimited number of responses
- **<bt_addr>**: 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: Miscellaneous
  - 0x5: Peripheral(mouse, joystick, keyboard)
  - 0x6: Imaging(printer, scanner, camera, display)
  - 0x7: Wearable
  - 0x8: Toy
  - 0x9: Health
  - 0x1F: Uncategorized: device not specified
- **<minor_dev_class>**
  - please refer to this web
- **<major_srv_class>**:
  - 0x0: None indicates an invalid value
  - 0x1: Limited Discoverable Mode
  - 0x8: Positioning (Location identification)
  - 0x10: Networking, e.g. LAN, Ad hoc
  - 0x20: Rendering, e.g. Printing, Speakers
  - 0x40: Capturing, e.g. Scanner, Microphone
  - 0x80: Object Transfer, e.g. v-Inbox, v-Folder
  - 0x100: Audio, e.g. Speaker, Microphone, Headerset service
  - 0x200: Telephony, e.g. Cordless telephony, Modem, Headset service
  - 0x400: Information, e.g., 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:

```
AT+BTSPPINIT?
Function: to check the initialization status of classic bluetooth SPP profile.
```

Response:

If classic bluetooth SPP profile is not initialized, it will return:

```
+BTSPPINIT:0
OK
```

If classic bluetooth SPP profile is initialized, it will return:

```
+BTSPPINIT:1
OK
```

Set Command:

```
AT+BTSPPINIT=<init>
Function: to init or deinit classic bluetooth SPP profile.
```

Response:

```
OK
```

Parameter:

- **<init>**:
  - 0: deinit classic bluetooth SPP profile
  - 1: init classic bluetooth SPP profile, the role is master
  - 2: init classic bluetooth SPP profile, the role is slave

Example:

```
AT+BTSPPINIT=1    //master
AT+BTSPPINIT=2    //slave
```

3.5.6 [ESP32 Only] AT+BTSPPCONN—Establishes SPP connection

Query Command:

```
AT+BTSPPCONN?
Function: to query classic bluetooth SPP connection.
```

Response:

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

If the connection has not been established, there will be return `+BTSPPCONN:-1`. Set Command:

```
AT+BTSPPCONN=<conn_index>,<sec_mode>,<remote_address>
Function: to establish the classic bluetooth SPP connection.
```

Response:

```
OK
```

It will prompt the following message, if the connection is established successfully:

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

It will prompt the following message, if NOT:

```
+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)
  - 0x0012 : Authentication required.
  - 0x0024 : Encryption required.
  - 0x0040 : Mode 4 level 4 service, i.e. incoming/outgoing MITM and P-256 encryption
  - 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—Ends SPP connection

Execute Command:

```
AT+BTSPPDISCONN=<conn_index>
Function: to end the classic bluetooth SPP connection.
```

Response:

```
OK
```

If the command is successful, it will prompt:

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

Parameter:

- **<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—Sends data to remote classic bluetooth spp device

Execute Command:

```
AT+BTSPPSEND
Function: Enter BT SPP mode.
```

Response:

```
>
```

Execute Command:

```
AT+BTSPPSEND=<conn_index>,<data_len>
Function: send data to the remote classic bluetooth SPP device.
```

Response:

```
OK
```

Parameter:

- **<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 was ready to send.

Notes:

- The wrap return is **>** after this command is executed. Then, ESP32 enters UART-BT passthrough mode. When a single packet containing **+++** is received, ESP32 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 the classic bluetooth SPP profile.

Execute Command:

```
AT+BTSPPSTART
Function: start the classic bluetooth SPP profile.
```

Response:

```
OK
```

Example:

```
AT+BTSPPSTART
```

3.5.10 [ESP32 Only] AT+BTA2DPINIT—Classic Bluetooth A2DP profile initialization

Query Command:

```
AT+BTA2DPINIT?
Function: to check the initialization status of classic bluetooth A2DP profile.
```

Response:

If classic bluetooth A2DP profile is not initialized, it will return

```
+BTA2DPINIT:0
OK
```

If classic bluetooth A2DP profile is initialized, it will return

```
+BTA2DPINIT:1
OK
```

Set Command:

```
AT+BTA2DPINIT=<role>,<init_val>
Function: to init or deinit classic bluetooth A2DP profile.
```

Response:

```
OK
```

Parameter:

- **<role>**:
  - 0: source
  - 1: sink
- **<init_val>**:
  - 0: deinit classic bluetooth A2DP profile
  - 1: init classic bluetooth A2DP profile

Example:

```
AT+BTA2DPINIT=0,1
```

3.5.11 [ESP32 Only] AT+BTA2DPCONN—Establishes A2DP connection

Query Command:

```
AT+BTA2DPCONN?
Function: to query classic bluetooth A2DP connection.
```

Response:

```
+BTA2DPCONN:<conn_index>,<remote_address>
OK
```

If the connection has not been established, there will NOT be <conn_index> and <remote_address> Set Command:

```
AT+BTA2DPCONN=<conn_index>,<remote_address>
Function: to establish the classic bluetooth A2DP connectionn.
```

Response:

```
OK
```

It will prompt the message below, if the connection is established successfully:

```
+BTA2DPCONN:<conn_index>,<remote_address>
```

It will prompt the message below, if NOT:

```
+BTA2DPCONN:<conn_index>,fail
```

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—Ends A2DP connection

Execute Command:

```
AT+BTA2DPDISCONN=<conn_index>
Function: to end the classic bluetooth A2DP connection.
```

Response:

```
OK
```

If the command is successful, it will prompt +BTA2DPDISCONN:<conn_index>,<remote_address> Parameter:

- **<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—Set or query the audio file URL

Execute Command:

```
AT+BTA2DPSRC=<conn_index>,<url>
Function: Set the audio file URL.
```

Response:

```
OK
```

Query Command:

```
AT+BTA2DPSRC?
Function: to query the audio file URL.
```

Response:

```
+BTA2DPSRC:<url>,<type>
OK
```

Parameter:

- **<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="https://dl.espressif.com/dl/audio/ff-16b-2c-44100hz.mp3"
AT+BTA2DPSRC="flash://spiffs/zhifubao.mp3"
```

3.5.14 [ESP32 Only] AT+BTA2DPCTRL—control the audio play

Execute Command:

```
AT+BTA2DPCTRL=<conn_index>,<ctrl>
Function: control the audio play
```

Response:

```
OK
```

Parameter:

- **<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—Set and query the Classic Bluetooth security parameters

Query Command:

```
AT+BTSECPARAM?
Function: to query classic bluetooth security parameters.
```

Response:

```
+BTSECPARAM:<io_cap>,<pin_type>,<pin_code>
OK
```

Set Command:

```
AT+BTSECPARAM=<io_cap>,<pin_type>,<pin_code>
Function: set the Classic Bluetooth security parameters.
```

Response:

```
OK
```

Parameters:

- **<io_cap>**: io 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 (upto 16 bytes).

Notes:

- If pin_type is variable, pin_code will be ignored,

Example:

```
AT+BTSECPARAM=3,1,"9527"
```

3.5.16 [ESP32 Only] AT+BTKEYREPLY—Input Simple Pair Key

Execute Command:

```
AT+BTKEYREPLY=<conn_index>,<Key>
Function: Input the Simple Pair Key.
```

Response:

```
OK
```

Parameter:

- **<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:

```
AT+BTPINREPLY=<conn_index>,<Pin>
Function: Input the Legacy Pair PIN Code.
```

Response:

```
OK
```

Parameter:

- **<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:

```
AT+BTSECCFM=<conn_index>,<accept>
Function: Reply the confirm value to the peer device in the legacy connection stage.
```

Response:

```
OK
```

Parameter:

- **<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 BT encryption device list

Query Command:

```
AT+BTENCDEV?
Function: to get the bonded devices.
```

Response:

```
+BTENCDEV:<enc_dev_index>,<mac_address>
OK
```

Parameters:

- **<enc_dev_index>**: index of the bonded devices.
- **<mac_address>**: Mac address.

Example:

```
AT+BTENCDEV?
```

3.5.20 [ESP32 Only] AT+BTENCCLEAR—Clear BT encryption device list

Set Command:

```
AT+BTENCCLEAR=<enc_dev_index>
Function: remove a device from the security database list with a specific index.
```

Response:

```
OK
```

Execute Command:

```
AT+BLEENCCLEAR
Function: remove all devices from the security database.
```

Response:

```
OK
```

Parameters:

- **<enc_dev_index>**: index of the bonded devices.

Example:

```
AT+BTENCCLEAR
```

3.5.21 [ESP32 Only] AT+BTCOD—Set class of device

Set Command:

```
AT+BTCOD=<major>,<minor>,<service>
Function: set the BT class of device.
```

Response:

```
OK
```

Parameters:

- **<major>**: major class.
- **<minor>**: minor class.
- **<service>**: service class.

