



ESP32

ESP-AT User Guide



Release latest
Espressif Systems
Mar 18, 2026



Table of contents

Table of contents	i
1 Get Started	3
1.1 What is ESP-AT	3
1.2 Technology Selection	4
1.2.1 Hardware Selection	4
1.2.2 AT Software Solution Selection	4
1.2.3 Initial Project Preparation	6
1.3 Hardware Connection	6
1.3.1 What You Need	7
1.3.2 ESP32 Series	7
1.4 Downloading Guide	12
1.4.1 Download AT Firmware	13
1.4.2 Flash AT Firmware into Your Device	13
1.4.3 Check Whether AT Works	16
2 AT Binary Lists	19
2.1 Released Firmware	19
2.1.1 Introduction	19
2.1.2 ESP32-WROOM-32 Series	20
2.1.3 ESP32-MINI-1 Series	20
2.1.4 ESP32-WROVER-32 Series	20
2.1.5 ESP32-PICO Series	20
2.1.6 ESP32-SOLO Series	21
2.1.7 Subscribe to AT Releases	21
2.1.8 AT Released Firmware Differences	21
2.2 Brief Introduction to AT Firmware	23
3 AT Command Set	25
3.1 Basic AT Commands	25
3.1.1 Introduction	26
3.1.2 AT: Test AT Startup	26
3.1.3 AT+RST: Restart a Module	26
3.1.4 AT+GMR: Check Version Information	26
3.1.5 AT+CMD: List all AT commands and types supported in current firmware	27
3.1.6 AT+GSLP: Enter Deep-sleep Mode	27
3.1.7 ATE: Configure AT Commands Echoing	28
3.1.8 AT+RESTORE: Restore Factory Default Settings	28
3.1.9 AT+SAVETRANSLINK: Set Whether to Enter Network or Bluetooth LE Passthrough Mode on Power-up	29
3.1.10 AT+TRANSINTVL: Set the Data Transmission Interval in Passthrough Mode	32
3.1.11 AT+UART_CUR: Current UART Configuration, Not Saved in Flash	33
3.1.12 AT+UART_DEF: Default UART Configuration, Saved in Flash	34
3.1.13 AT+SLEEP: Set the Sleep Mode	35
3.1.14 AT+SYSRAM: Query the Heap Memory Status	37
3.1.15 AT+SYMSG: Query or Set System Prompt Information	38
3.1.16 AT+SYMSGFILTER: Enable or Disable the system message Filter	39

