ESP AT Commands Set

1. ATCo

Notes: 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.

2.15 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.

2.16 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
```

2.17 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
```

2.18 AT+SYSLSP—Enters light-sleep mode (Only Support ESP32)

Execute Command:
AT+SYSLSP
Response:

```
OK
```

Example:

```
AT+SYSLSP
```

2.19 AT+SYSLSPCFG—Config the light-sleep wakeup source (Only Support ESP32)

Set Command:

```
AT+SYSLSPCFG=<wakeup source>,<param>[,<wakeup level>]
```

Response:

```
OK
```

Parameters:

- **<wakeup source>**:
  - 0: Wakeup by timer.
  - 1: Wakeup by UART.
  - 2: Wakeup by GPIO.
- **<param>**:
  - 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.
- **<wakeup level>**: only for wakeup source GPIO, 0: Low level, 1: High level.

Example:

```
AT+SYSLSPCFG=0,1000  // Timer wakeup
AT+SYSLSPCFG=1,1     // Uart1 wakeup
AT+SYSLSPCFG=2,12,1  // GPIO12 wakeup, high level
```

3 Wi-Fi AT Commands

3.1 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>
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

Note:

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

Example:

```
AT+CWMODE=3
```

3.2 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>
OK
```

Parameters:

- **<ssid>**: a string parameter showing the SSID of the AP.
- **<bssid>**: the AP's MAC address.
- **<channel>**: channel
- **<rssi>**: signal strength

Set Command:

```
AT+CWJAP=<ssid>,<pwd>[,<bssid>][,<pci_en>][,<reconn>][,<listen_interval>]
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],
- **<error code>**: (for reference only)
  - 1: connection timeout.
  - 2: wrong password.
  - 3: cannot find the target AP.
  - 4: connection failed.
  - others: unknown error occurred.

Note:

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

3.3 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.
```

3.4 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"
```

3.5 AT+CWQAP—Disconnects from the AP

Execute Command:

```
AT+CWQAP
```

Response:

```
OK
```

3.6 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.

Example:

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

3.7 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.

3.8 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.

3.9 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 stored in the NVS area.
- 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.
```

3.10 AT+CWDHCPS—Sets the IP Address Allocated by ESP32 SoftAP DHCP (The configuration is saved in Flash.)

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.
- 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.
```

3.11 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
```

3.12 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.
- 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"
```

3.13 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.
- 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"
```

3.14 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.
- 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"
```

3.15 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.
- 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"
```

3.16 AT+CWSTARTSMART—Starts SmartConfig

Execute Command:

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

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
```

3.17 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
```

3.18 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
```

3.19 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
```

3.20 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
```

Note:

- The configuration changes will be saved in the NVS area.
- This command requires Station mode to be active.
- TLS mode will use client certificate, make sure enabled.

3.21 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"
```

4. TCP/IP-Related AT Commands

4.1 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 ESP32 Station interface.
  - 0: The ESP32 station is inactive.
  - 1: The ESP32 station is idle.
  - 2: The ESP32 Station is connected to an AP and its IP is obtained.
  - 3: The ESP32 Station has created a TCP or UDP transmission.
  - 4: The TCP or UDP transmission of ESP32 Station is disconnected.
  - 5: The ESP32 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>**: ESP32 local port number.
- **<tetype>**:
  - 0: ESP32 runs as a client.
  - 1: ESP32 runs as a server.

4.2 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
```

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

4.3.1 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.

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"
```

4.3.2 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"
```

4.3.3 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:

- ESP32 can only set one SSL connection at most.
- 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.

Example:

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

4.4 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.

4.5 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.

4.6 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.

4.7 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
```

4.8 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.

4.9 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
```

4.10 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
```

4.11 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
```

4.12 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
```

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

4.13.1 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"
```

4.13.2 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
```

4.14 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
```

4.15 AT+CIPSNTPCFG—Sets 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=<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=8,"cn.ntp.org.cn","ntp.sjtu.edu.cn"
```

4.16 AT+CIPSNTPTIME—Queries the SNTP Time

Query Command:

```
AT+CIPSNTPTIME?
```

Response:

```
+CIPSNTPTIME:SNTP time
OK
```

Example:

```
AT+CIPSNTPCFG=8,"cn.ntp.org.cn","ntp.sjtu.edu.cn"
OK
AT+CIPSNTPTIME?
+CIPSNTPTIME:Mon Dec 12 02:33:32 2016
OK
```

4.17 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.

4.18 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
```

4.19 +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.