Example:

```
AT+BTCOD=6,32,32   //the printer
```

3.6 MQTT AT Commands

- *AT+MQTTUSERCFG* : Set MQTT User Config
- *AT+MQTTCLIENTID* : Set MQTT Client ID
- *AT+MQTTUSERNAME* : Set MQTT Username
- *AT+MQTTPASSWORD* : Set MQTT Password
- *AT+MQTTCONNCFG* : Set configuration of MQTT Connection
- *AT+MQTTCONN* : Connect to MQTT Broker
- *AT+MQTTPUB* : Publish MQTT Data in string
- *AT+MQTTPUBRAW* : Publish MQTT message in binary
- *AT+MQTTSUB* : Subscribe to MQTT Topic
- *AT+MQTTUNSUB* : Unsubscribe from MQTT Topic
- *AT+MQTTCLEAN* : Close the MQTT Connection
- *MQTT Error Codes*
- *MQTT Notes*

3.6.1 AT+MQTTUSERCFG - Set MQTT User Config

Set Command:

```
AT+MQTTUSERCFG=<LinkID>,<scheme>,<"client_id">,<"username">,<"password">,<cert_key_ID>
↪,<CA_ID>,<"path">
```

Function:

```
Set MQTT User Config
```

Response:

```
OK
```

Query Command:

```
AT+MQTTUSERCFG?
```

Function:

```
Get the MQTT user configuration.
```

Response:

```
+MQTTUSERCFG:<LinkID>,<scheme>,<"client_id">,<"username">,<"password">,<cert_key_ID>,
↪<CA_ID>,<"path">
OK
```

Parameters:

- **<LinkID>**: only supports link ID 0 for now
- **<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, max length 256Bytes
- **<username>**: the user name to login to the MQTT broker, max length 64Bytes
- **<password>**: the password to login to the MQTT broker, max length 64Bytes
- **<cert_key_ID>**: certificate ID, only supports one certificate of ID 0 for now
- **<CA_ID>**: CA ID, only supports one CA of ID 0 for now
- **<path>**: path of the resource, max length 32Bytes

Note:

- The total length of the entire AT command should be less than 256Bytes.

3.6.2 AT+MQTTCLIENTID - Set MQTT Client ID

Set Command:

```
AT+MQTTCLIENTID=<LinkID><"client_id">
```

Function:

```
Set MQTT Client ID, will cover the parameter client_id in AT+MQTTUSERCFG
User can set a long client id by AT+MQTTCLIENTID.
```

Response:

```
OK
```

Query Command:

```
AT+MQTTCLIENTID?
```

Function:

```
Get the MQTT client ID.
```

Response:

```
+MQTTCLIENTID:<LinkID>,<"client_id">
OK
```

Parameters:

- **<LinkID>**: only supports link ID 0 for now
- **<client_id>**: MQTT client ID, max length 256Bytes

Note:

- The total length of the entire AT command should be less than 256Bytes.
- AT+MQTTCLIENTID command only could be set after AT+MQTTUSERCFG command

3.6.3 AT+MQTTUSERNAME - Set MQTT Username

Set Command:

```
AT+MQTTUSERNAME=<LinkID><"username">
```

Function:

```
Set MQTT Username, will cover the parameter username in AT+MQTTUSERCFG
User can set a long username by AT+MQTTUSERNAME.
```

Response:

```
OK
```

Query Command:

```
AT+MQTTUSERNAME?
```

Function:

```
Get the MQTT client username.
```

Response:

```
+MQTTUSERNAME:<LinkID>,<"username">
OK
```

Parameters:

- **<LinkID>**: only supports link ID 0 for now
- **<username>**: the user name to login to the MQTT broker, max length 256Bytes

Note:

- The total length of the entire AT command should be less than 256Bytes.
- AT+MQTTUSERNAME command only could be set after AT+MQTTUSERCFG command

3.6.4 AT+MQTTPASSWORD - Set MQTT Password

Set Command:

```
AT+MQTTPASSWORD=<LinkID><"password">
```

Function:

```
Set MQTT Password, will cover the parameter password in AT+MQTTUSERCFG
User can set a long password by AT+MQTTPASSWORD.
```

Response:

```
OK
```

Query Command:

```
AT+MQTTPASSWORD?
```

Function:

```
Get the MQTT client password.
```

Response:

```
+MQTTPASSWORD:<LinkID>,<"password">
OK
```

Parameters:

- **<LinkID>**: only supports link ID 0 for now

- **<password>**: the password to login to the MQTT broker, max length 256Bytes

Note:

- The total length of the entire AT command should be less than 256Bytes.
- AT+MQTTPASSWORD command only could be set after AT+MQTTUSERCFG command

3.6.5 AT+MQTTCONNCFG - Set configuration of MQTT Connection

Set Command:

```
AT+MQTTCONNCFG=<LinkID>,<keepalive>,<disable_clean_session>,<"lwt_topic">,<"lwt_msg">,
↪<lwt_qos>,<lwt_retain>
```

Function:

```
Set configuration of MQTT Connection
```

Response:

```
OK
```

Query Command:

```
AT+MQTTCONNCFG?
```

Function:

```
Get configuration of MQTT Connection
```

Response:

```
+MQTTCONNCFG:<LinkID>,<keepalive>,<disable_clean_session>,<"lwt_topic">,<"lwt_msg">,
↪<lwt_qos>,<lwt_retain>
OK
```

Parameters:

- **<LinkID>**: only supports link ID 0 for now
- **<keepalive>**: timeout of MQTT ping, range [0, 7200], unit:second. 0 means default, default is 120s.
- **<disable_clean_session>**: set MQTT clean session
  - 0: enable clean session
  - 1: disable clean session
- **<lwt_topic>**: LWT (Last Will and Testament) message topic, max length 64Bytes
- **<lwt_msg>**: LWT message, max length 64Bytes
- **<lwt_qos>**: LWT QoS, can be set to 0, or 1, or 2. Default is 0.
- **<lwt_retain>**: LWT retain, can be set to 0 or 1. Default is 0.

3.6.6 AT+MQTTCONN - Connect to MQTT Broker

Set Command:

```
AT+MQTTCONN=<LinkID>,<"host">,<port>,<reconnect>
```

Function:

```
Connect to a MQTT broker.
```

Response:

```
OK
```

Query Command:

```
AT+MQTTCONN?
```

Function:

```
Get the MQTT broker that the ESP chip connected to.
```

Response:

```
+MQTTCONN:<LinkID>,<state>,<scheme><"host">,<port>,<"path">,<reconnect>
OK
```

Parameters:

- **<LinkID>**: only supports link ID 0 for now
- **<host>**: MQTT broker domain, max length 128Bytes
- **<port>**: MQTT broker port, max is port 65535
- **<path>**: path, max length 32Bytes
- **<reconnect>**:
  - 0: MQTT will not auto-reconnect
  - 1: MQTT will auto-reconnect, it will take more resource
- **<state>**: MQTT states
  - 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 topic
- **<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)

3.6.7 AT+MQTTPUB - Publish MQTT message in string

Set Command:

```
AT+MQTTPUB=<LinkID>,<"topic">,<"data">,<qos>,<retain>
```

Function:

```
Publish MQTT message in string to defined topic. If you need to publish message in␣
↪binary, please use command `AT+MQTTPUBRAW` instead.
```

Response:

```
OK
```

Parameters:

- **<LinkID>**: only supports link ID 0 for now
- **<topic>**: MQTT topic, max length 64Bytes
- **<data>**: MQTT message in string.
- **<qos>**: qos of publish message, can be set to 0, or 1, or 2. Default is 0.
- **<retain>**: retain flag

Note:

- The total length of the entire AT command should be less than 256Bytes.
- This command cannot send data `\0`, if you need to send `\0`, please use command `AT+MQTTPUBRAW` instead.

3.6.8 AT+MQTTPUBRAW - Publish MQTT message in binary

Set Command:

```
AT+MQTTPUBRAW=<LinkID>,<"topic">,<length>,<qos>,<retain>
```

Function:

```
Publish MQTT message in binary to defined topic.
```

Response:

```
OK
>
```

Wrap return > after the Set Command. Begin receiving serial data. The AT firmware will keep waiting until the data length defined by is met, all data received will be considered as the MQTT publish message. When the data is met, the transmission of data starts. And then it will respond as the following message.

```
+MQTTPUB:FAIL
```

Or

```
+MQTTPUB:OK
```

Parameters:

- **<LinkID>**: only supports link ID 0 for now
- **<topic>**: MQTT topic, max length 64Bytes
- **<length>**: length of MQTT message, max length depends on platform.
  - max length on ESP32 is limited by available memory
  - max length on ESP8266 is limited by marco `MQTT_BUFFER_SIZE_BYTE` and available memory

    Default value of `MQTT_BUFFER_SIZE_BYTE` is `512`Users can change the max length limitation by setting `MQTT_BUFFER_SIZE_BYTE` in `make menuconfig`max publish data length plus the MQTT header length(depends on topic name length) is equal to `MQTT_BUFFER_SIZE_BYTE`
- **<qos>**: qos of publish message, can be set to 0, or 1, or 2. Default is 0.
- **<retain>**: retain flag

3.6.9 AT+MQTTSUB - Subscribe to MQTT Topic

Set Command:

```
AT+MQTTSUB=<LinkID>,<"topic">,<qos>
```

Function:

```
Subscribe to defined MQTT topic with defined QoS. It supports subscribing to multiple␣
↪topics.
```

Response:

```
OK
```

When received MQTT message 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`

Query Command:

```
AT+MQTTSUB?
```

Function:

```
Get all MQTT topics that already subscribed.
```

Response:

```
+MQTTSUB:<LinkID>,<state>,<"topic1">,<qos>
+MQTTSUB:<LinkID>,<state>,<"topic2">,<qos>
+MQTTSUB:<LinkID>,<state>,<"topic3">,<qos>
...
OK
```

Parameters:

- **<LinkID>**: only supports link ID 0 for now
- **<state>**: MQTT states
  - 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 topic
- **<topic>**: the topic that subscribed to
- **<qos>**: the QoS that subscribed to

3.6.10 AT+MQTTUNSUB - Unsubscribe from MQTT Topic

Set Command:

```
AT+MQTTUNSUB=<LinkID>,<"topic">
```

Function:

```
 Unsubscribe the client from defined topic. This command can be called multiple times␣
↪to unsubscribe from differrent topics.
```

Response:

```
OK
```

Parameters:

- **<LinkID>**: only supports link ID 0 for now
- **<topic>**: MQTT topic, max length 64Bytes

Note:

- If the topic has not been subscribed, then the AT log will prompt `NO UNSUBSCRIBE`. And the AT command will still respond `OK`.