3.1.17	AT+SYSMSGFILTERCFG: Query or Set the system message Filters	41
3.1.18	AT+SYSFLASH: Query or Set User Partitions in Flash	44
3.1.19	AT+SYSMFG: Query or Set manufacturing nvs User Partitions	45
3.1.20	AT+RFPOWER: Query or Set RF TX Power	48
3.1.21	Note	49
3.1.22	AT: RF Full Calibration	49
3.1.23	Note	50
3.1.24	AT+SYSROLLBACK: Roll Back to the Previous Firmware	50
3.1.25	AT+SYSTIMESTAMP: Query or Set Local Time Stamp	51
3.1.26	AT+SYSLOG: Enable or Disable the AT Error Code Prompt	51
3.1.27	AT+SLEEPWKCFG: Set the Light-sleep Wakeup Source and Awake GPIO	53
3.1.28	AT+SYSSTORE: Query or Set Parameter Store Mode	54
3.1.29	AT+SYSREG: Read or Write the Register	55
3.2	Wi-Fi AT Commands	56
3.2.1	Introduction	56
3.2.2	AT+CWINIT: Initialize or Deinitialize Wi-Fi Driver	57
3.2.3	AT+CWMODE: Query or Set the Wi-Fi Mode (Station/SoftAP/Station+SoftAP)	58
3.2.4	AT+CWBANDWIDTH: Query or Set Wi-Fi Bandwidth	59
3.2.5	AT+CWSTATE: Query the Wi-Fi state and Wi-Fi information	60
3.2.6	AT+CWCONFIG: Query or Set Wi-Fi Inactive Time and Listen Interval Time	60
3.2.7	AT+CWJAP: Connect to an AP	61
3.2.8	AT+CWRECONNCFG: Query or Set the Wi-Fi Reconnecting Configuration	64
3.2.9	AT+CWLAPOPT: Set the Configuration for the Command AT+CWLAP	65
3.2.10	AT+CWLAP: List Available APs	66
3.2.11	AT+CWQAP: Disconnect from an AP	68
3.2.12	AT+CWSAP: Query or set the configuration of an ESP32 SoftAP	68
3.2.13	AT+CWLIF: Obtain IP Address of the Station That Connects to an ESP32 SoftAP	69
3.2.14	AT+CWQIF: Disconnect Stations from an ESP32 SoftAP	70
3.2.15	AT+CWDHCP: Enable or disable DHCP	70
3.2.16	AT+CWDHCPS: Query or Set the IPv4 Addresses Allocated by an ESP32 SoftAP DHCP Server	71
3.2.17	AT+CWAUTOCONN: Query or Set Automatic Connection to an AP When Powered on	72
3.2.18	AT+CWAPPROTO: Query or Set the Wi-Fi Protocol Standard of SoftAP Mode	73
3.2.19	AT+CWSTAPROTO: Query or Set the Wi-Fi Protocol Standard of Station Mode	74
3.2.20	AT+CIPSTAMAC: Query or Set the MAC Address of an ESP32 Station	75
3.2.21	AT+CIPAPMAC: Query or Set the MAC Address of an ESP32 SoftAP	76
3.2.22	AT+CIPSTA: Query or Set the IP Address of an ESP32 Station	77
3.2.23	AT+CIPAP: Query or Set the IP Address of an ESP32 SoftAP	78
3.2.24	AT+CWSTARTSMART: Start SmartConfig	79
3.2.25	AT+CWSTOPSMART: Stop SmartConfig	80
3.2.26	AT+WPS: Enable the WPS Function	80
3.2.27	AT+CWJEAP: Connect to a WPA2 Enterprise AP	81
3.2.28	AT+CWHOHOSTNAME: Query or Set the Host Name of an ESP32 Station	83
3.2.29	AT+CWCOUNTRY: Query or Set the Wi-Fi Country Code	84
3.3	TCP/IP AT Commands	85
3.3.1	Introduction	86
3.3.2	AT+CIPV6: Enable or disable the network of Internet Protocol Version 6 (IPv6)	86
3.3.3	AT+CIPSTATE: Obtain the TCP/UDP/SSL Connection Information	87
3.3.4	AT+CIPDOMAIN: Resolve a Domain Name	88
3.3.5	AT+CIPSTART: Establish TCP Connection, UDP Transmission, or SSL Connection	89
3.3.6	AT+CIPSTARTEX: Establish TCP connection, UDP transmission, or SSL connection with an Automatically Assigned ID	94
3.3.7	[Data Mode Only] +++: Exit from Data Mode	94
3.3.8	AT+CIPSEND: Send Data in the Normal Transmission Mode or Network Passthrough Mode	94
3.3.9	AT+CIPSENDL: Send Long Data in Parallel in the Normal Transmission Mode.	95
3.3.10	AT+CIPSENDCFG: Set the Configuration for the Command AT+CIPSENDL	97
3.3.11	AT+CIPSENDEX: Send Data in the Normal Transmission Mode in Expanded Ways	98
3.3.12	AT+CIPCLOSE: Close TCP/UDP/SSL Connection	99