4.20 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.
  - 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:

- Call 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.

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

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
```

4.22 +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.

4.23 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 5840 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 mode data 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

4.24 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>**: data length that you want to get, max is 2048 bytes per time.
- **<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
```

Notes:

- In a case of disconnection, the buffered Socket data will still be there and can be read by MCU until you send `AT+CIPCLOSE`, or a new connection occupied the previous link_id instead.

4.25 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.

4.26 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"
```

4.27 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.
- 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.

5. [ESP32 Only] BLE-Related AT Commands

5.1 [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
```

5.2 [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
```

5.3 [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 default BLE device name is "BLE_AT".

Example:

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

5.4 [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
```

5.5 [ESP32 Only] AT+BLESCAN—Enables BLE Scanning

Set Command:

```
AT+BLESCAN=<enable>[,<interval>]
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.
- **<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
```

5.6 [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"
```

5.7 [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>**:
  - 0 ADV_TYPE_IND
  - 2 ADV_TYPE_SCAN_IND
  - 3 ADV_TYPE_NONCONN_IND
- **<own_addr_type>** own BLE address type
  - 0 BLE_ADDR_TYPE_PUBLIC
  - 1 BLE_ADDR_TYPE_RANDOM
- **<channel_map>** channel of advertising
  - 1 ADV_CHNL_37
  - 2 ADV_CHNL_38
  - 4 ADV_CHNL_39
  - 7 ADV_CHNL_ALL
- **[<adv_filter_policy>]**(optional parameter) filter policy of advertising
  - 0 ADV_FILTER_ALLOW_SCAN_ANY_CON_ANY
  - 1 ADV_FILTER_ALLOW_SCAN_WLST_CON_ANY
  - 2 ADV_FILTER_ALLOW_SCAN_ANY_CON_WLST
  - 3 ADV_FILTER_ALLOW_SCAN_WLST_CON_WLST
- **[<peer_addr_type>]**(optional parameter) remote BLE address type
  - 0 PUBLIC
  - 1 RANDOM
- **[<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"
```

5.8 [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"`.

Example:

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

5.9 [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.

Example:

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

5.10 [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
```

5.11 [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; only 0 is supported for the single connection right now, but multiple BLE connections will be supported in the future.
- **<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
```

5.12 [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; only 0 is supported for the single connection right now, but multiple BLE connections will be supported in the future.
- **<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
```

5.13 [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; only 0 is supported for the single connection right now, but multiple BLE connections will be supported in the future.
- **<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
```

5.14 [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; only 0 is supported for the single connection right now, but multiple BLE connections will be supported in the future.
- **<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
```

5.15 [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
```

5.16 [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
```

5.17 [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>**:
  - 0 the service has not started
  - 1 the service has already started
- **<srv_uuid>**: service's UUID
- **<srv_type>**: service's type
  - 0 is not a primary service
  - 1 is a primary service

Example:

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

5.18 [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?
```

5.19 [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; 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+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
```

5.20 [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; 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+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
```

5.21 [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:
AT+BLEGATTSSETATTR=1,1,,4
// after > shows, inputs 4 bytes of data, such as "1234"; then, the setting
starts
```

5.22 [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; 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 starting from 1
- **<srv_uuid>**: service's UUID
- **<srv_type>**: service's type
  - 0 is not a primary service
  - 1 is 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
```

5.23 [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
  - 0 is not a primary service
  - 1 is a primary service
- **<included_srv_uuid>**: included service's UUID
- **<included_srv_type>**: included service's type
  - 0 is not a primary service
  - 1 is 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
```

5.24 [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; 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>`
- **<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
```

5.25 [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; 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>`

- **<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
```

5.26 [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:
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>`
- **<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
```

5.27 [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
```

5.28 [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
```

5.29 [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
```

5.30 [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.
- **<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
```

5.31 [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.
- **<accept>**:
  - 0 : reject
  - 1 : accept;

Example:

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

5.32 [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.
- **<key>**: pairing key

Example:

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

5.33 [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.
- **<confirm>**:
  - 0 : NO
  - 1 : Yes

Example:

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

5.34 [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?
```

5.35 [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
```