3.6.11 AT+MQTTCLEAN - Close the MQTT Connection

Set Command:

```
AT+MQTTCLEAN=<LinkID>
```

Function:

```
Close the MQTT connection, and release the resource.
```

Response:

```
OK
```

Parameters:

- **<LinkID>**: only supports link ID 0 for now

3.6.12 MQTT Error Codes

The MQTT Error code will be prompt as `ERR CODE:0x<%08x>`.

```
    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
    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
```

(continues on next page)

(continued from previous page)

```
    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
    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
```

3.6.13 MQTT Notes

- In general, AT MQTT commands will be responded within 10s, except command `AT+MQTTCONN`.For example, if the router fails to access to the internet, the command `AT+MQTTPUB` will respond within 10s. But the command `AT+MQTTCONN` may need more time due to the packet retransmission in bad network environment.
- If the `AT+MQTTCONN` is based on a TLS connection, the timeout of each packet is 10s, then the total timeout will be much longer depending on the handshake packets count.

- When the MQTT connection ends, it will prompt message `+MQTTDISCONNECTED:<LinkID>`
- When the MQTT connection established, it will prompt 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
- *HTTP AT Error Code*

3.7.1 AT+HTTPCLIENT-Send HTTP Client Request

Set 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>** : optional parameter, HTTP URL, The url field can override the host and path parameters if they are null.
- **<host>**: optional parameter, domain name or IP address
- **<path>**: optional parameter, HTTP Path
- **<transport_type>** HTTP Client transport type, default type is 1.
  - 1 : `HTTP_TRANSPORT_OVER_TCP`
  - 2 : `HTTP_TRANSPORT_OVER_SSL`
- **<data>**: optional parameter. When it is a POST request, `<data>` is the user data sent to HTTP server.

- **<http_req_header>**: optional parameter. The number of request headers can be customized by the user.

Note:

- If <url> is omitted, <host> and <path> must be set.

Example:

```
//HEAD Request
AT+HTTPCLIENT=1,0,"http://httpbin.org/get","httpbin.org","/get",1
AT+HTTPCLIENT=1,0,"http://httpbin.org/get",,,0
AT+HTTPCLIENT=1,0,,"httpbin.org","/get",1
//GET Request
AT+HTTPCLIENT=2,0,"http://httpbin.org/get","httpbin.org","/get",1
AT+HTTPCLIENT=2,0,"http://httpbin.org/get",,,0
AT+HTTPCLIENT=2,0,,"httpbin.org","/get",1
//POST Request
AT+HTTPCLIENT=3,0,"http://httpbin.org/post","httpbin.org","/post",1,"field1=value1&
↪field2=value2"
AT+HTTPCLIENT=3,0,"http://httpbin.org/post",,,0,"field1=value1&field2=value
//HTTP offset continue download
HTTPCLIENT=2,0,"http://www.baidu.com/img/bdlogo.gif",,,0,"Range: bytes=100-200"
```

3.7.2 AT+HTTPGETSIZE-Get HTTP Resource Size

Set Command:

```
AT+HTTPGETSIZE=<url>
```

Response:

```
+HTTPGETSIZE:size

OK
```

Parameters: - **<url>** : HTTP URL.

Example:

```
AT+HTTPGETSIZE="http://www.baidu.com/img/bdlogo.gif"
```

3.7.3 HTTP Error Code

- HTTP Client:
- HTTP Server:
- HTTP AT:The error code of command `AT+HTTPCLIENT` will be `0x7000+Standard HTTP Error Code`.For example, if it gets the HTTP error 404 when calling command `AT+HTTPCLIENT`, then the AT will respond error code as `0x7194`, `hex(0x7000+404)=0x7194`.

More details of Standard HTTP/1.1 Error Code are in RFC 2616: https://tools.ietf.org/html/rfc2616

3.8 [ESP32 Only] Ethernet AT Commands

- [ESP32 Only] *AT+CIPETHMAC* : Sets the MAC address of ESP32 Ethernet.

- [ESP32 Only] *AT+CIPETH* : Sets the IP address of ESP32 Ethernet.

3.8.1 [ESP32 Only] AT+CIPETHMAC—Sets the MAC Address of the ESP32 Ethernet

Query Command:

```
AT+CIPETHMAC?
Function: to obtain the MAC address of the ESP32 Ethernet.
```

Response:

```
+CIPETHMAC:<mac>
OK
```

Set Command:

```
AT+CIPETHMAC =<mac>
Function: to set the MAC address of the ESP32 Ethernet.
```

Response:

```
OK
```

Parameters:

- **<mac>**: string parameter, MAC address of the ESP8266 Ethernet.

Notes:

- The configuration changes will be saved in the NVS area if `AT+SYSSTORE=1`.
- The MAC address of ESP32 SoftAP is different from that of the ESP32 Station. Please make sure that you do not set the same MAC address for both of them.
- Bit 0 of the ESP32 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 and cannot be set.

Example:

```
AT+CIPETHMAC ="1a:fe:35:98:d4:7b"
```

3.8.2 [ESP32 Only] AT+CIPETH—Sets the IP Address of the ESP32 Ethernet

Query Command:

```
AT+CIPETH?
Function: to obtain the IP address of the ESP32 Ethernet.
Notice: Only after calling esp_at_eth_cmd_regist can its IP address be queried.
```

Response:

```
+CIPETH:ip:<ip>
+CIPETH:gateway:<gateway>
+CIPETH:netmask:<netmask>
OK
```

Set Command:

```
AT+CIPETH=<ip>[,<gateway>,<netmask>]
Function: to set the IP address of the ESP32 Ethernet.
```

Response:

```
OK
```

Parameters:

- **<ip>**: string parameter, the IP address of the ESP32 Ethernet.
- **[<gateway>]**: gateway.
- **[<netmask>]**: netmask.

Notes:

- The configuration changes will be saved in the NVS area if `AT+SYSSTORE=1`.
- The set command interacts with DHCP-related AT commands (AT+CWDHCP-related commands):
  - If static IP is enabled, DHCP will be disabled;
  - If DHCP is enabled, static IP will be disabled;
  - Whether it is DHCP or static IP that is enabled depends on the last configuration.

Example:

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

3.9 Signaling Test AT Commands

- *AT+FACTPLCP* - Send with long or short PLCP(physical layer convergence procedure)

3.9.1 AT+FACTPLCP-Send with long or short PLCP(physical layer convergence procedure)

Set Command:

```
AT+FACTPLCP=<enable>,<tx_with_long>
```

Response:

```
OK
```

Parameters:

- **<enable>** : Enable or disable manual configuration
  - 0 : Disable manual configuration, it will use the defalut configuration
  - 1 : Enable manual configuration, send PLCP as `tx_with_long`
- **<tx_with_long>** : Send with long PLCP or not
  - 0 : Send with short PLCP

  - 1 : Send with long PLCP

Before checking the command set details, please review some common information on command types, configurations that can be saved in the flash, as well as messages returned after entering commands.

- *AT Command Types*
- *AT Commands with Configuration Saved in the Flash*
- *AT Messages*

3.10 AT Command Types

Generic AT command has four types:

Type	Command Format	Description
Test Command	AT+=?	Queries the Set Commands' internal parameters and their range of values.
Query Command	AT+?	Returns the current value of parameters.
Set Command	AT+=<...>	Sets the value of user-defined parameters in commands, and runs these commands.
Execute Command	AT+	Runs commands with no user-defined parameters.

- Not all AT commands support all four types mentioned above.
- Square brackets [ ] designate parameters that may be omitted; default value of the parameter will be used instead.

  Below are examples of entering command *AT+CWJAP* with some parameters omitted:

```
AT+CWJAP="ssid","password"
AT+CWJAP="ssid","password","11:22:33:44:55:66"
```

- If the parameter which is not the last one is omitted, you can give a `,` to indicate it.

  Example:

```
AT+CWJAP="ssid","password",,1
```

- String values need to be included in double quotation marks, for example: `AT+CWSAP="ESP756290","21030826",1,4`.
- Escape character syntax is needed if a string contains any special characters, such as `,`, `"` or `\`:
  - `\\`: escape backslash itself
  - `\,`: escape comma which is used to separate each parameter
  - `\"`: escape double quotation marks which used to mark string input
  - `\<any>`: escape `<any>` character means that drop backslash symbol and only use `<any>` character

    Example:

```
AT+CWJAP="comma\,backslash\\ssid","1234567890"
AT+MQTTPUB=0,"topic","\"{\"sensor\":012}\"",1,0
```

- The default baud rate of AT command is 115200.
- AT commands are ended with a new-line (CR-LF), so the serial tool should be set into "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*

3.11 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*: for example, `AT+UART_DEF=115200,8,1,0,3`
- *AT+SAVETRANSLINK* : for example, `AT+SAVETRANSLINK=1,"192.168.6.10",1001`
- *AT+CWAUTOCONN*: for example, `AT+CWAUTOCONN=1`

Saving of configuration settings by several other commands can be switched on or off with *AT+SYSSTORE* command. Please see description of *AT+SYSSTORE* for details.