3.3.13	AT+CIPCONNPERSIST: Query or Set TCP/SSL Connection Persistence	99
3.3.14	AT+CIFSR: Obtain the Local IP Address and MAC Address	100
3.3.15	AT+CIPMUX: Enable or disable Multiple Connections	101
3.3.16	AT+CIPSERVER: Delete or create a TCP/SSL Server	102
3.3.17	AT+CIPSERVERMAXCONN: Query or Set the Maximum Connections Allowed by a Server	103
3.3.18	AT+CIPMODE: Query or Set the Transmission Mode	104
3.3.19	AT+CIPSTO: Query or Set the local TCP/SSL Server Timeout	105
3.3.20	AT+CIPSNTPCFG: Query or Set the Time Zone and the SNTP Server	106
3.3.21	AT+CIPSNTPTIME: Query the SNTP Time	107
3.3.22	AT+CIPSNTPTINTV: Query or Set the SNTP time synchronization interval	108
3.3.23	AT+CIPFWVER: Query the Existing AT Firmware Version on the Server	109
3.3.24	AT+CIUPDATE: Upgrade Firmware Through Network	109
3.3.25	AT+CIPDINFO: Set “+IPD” Message Mode	111
3.3.26	AT+CIPSSLCCONF: Query or Set SSL Clients	112
3.3.27	AT+CIPSSLCCIPHER: Query or Set the Cipher Suite of the SSL Client	113
3.3.28	AT+CIPSSLCCN: Query or Set the Common Name of the SSL Client	114
3.3.29	AT+CIPSSLCSNI: Query or Set SSL Client Server Name Indication (SNI)	115
3.3.30	AT+CIPSSLCALPN: Query or Set SSL Client Application Layer Protocol Negotiation (ALPN)	116
3.3.31	AT+CIPSSLCPSK: Query or Set SSL Client Pre-shared Key (PSK) in String Format . . .	117
3.3.32	AT+CIPSSLCPSKHEX: Query or Set SSL Client Pre-shared Key (PSK) in Hexadecimal Format	117
3.3.33	AT+CIPRECONNINTV: Query or Set the TCP/UDP/SSL reconnection Interval for the Network Passthrough Mode	118
3.3.34	AT+CIPRECVMODE: Query or Set Socket Receiving Mode	119
3.3.35	AT+CIPRECVDATA: Obtain Socket Data in Passive Receiving Mode	120
3.3.36	AT+CIPRECLEN: Obtain Socket Data Length in Passive Receiving Mode	121
3.3.37	AT+PING: Ping the Remote Host	122
3.3.38	AT+CIPDNS: Query or Set DNS Server Information	122
3.3.39	AT+MDNS: Configure the mDNS Function	124
3.3.40	AT+CIPTCPOPT: Query or Set the Socket Options	126
3.4	Bluetooth® Low Energy AT Commands	128
3.4.1	Introduction	129
3.4.2	AT+BLEINIT: Bluetooth LE Initialization	129
3.4.3	AT+BLEADDR: Query or Set Bluetooth LE Device Address	130
3.4.4	AT+BLENAME: Query or Set Bluetooth LE Device Name	131
3.4.5	AT+BLES SCANPARAM: Query or Set Parameters of Bluetooth LE Scanning	132
3.4.6	AT+BLES SCAN: Enable Bluetooth LE Scanning	133
3.4.7	AT+BLES SCANRSPDATA: Set Bluetooth LE Scan Response	134
3.4.8	AT+BLES ADVPARAM: Query or Set Parameters of Bluetooth LE Advertising	135
3.4.9	AT+BLES ADVDATA: Set Bluetooth LE Advertising Data	136
3.4.10	AT+BLES ADVDATAEX: Automatically Set Bluetooth LE Advertising Data	137
3.4.11	AT+BLES ADVSTART: Start Bluetooth LE Advertising	138
3.4.12	AT+BLES ADVSTOP: Stop Bluetooth LE Advertising	139
3.4.13	AT+BLES CONN: Establish Bluetooth LE Connection	139
3.4.14	AT+BLES CONNPARAM: Query/Update Parameters of Bluetooth LE Connection	141
3.4.15	AT+BLES DISCONN: End Bluetooth LE Connection	142
3.4.16	AT+BLES DATALEN: Set Bluetooth LE Data Packet Length	142
3.4.17	AT+BLES CFGMTU: Set Bluetooth LE MTU Length	143
3.4.18	AT+BLES GATTSSRVCRE: GATTS Creates Services	144
3.4.19	AT+BLES GATTSSRVSTART: GATTS Starts Services	144
3.4.20	AT+BLES GATTSSRVSTOP: GATTS Stops Services	145
3.4.21	AT+BLES GATTSSRV: GATTS Discovers Services	146
3.4.22	AT+BLES GATTSSCHAR: GATTS Discovers Characteristics	146
3.4.23	AT+BLES GATTSSNTFY: Notify a Client of the Value of a Characteristic Value from the Server	147
3.4.24	AT+BLES GATTSSIND: Indicate the Characteristic Value from the Server to a Client	148