5.36 [ESP32 Only][AT+BLESETKEY](#BLE-AT)—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
```

5.37 [ESP32 Only][AT+BLEHIDINIT](#BLE-AT)—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
```

5.38 [ESP32 Only][AT+BLEHIDKB](#BLE-AT)—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
```

5.39 [ESP32 Only][AT+BLEHIDMUS](#BLE-AT)—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
```

5.40 [ESP32 Only][AT+BLEHIDCONSUMER](#BLE-AT)—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
```

6. [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+BLEADDR?                                // get server's BLE address

   Response:
   +BLEADDR:24:0a:c4:03:f4:d6
   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:
+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.

7 [ESP32 Only] ETH AT Commands

7.1 [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.
- 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"
```

7.2 [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.
- 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"
```

8. [ESP32 Only] BT-Related AT Commands

8.1 [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
```

8.2 [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 248.
```

Response:

```
OK
```

Parameter:

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

Notes:

- The default classic bluetooth device name is "ESP32_AT".

Example:

```
AT+BTNAME="esp_demo"
```

8.3 [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

Example:

```
AT+BTSCANMODE=2   // both discoverable and connectable
```

8.4 [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
```

8.5 [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
```

8.6 [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"
```

8.7 [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
```

8.8 [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
```

8.9 [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
```

8.10 [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
```

8.11 [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"
```

8.12 [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
```

8.13 [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"
```

8.14 [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
```

8.15 [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"
```

8.16 [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
```

8.17 [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"
```

8.18 [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
```

8.19 [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?
```

8.20 [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
```

9.[ESP32 Only] MQTT AT Commands List

9.1 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
```

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.

9.2 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
```

Parameters:

- **<LinkID>**: only supports link ID 0 for now
- **<keepalive>**: timeout of MQTT ping, range [60, 7200], unit:second. 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.

9.3 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)

9.4 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.

9.5 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 is 1024 by default. Users can change the max length limitation by setting `MQTT_BUFFER_SIZE_BYTE` in `make menuconfig`
- **<qos>**: qos of publish message, can be set to 0, or 1, or 2. Default is 0.
- **<retain>**: retain flag

9.6 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

9.7 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`.

9.8 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

9.9 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
    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
```

9.10 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>`

9.11 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
```

9.12 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
```

9.13 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
```

10. HTTP AT Command

10.1 AT+HTTPCLIENT-Send HTTP Client Request

Set Command:

```
AT+HTTPCLIENT=<opt>,[<url>],[<host>],[<path>],<transport_type>,[<data>]
```

Response:

```
OK
```

Parameters:

- **<opt>** : method of HTTP client request
  - 1 : HEAD
  - 2 : GET
  - 3 : POST
  - 4 : PUT
  - 5 : DELETE
- **<content-type>** : date 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 is 0.
  - 0 : `HTTP_TRANSPORT_UNKNOWN`
  - 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.

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=valu
e1&field2=value2"
AT+HTTPCLIENT=3,0,"http://httpbin.org/post",,,0,"field1=value1&field2=value
```

10.2 HTTP Error Code

- HTTP Client:

HTTP Client Error Code	Description
0x7190	Bad Request
0x7191	Unauthorized
0x7192	Payment Required
0x7193	Forbidden
0x7194	Not Found
0x7195	Method Not Allowed
0x7196	Not Acceptable
0x7197	Proxy Authentication Required
0x7198	Request Timeout
0x7199	Conflict
0x719a	Gone
0x719b	Length Required
0x719c	Precondition Failed
0x719d	Request Entity Too Large
0x719e	Request-URI Too Long
0x719f	Unsupported Media Type
0x71a0	Requested Range Not Satisfiable
0x71a1	Eectation Failed

- HTTP Server:

HTTP Server Error Code	Description
0x71f4	Internal Server Error
0x71f5	Not Implemented
0x71f6	Bad Gateway
0x71f7	Service Unavailable
0x71f8	Gateway Timeout
0x71f9	HTTP Version Not Supported

- HTTP AT:
  The error code of command `AT+HTTP` will be `0x7000+Standard HTTP Error Code`.
  For example, if it gets the HTTP error 404 when calling command `AT+HTTP`, 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

Appendix. How to generate an ESP8266 AT firmware

1. Download the master branch of https://github.com/espressif/esp-at
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. Compile the esp-at project to get the ESP8266 AT firmware.
4. More details are in the esp-at/docs/How_to_Add_New_Platforom.