3.12 AT Messages

Messages	Description
ready	The AT firmware is ready.
ERROR	AT command error, or error occurred during execution.
WIFI CONNECTED	ESP station connected to an AP.
WIFI GOT IP	ESP station got IP address.
WIFI DISCONNECT	ESP station disconnected from an AP.
busy p...	Busy processing. The system is in process of handling the previous command, cannot accept the newly input.
<conn_id>,CONNECT	A network connection of which ID is <conn_id> is established.
<conn_id>,CLOSED	A network connection of which ID is <conn_id> ends.
+IPD	Network data received.
+STA_CONNECTED: <sta_mac>	A station connects to the ESP softAP.
+DIST_STA_IP: <sta_mac>,<sta_ip>	ESP softAP distributes an IP address to the station connected.
+STA_DISCONNECTED: <sta_mac>	A station disconnects from the ESP softAP.
+BLECONN	A BLE connection established.
+BLEDISCONN	A BLE connection ends.
+READ	A read operation from BLE connection.
+WRITE	A write operation from BLE connection.
+NOTIFY	A notification from BLE connection.
+INDICATE	An indication from BLE connection.
+BLESECNTFYKEY	BLE SMP key
+BLEAUTHCMPL	BLE 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=xxxxxxxxxx
```

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.

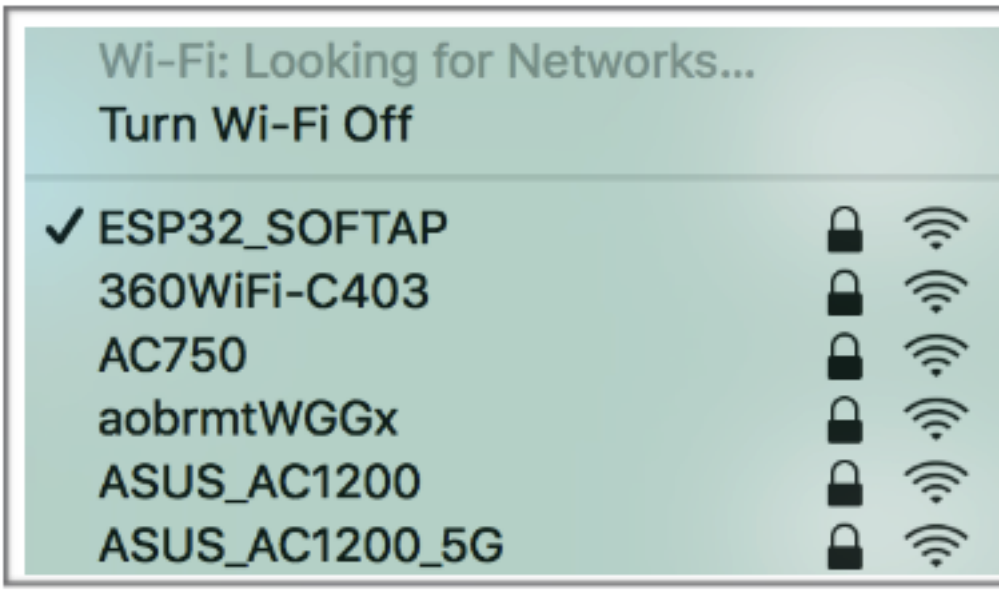
```
AT+CIPMUX=1

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=xxxxxxxxxx
```

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=xxxxxxxxxx
```

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=xxxxxxxxxx
```

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
```

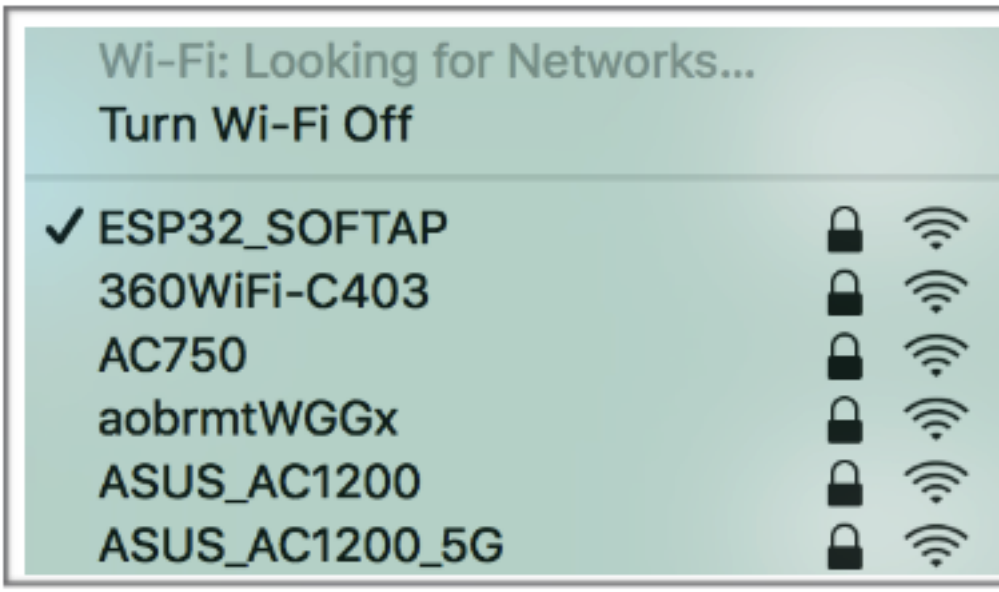
Example 4.2. UDP Transmission in UART-Wi-Fi Passthrough Mode

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:

- 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="0201060A0945737072657373696603030 2A0"

/*  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
```

CHAPTER 5

How to compile and develop your own AT project

[]

5.1 How to clone project and compile it

- *ESP32 platform*
- *ESP32S2 platform*
- *ESP8266 platform*

For specific supported modules, please refer to factory_param_data.csv

5.1.1 ESP32 platform

Hardware Introduction

The WROOM32 Board sends AT commands through UART1 by default.

- GPIO16 is RXD
- GPIO17 is TXD
- GPIO14 is RTS
- GPIO15 is CTS

The debug log will be output through UART0 by default, where TXD0 is GPIO1 and RXD0 is GPIO3, but user can change it in menuconfig if needed:

```
make menuconfig -> Component config -> Common ESP-related -> UART for console output
```

Note: Please pay attention to possible conflicts of the pins

- If choosing `AT through HSPI`, you can get the information of the HSPI pin by `make menuconfig` –> `Component config` –> `AT` –> `AT hspi settings`
- If enabling `AT ethernet support`, you can get the information of the ethernet pin from `ESP32_AT_Ethernet.md`.

Compiling and flashing the project

Suppose you have completed the installation of the compiler environment for ESP-IDF, if not, you should complete it referring to ESP-IDF Getting Started Guide. Then, to compile ESP-AT project properly, please do the following additional steps:

```
step 1: install python>=3.8
step 2: [install pip](https://pip.pypa.io/en/latest/installing/)
step 3: install the following python packages with pip: pip install pyyaml xlrd
```

Compiling the ESP-AT is the same as compiling any other project based on the ESP-IDF:

Note: Please do not set `IDF_PATH` unless you know ESP-AT project in particular. ESP-IDF will automatically be cloned.

1. You can clone the project into an empty directory using command:

   ```
   git clone --recursive https://github.com/espressif/esp-at.git
   ```

2. `rm sdkconfig` to remove the old configuration and `rm -rf esp-idf` to remove the old ESP-IDF if you want to compile other esp platform AT.
3. Set the latest default configuration by `make defconfig`.
4. `make menuconfig` -> `Serial flasher config` to configure the serial port for downloading.
5. `make flash` or `make flash SILENCE=1` to compile the project and download it into the flash, and `make flash SILENCE=1` will remove some logs to reduce firmware size.
   - Or you can call `make` to compile it, and follow the printed instructions to download the bin files into flash by yourself.
   - `make print_flash_cmd` can be used to print the addresses of downloading.
   - More details are in the esp-idf README.
   - If enable BT feature, the firmware size will be much larger, please make sure it does not exceed the ota partition size.
6. `make factory_bin` to combine factory bin, by default, the factory bin is 4MB flash size, DIO flash mode and 40MHz flash speed. If you want use this command, you must first run:: - `make print_flash_cmd | tail -n 1 > build/download.config` to generate `build/download.config`.
7. If the ESP-AT bin fails to boot, and prints "ota data partition invalid", you should run `make erase_flash` to erase the entire flash.

5.1.2 ESP32S2 platform

Hardware Introduction

The WROOM32S2 Board sends AT commands through UART1 by default.

- GPIO18 is RXD
- GPIO17 is TXD
- GPIO19 is RTS
- GPIO20 is CTS

The debug log will output through UART0 by default, which TXD0 is GPIO1 and RXD0 is GPIO3, but user can change it in menuconfig if needed:

`make menuconfig -> Component config -> Common ESP-related -> UART for console output`

Compiling and flashing the project

Suppose you have completed the installation of the compiler environment for ESP-IDF, if not, you should complete it referring to ESP-IDF Getting Started Guide. If required download the compiler toolchain. Then, to compile ESP-AT project properly, please do the following additional steps:

```
step1:python > 3.8.0
step2:[install pip](https://pip.pypa.io/en/latest/installing/)
step3:install the following python packages with pip3: pip3 install pyyaml xlrd
```

Compiling the ESP-AT is the same as compiling any other project based on the ESP-IDF:

Note: Please do not set `IDF_PATH` unless you know ESP-AT project in particular. ESP-IDF will automatically be cloned.

---

1. You can clone the project into an empty directory using command:

   ```
   git clone --recursive https://github.com/espressif/esp-at.git
   ```

2. `rm sdkconfig` to remove the old configuration and `rm -rf esp-idf` to remove the old ESP-IDF if you want to compile other esp platform AT.

3. Set esp module information:

   ```
   export ESP_AT_PROJECT_PLATFORM=PLATFORM_ESP32S2
   export ESP_AT_MODULE_NAME=WROOM
   export ESP_AT_PROJECT_PATH=$(pwd)
   ```

4. `./esp-idf/tools/idf.py -DIDF_TARGET=esp32s2 build` to compile the project and download it into the flash, and `./esp-idf/tools/idf.py -DIDF_TARGET=esp32s2 -DSILENCE=1 build` will remove some logs to reduce firmware size.

Follow the printed instructions to download the bin files into flash by yourself.

5.1.3 ESP8266 platform

Hardware Introduction

The ESP8266 WROOM 02 Board sends AT commands through UART0 by default.

- GPIO13 is RXD
- GPIO15 is TXD
- GPIO1 is RTS
- GPIO3 is CTS

The debug log will output through UART1 by default, which TXD0 is GPIO2, but user can change it in menuconfig if needed:

`make menuconfig` –> `Component config` –> `ESP8266-specific` –> `UART for console output`

Compiling and flashing the project

Suppose you have completed the installation of the compiler environment for ESP-IDF, if not, you should complete it referring to ESP8266 RTOS SDK Getting Started Guide. Then, to compile ESP-AT project properly, please do the following additional steps:

```
step1:install python 2.7 or python 3.x
step2:[install pip](https://pip.pypa.io/en/latest/installing/)
step3:install the following python packages with pip: pip install pyyaml xlrd
```

Compiling the ESP-AT is the same as compiling any other project based on the ESP-IDF:

Note: Please do not set `IDF_PATH` unless you know ESP-AT project in particular. ESP-IDF will automatically be cloned.**

---

1. You can clone the project into an empty directory using command:

   ```
   git clone --recursive https://github.com/espressif/esp-at.git
   ```

2. Change the Makefile from:

   ```
   export ESP_AT_PROJECT_PLATFORM ?= PLATFORM_ESP32
   export ESP_AT_MODULE_NAME ?= WROOM-32
   ```

   to be:

   ```
   export ESP_AT_PROJECT_PLATFORM ?= PLATFORM_ESP8266
   export ESP_AT_MODULE_NAME ?= WROOM-02
   ```

3. `rm sdkconfig` to remove the old configuration and `rm -rf esp-idf` to remove the old ESP-IDF if you want to compile other esp platform AT.
4. Set the latest default configuration by `make defconfig`.
5. `make menuconfig` -> `Serial flasher config` to configure the serial port for downloading.
6. `make flash` or `make flash SILENCE=1` to compile the project and download it into the flash, and `make flash SILENCE=1` will remove some logs to reduce firmware size.
   - Or you can call `make` to compile it, and follow the printed instructions to download the bin files into flash by yourself.
   - `make print_flash_cmd` can be used to print the addresses of downloading.
   - More details are in the ESP-IDF README.

7. `make factory_bin` to combine factory bin, by default, the factory bin is 4MB flash size, DIO flash mode and 40MHz flash speed. If you want use this command, you must fisrt run `make print_flash_cmd | tail -n 1 > build/download.config` to generate `build/download.config`.
8. If the ESP-AT bin fails to boot, and prints "ota data partition invalid", you should run `make erase_flash` to erase the entire flash.

5.2 How to Set `AT Port` Pin

In the project `esp-at`, UART0 and UART1 are used by default. And users can change those UART pins according to their actual hardware design. Since the `esp-at` supports both ESP32 and ESP8266, there are some differences between the configurations.

5.2.1 ESP32 AT

The UART pin of ESP32 can be user-defined to other pins, refer to ESP32 Technical Reference Manual. In the official Espressif ESP32 AT bin, UART0 is the default port to print log, using the following pins:

```
TX ---> GPIO1
RX ---> GPIO3
```

The log pins can be set in `make menuconfig > Component config > Common ESP-related > UART for console output`. UART1 is for sending AT commands and receiving response, but its pins can be changed. The pins of UART1 is in the `factory_param.bin`, they can be changed in the component file customized_partitions/raw_data/factory_param/factory_param_data.csv. The UART1 pins may be different for different ESP modules. More details of `factory_param_data.csv` are in the `How_to_create_factory_parameter_bin.md`.

For example, the configuration of the `ESP32-WROOM-32` is as the following table.

Parameter	Value
platform	PLATFORM_ESP32
module_name	WROOM-32
magic_flag	0xfcfc
version	2
module_id	1
tx_max_power	78
uart_port	1
start_channel	1
channel_num	13
country_code	CN
uart_baudrate	115200
uart_tx_pin	17
uart_rx_pin	16
uart_cts_pin	15
uart_rts_pin	14
tx_control_pin	-1
rx_control_pin	-1

In this case, the pins of `ESP32-WROOM-32` AT port is:

```
TX ---> GPIO17
RX ---> GPIO16
CTS ---> GPIO15
RTS ---> GPIO14
```

For example, if you need to set GPIO1 (TX) and GPIO3 (RX) to be both the log pin and AT port pin, then you can set it as the following steps.

1. Open component file customized_partitions/raw_data/factory_param/factory_param_data.csv.
2. Choose the line of `WROOM-32`, set `uart_port` to be 0, `uart_tx_pin` to be 1 and `uart_rx_pin` to be 3, and then save it.

Parameter	Value
platform	PLATFORM_ESP32
module_name	WROOM-32
magic_flag	0xfcfc
version	2
module_id	1
tx_max_power	78
uart_port	0
start_channel	1
channel_num	13
country_code	CN
uart_baudrate	115200
uart_tx_pin	1
uart_rx_pin	3
uart_cts_pin	-1
uart_rts_pin	-1
tx_control_pin	-1
rx_control_pin	-1

3. Recompile the `esp-at` project, download the new `factory_param.bin` and AT bin into flash.

5.2.2 ESP8266 AT

ESP8266 has two UART ports, UART0 and UART1. UART1 only supports TX pin to print debug log. UART0 has both TX and RX pin, to send AT commands and receive response. Unlike ESP32, UART0 pins of ESP8266 cannot be set to any pins, there are only two choice, `GPIO15 as TX pin, GPIO13 as RX` or `GPIO1 as TXGPIO3 as RX`.

The default setting of ESP8266 AT UART is

- Use UART0 is the AT port to send/receive AT commands/responses. GPIO15 is the UART0 TX, GPIO13 is the UART0 RX.
- Use UART1 to print debug log, GPIO2 is the UART1 TX pin.

For example, if you need to set GPIO1 (TX) and GPIO3 (RX) of ESP-WROOM-02 to be both the log pin and AT port pin, then you can set it as the following steps.

1. `make menuconfig > Component config > ESP8266-specific > UART for console output > Default: UART0`
2. Open component file customized_partitions/raw_data/factory_param/factory_param_data.csv, choose the line of `WROOM-02`, set `uart_tx_pin` to be 1 and `uart_rx_pin` to be 3, and then save it.

Parameter	Value
platform	PLATFORM_ESP8266
module_name	WROOM-02
magic_flag	0xfcfc
...	...
uart_baudrate	115200
uart_tx_pin	1
uart_rx_pin	3
uart_cts_pin	-1
uart_rts_pin	-1
...	...

3. Recompile the `esp-at` project, download the new `factory_param.bin` and AT bin into flash.

5.2.3 ESP32S2 AT

The UART pin of ESP32S2 can be user-defined to other pins, refer to ESP32S2 Technical Reference Manual. In the official Espressif ESP32S2 AT bin, UART0 is the default port to print log, using the following pins:

```
TX ---> GPIO43
RX ---> GPIO44
```

The log pins can be set in `make menuconfig > Component config > Common ESP-related > UART for console output`. UART1 is for sending AT commands and receiving response, but its pins can be changed. The pins of UART1 are configured in the `factory_param.bin`, they can be changed in the component file customized_partitions/raw_data/factory_param/factory_param_data.csv. The UART1 pins may be different for different ESP modules. More details of `factory_param_data.csv` are in the `How_to_create_factory_parameter_bin.md`.

For example, the configuration of the `ESP32S2-WROVER` is as the following table.

Parameter	Value
platform	PLATFORM_ESP32S2
module_name	WROVER
magic_flag	0xfcfc
version	2
module_id	0
tx_max_power	78
uart_port	1
start_channel	1
channel_num	13
country_code	CN
uart_baudrate	115200
uart_tx_pin	17
uart_rx_pin	21
uart_cts_pin	20
uart_rts_pin	19
tx_control_pin	-1
rx_control_pin	-1

In this case, the pins of `ESP32S2-WROVER` AT port is:

```
TX ---> GPIO17
RX ---> GPIO21
CTS ---> GPIO20
RTS ---> GPIO19
```

For example, if you need to set GPIO43 (TX) and GPIO44 (RX) to be both the log pin and AT port pin, then you can set it as the following steps.

1. Open component file customized_partitions/raw_data/factory_param/factory_param_data.csv.
2. Choose the line of `WROVER`, set `uart_port` to be 0, `uart_tx_pin` to be 43 and `uart_rx_pin` to be 44, and then save it.

Parameter	Value
platform	PLATFORM_ESP32S2
module_name	WROVER
magic_flag	0xfcfc
version	2
module_id	0
tx_max_power	78
uart_port	0
start_channel	1
channel_num	13
country_code	CN
uart_baudrate	115200
uart_tx_pin	43
uart_rx_pin	44
uart_cts_pin	-1
uart_rts_pin	-1
tx_control_pin	-1
rx_control_pin	-1

3. Recompile the `esp-at` project, download the new `factory_param.bin` and AT bin into flash.
4. If you don't want to compile the entire project in the third step, you can refer to `How_to_create_factory_parameter_bin.md`.

5.3 How to add user-defined AT commands

AT firmware is based on the Espressif IoT Development Framework (ESP-IDF). Espressif Systems' AT commands are provided in `libat_core.a`, which is included in the *AT BIN firmware*.

Examples of implementing user-defined AT commands are provided in main/interface/uart/at_uart_task.c.

- The total length of an AT command cannot be longer than 256 bytes.
- Only alphabetic characters (A~Z, a~z), numeric characters (0~9), and some other characters (`!`, `%`, `-`, `.`, `/`, `:`, `_`) are supported when naming user-defined AT commands.
- The structure, *`esp_at_cmd_struct`*, is used to define *four types* of a command.

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:

- version:
  - the version of factory param mangement
- module_id
  - the index of development boards, it MUST be unique.
    - 1 - WROOM32
    - 2 - WROVER32
    - 3 - PICO-D4
    - 4 - SOLO
- tx_max_power
  - Wi-Fi maximum tx power
- 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

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
#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

#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
};
```

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

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`

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 `make 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. `make menuconfig` -> `Serial flasher config` to configure the serial port for downloading.
2. `make menuconfig` -> `Component config` -> `AT` -> `AT ethernet support` to enable ethernet.
3. `make menuconfig` -> `Component config` -> `AT` -> `Ethernet PHY` to choose ethernet.
4. 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
```

(continues on next page)

(continued from previous page)

```
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 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.

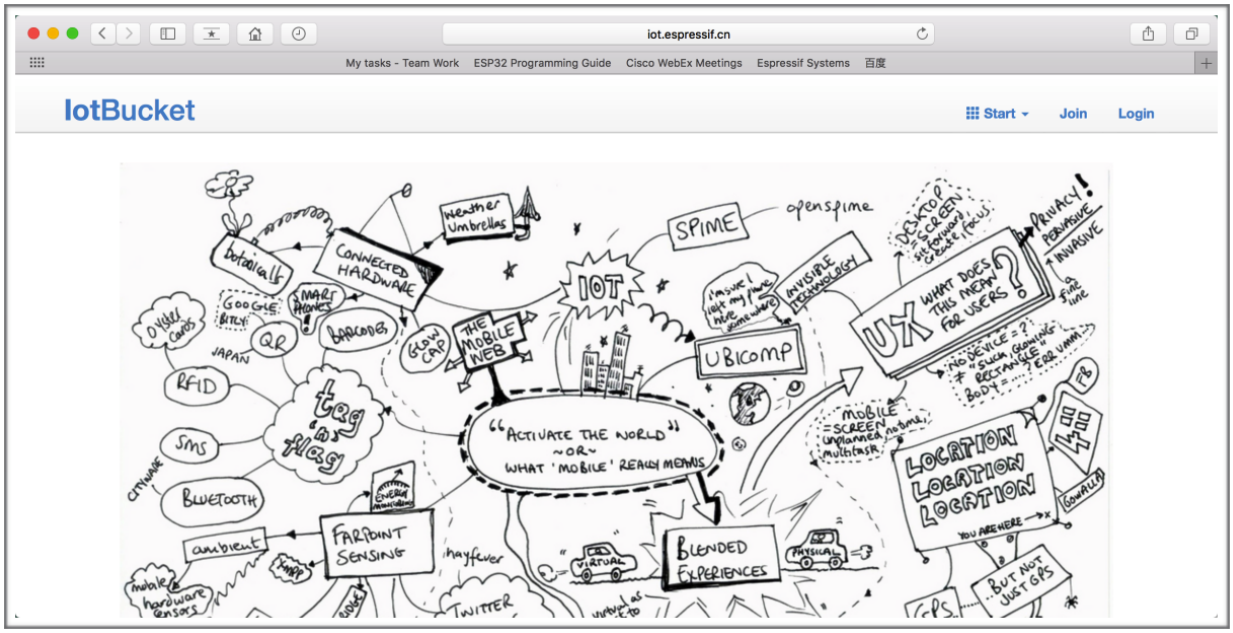


Fig. 1: Open iot.espressif.cn website

2. Click “Join” in the upper right corner of the webpage, and enter your name, email address, and password.

Fig. 2: Join iot.espressif.cn website

3. Click on “Device” in the upper right corner of the webpage, and click on “Create” to create a device.
4. A key is generated when the device is successfully created, as the figure below shows.
5. Use the key to compile your own OTA BIN. The process of configuring the AT OTA token key is as follows:

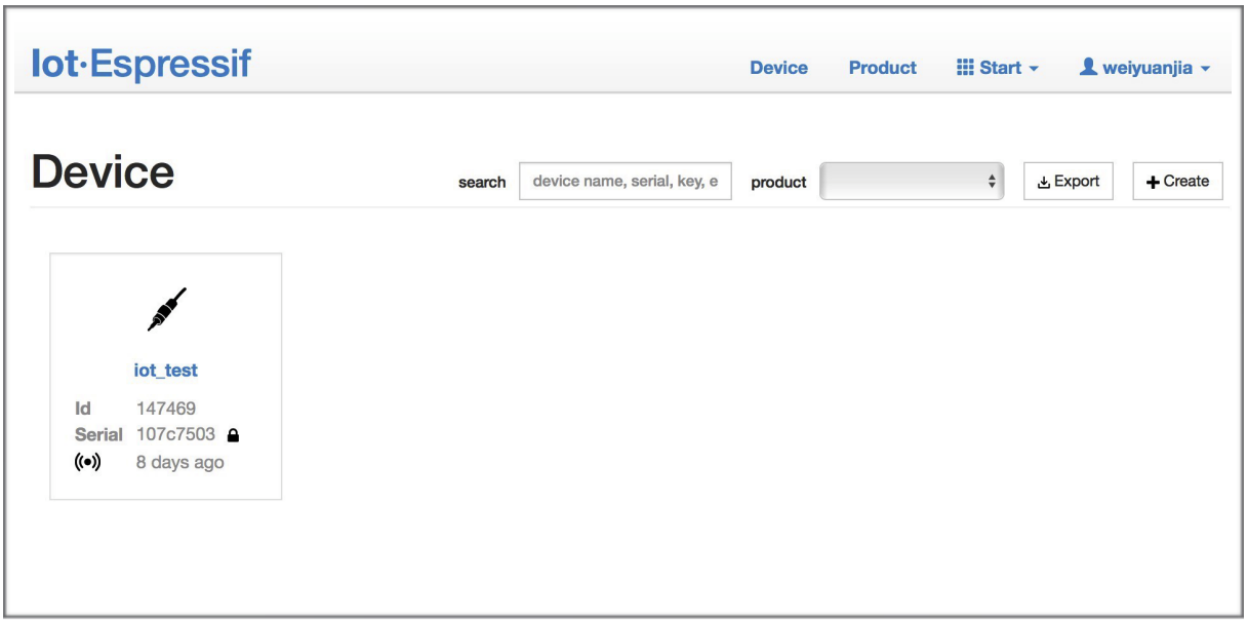


Fig. 3: Click on “Device”