3.4.25	AT+BLEGATTSSSETATTR: GATTS Sets Characteristics	149
3.4.26	AT+BLEGATTCPRIMSRV: GATTC Discovers Primary Services	150
3.4.27	AT+BLEGATTCINCLSRV: GATTC Discovers Included Services	150
3.4.28	AT+BLEGATTCCHAR: GATTC Discovers Characteristics	151
3.4.29	AT+BLEGATTCRD: GATTC Reads Characteristics	152
3.4.30	AT+BLEGATTCWR: GATTC Writes Characteristics	153
3.4.31	AT+BLESPPCFG: Query or Set Bluetooth LE SPP Parameters	154
3.4.32	AT+BLESPP: Enter Bluetooth LE SPP Mode	155
3.4.33	AT+BLESECPARAM: Query or Set Bluetooth LE Encryption Parameters	156
3.4.34	AT+BLEENC: Initiate Bluetooth LE Encryption Request	157
3.4.35	AT+BLEENCRSP: Respond to the Pairing Request from the Peer Device	158
3.4.36	AT+BLEKEYREPLY: Reply the Key Value to the Peer Device	159
3.4.37	AT+BLECONFREPLY: Reply the Confirm Value to the Peer Device in the Legacy Con- nection Stage	159
3.4.38	AT+BLEENCDEV: Query Bonded Bluetooth LE Encryption Device List	160
3.4.39	AT+BLEENCCLEAR: Clear Bluetooth LE Encryption Device List	160
3.4.40	AT+BLESETKEY: Set Bluetooth LE Static Pair Key	161
3.4.41	AT+BLEHIDINIT: Bluetooth LE HID Profile Initialization	162
3.4.42	AT+BLEHIDKB: Send Bluetooth LE HID Keyboard Information	163
3.4.43	AT+BLEHIDMUS: Send Bluetooth LE HID Mouse Information	163
3.4.44	AT+BLEHIDCONSUMER: Send Bluetooth LE HID Consumer Information	164
3.4.45	AT+BLUFI: Start or Stop BluFi	165
3.4.46	AT+BLUFINAME: Query or Set BluFi Device Name	166
3.4.47	AT+BLUFISEND: Send User-Customized Data over BluFi	167
3.4.48	AT+BLERDRSSI: Query the current connection RSSI	168
3.4.49	AT+BLEWL: Set the white list	168
3.5	ESP32 Classic Bluetooth® AT Commands	169
3.5.1	Introduction	169
3.5.2	AT+BTINIT: Classic Bluetooth Initialization	170
3.5.3	AT+BTNAME: Query or Set Classic Bluetooth Device Name	171
3.5.4	AT+BTSCANMODE: Set Classic Bluetooth Scan Mode	171
3.5.5	AT+BTSTARTDISC: Start Classic Bluetooth Discovery	172
3.5.6	AT+BTSPPINIT: Classic Bluetooth SPP Profile Initialization	173
3.5.7	AT+BTSPPCONN: Query or establish SPP Connection	174
3.5.8	AT+BTSPDISCONN: End SPP Connection	175
3.5.9	AT+BTSPSEND: Send Data to Remote Classic Bluetooth SPP Device	176
3.5.10	AT+BTSPSTART: Start Classic Bluetooth SPP Profile	176
3.5.11	AT+BTA2DPINIT: Classic Bluetooth A2DP Profile Initialization	177
3.5.12	AT+BTA2DPCONN: Query or establish A2DP Connection	178
3.5.13	AT+BTA2DPDISCONN: End A2DP Connection	179
3.5.14	AT+BTA2DPSRC: Query or Set the Audio File URL	179
3.5.15	AT+BTA2DPCTRL: Control the Audio Play	180
3.5.16	AT+BTSECPARAM: Query or Set the Classic Bluetooth Security Parameters	181
3.5.17	AT+BTKEYREPLY: Input the Simple Pair Key	182
3.5.18	AT+BTPINREPLY: Input the Legacy Pair PIN Code	182
3.5.19	AT+BTSECCFM: Reply the Confirm Value to the Peer Device in the Legacy Connection Stage	183
3.5.20	AT+BTENCDEV: Query Classic Bluetooth Encryption Device List	183
3.5.21	AT+BTENCCLEAR: Clear Classic Bluetooth Encryption Device List	184
3.5.22	AT+BTCOD: Set Class of Devices	184
3.5.23	AT+BTPOWER: Query or Set TX power of Classic Bluetooth	185
3.6	MQTT AT Commands	186
3.6.1	Introduction	186
3.6.2	AT+MQTTUSERCFG: Set MQTT User Configuration	186
3.6.3	AT+MQTTLONGCLIENTID: Set MQTT Client ID	187
3.6.4	AT+MQTTLONGUSERNAME: Set MQTT Username	188
3.6.5	AT+MQTTLONGPASSWORD: Set MQTT Password	188
3.6.6	AT+MQTTCONNCFG: Set Configuration of MQTT Connection	189