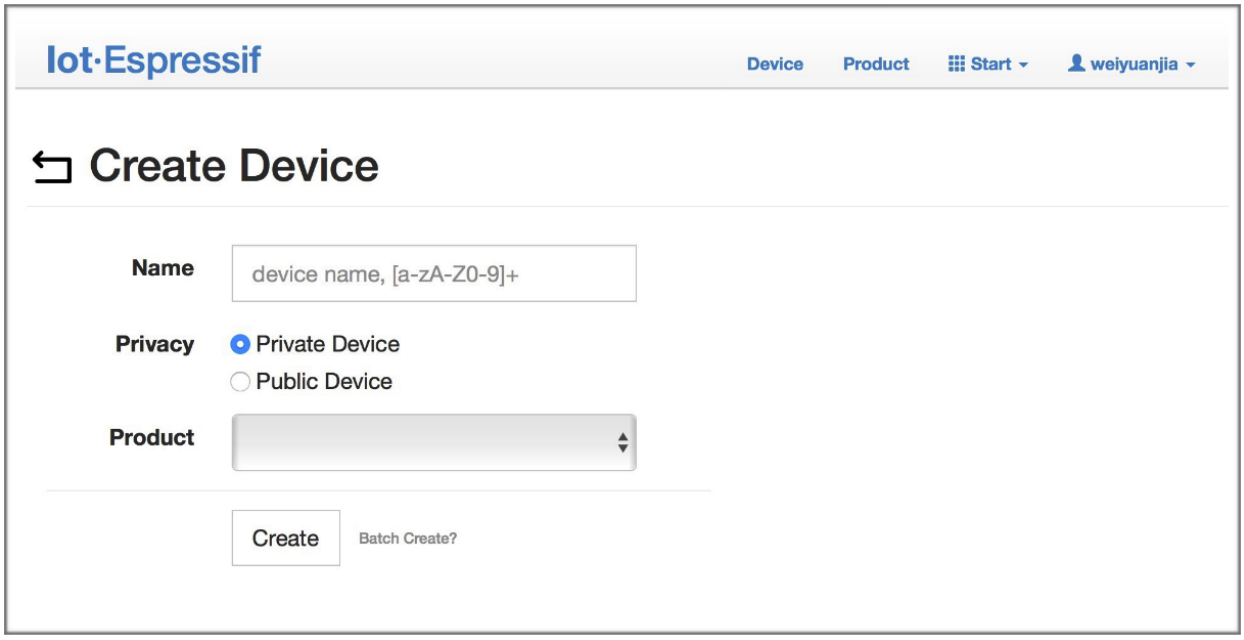


Fig. 4: Click on “Create”

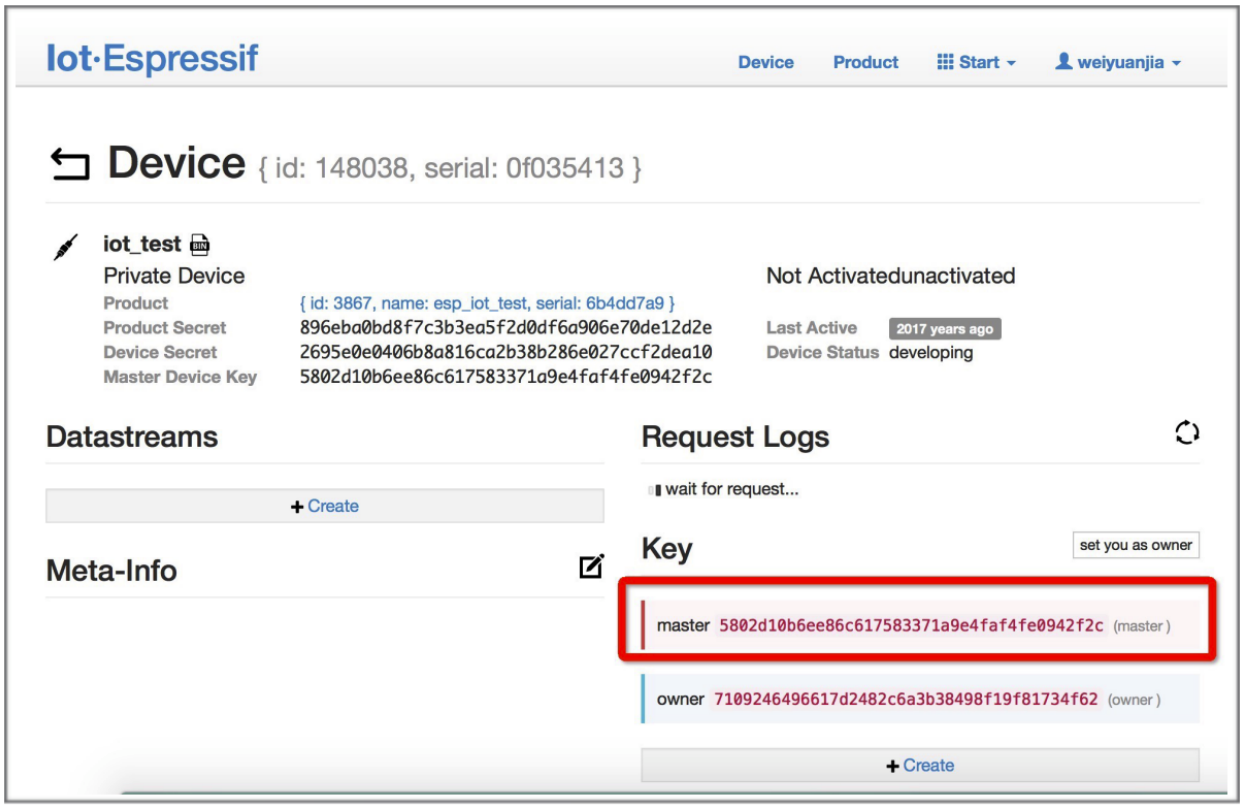


Fig. 5: A key has been generated

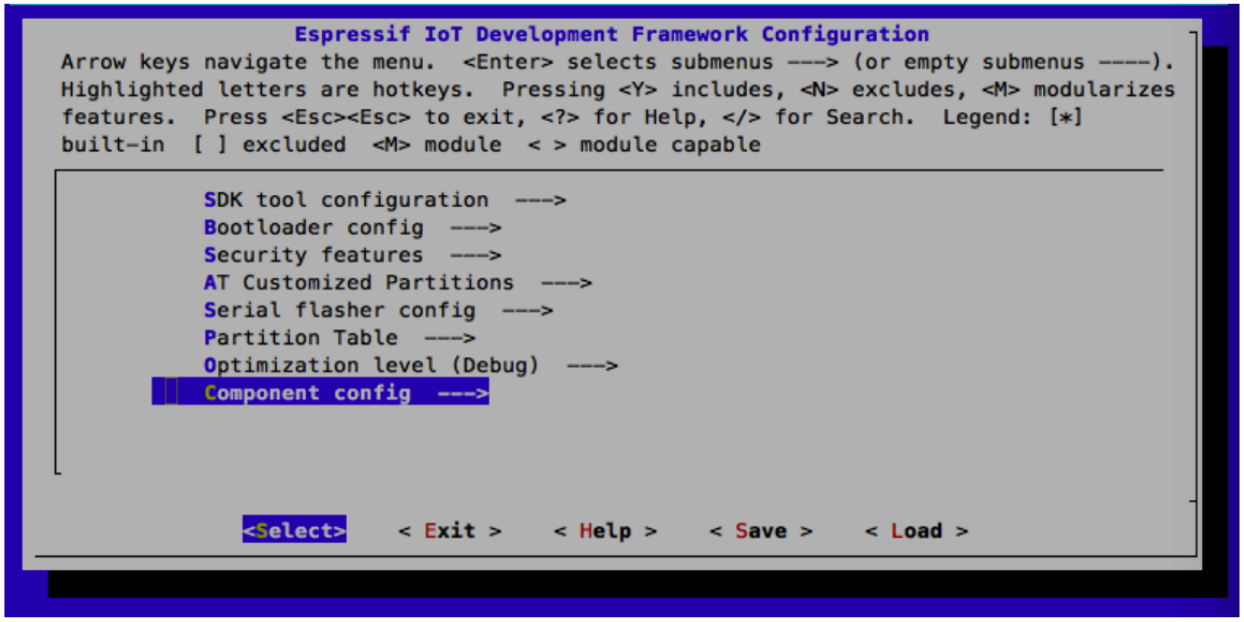


Fig. 6: Configuring the AT OTA token key - Step 1

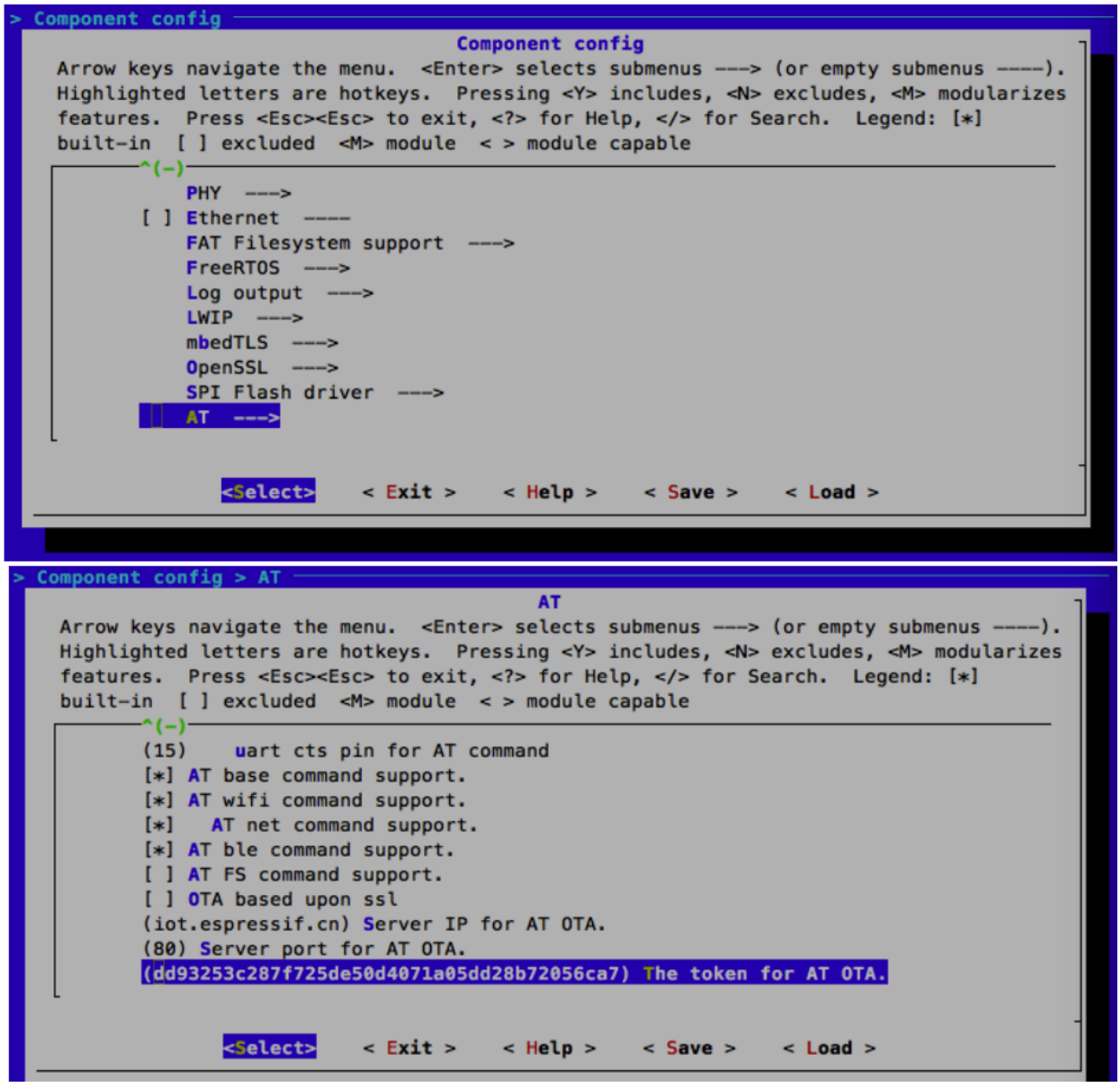


Fig. 7: Configuring the AT OTA token key - Step 2 and 3

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.

5.12 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
```

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.

5.13 AT API Reference

5.13.1 Header File

- at/include/esp_at.h

5.13.2 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.

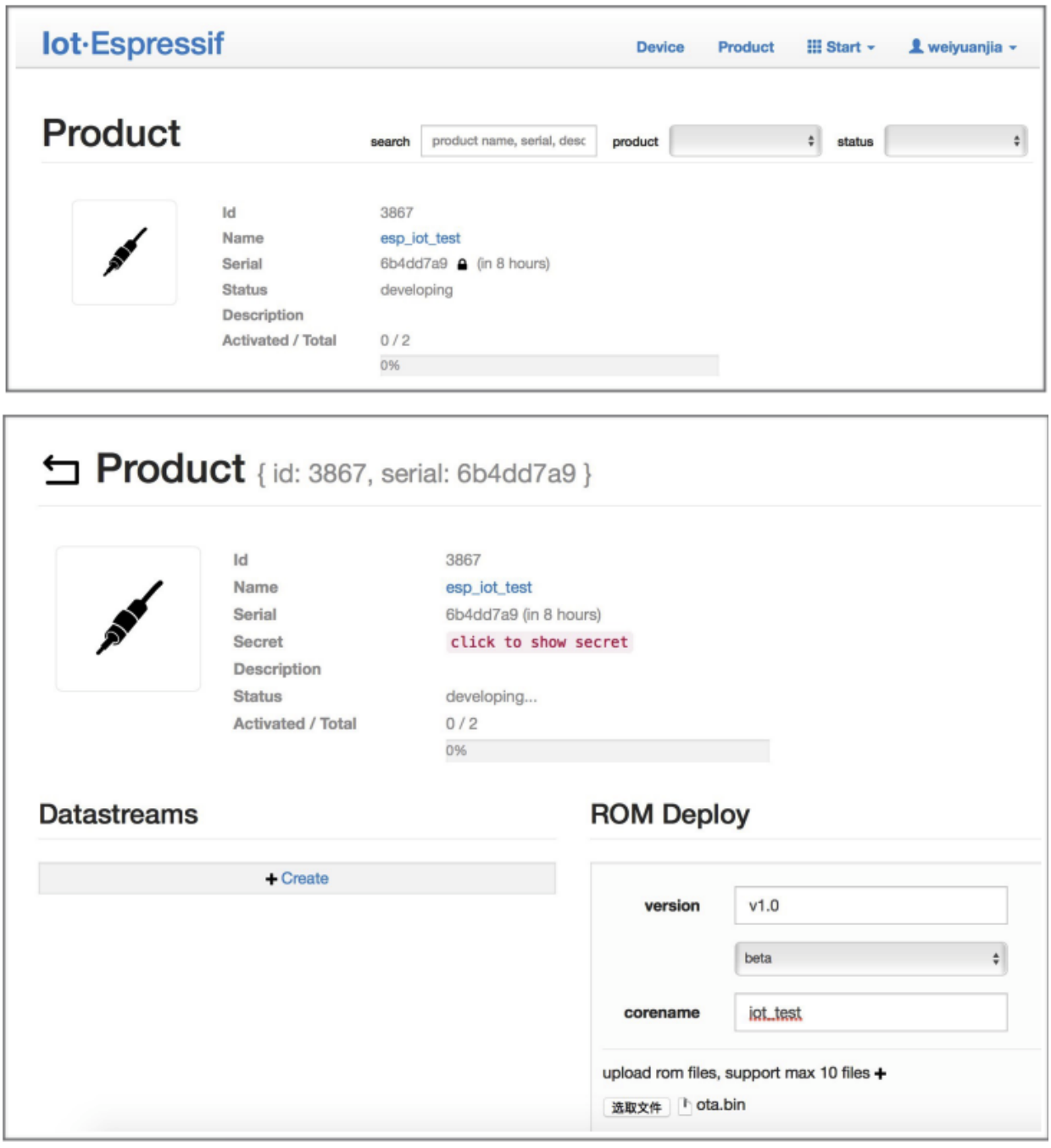


Fig. 8: Enter version and corename

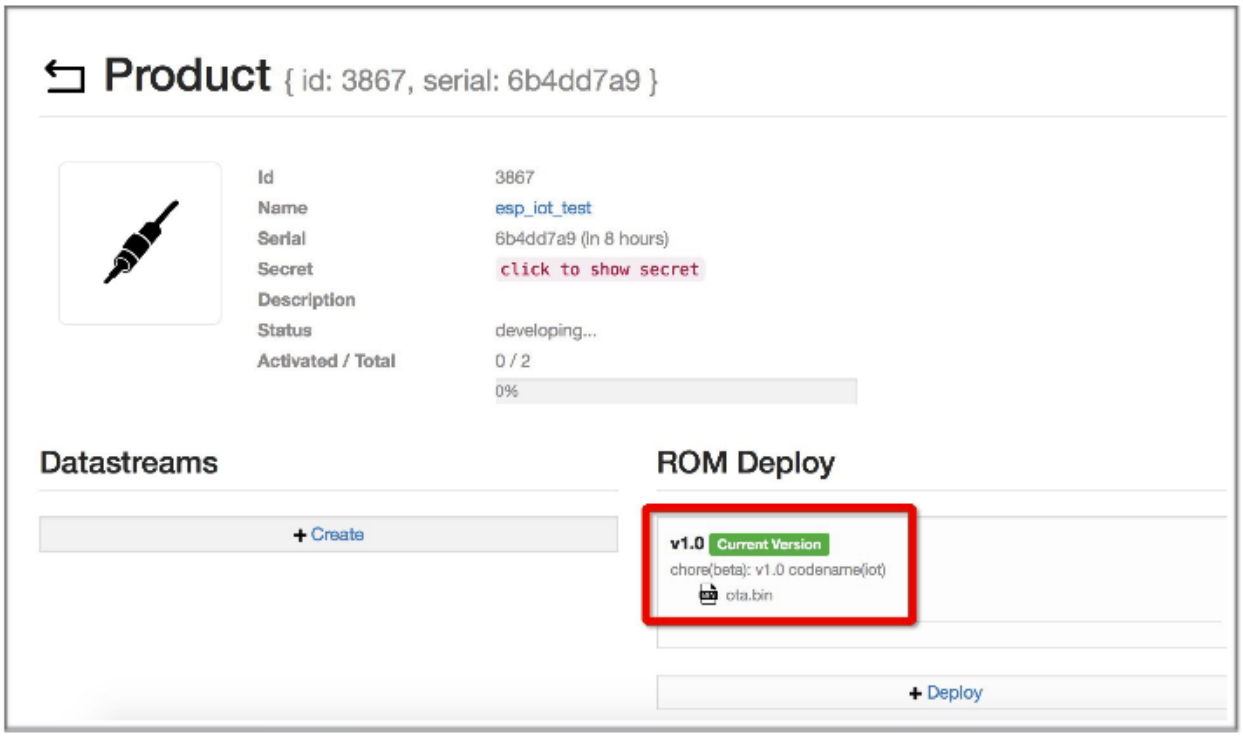


Fig. 9: Save the current version of ota.bin

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_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_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_type_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

5.13.3 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

5.13.4 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

5.13.5 Type Definitions

typedef void (***esp_at_port_specific_callback_t**)(void)
AT specific callback type.

5.13.6 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

- genindex

Index

E