3.6.7	AT+MQTTALPN: Set MQTT Application Layer Protocol Negotiation (ALPN)	190
3.6.8	AT+MQTTSNI: Set MQTT Server Name Indication (SNI)	190
3.6.9	AT+MQTTCONN: Connect to MQTT Brokers	191
3.6.10	AT+MQTTPUB: Publish MQTT Messages in String	192
3.6.11	AT+MQTTPUBRAW: Publish Long MQTT Messages	193
3.6.12	AT+MQTTSUB: Subscribe to MQTT Topics	194
3.6.13	AT+MQTTUNSUB: Unsubscribe from MQTT Topics	195
3.6.14	AT+MQTTCLEAN: Close MQTT Connections	195
3.6.15	MQTT AT Error Codes	196
3.6.16	MQTT AT Notes	197
3.7	HTTP AT Commands	198
3.7.1	Introduction	198
3.7.2	AT+HTTPCLIENT: Send HTTP Client Request	198
3.7.3	AT+HTTPGETSIZE: Get HTTP Resource Size	199
3.7.4	AT+HTTPPCGET: Get HTTP Resource	200
3.7.5	AT+HTTPPCPOST: Post HTTP data of specified length	200
3.7.6	AT+HTTPPCPUT: Put HTTP data of specified length	201
3.7.7	AT+HTTPURLCFG: Set or get long HTTP URL	202
3.7.8	AT+HTTPPCHEAD: Set or query HTTP Request Headers	203
3.7.9	AT+HTTPPCFG: Set HTTP Client Configuration	204
3.7.10	HTTP AT Error Codes	204
3.8	FileSystem AT Commands	205
3.8.1	Introduction	205
3.8.2	AT+FS: Filesystem Operations	205
3.8.3	AT+FSMOUNT: Mount or unmount Filesystem	207
3.9	WebSocket AT Commands	207
3.9.1	Introduction	208
3.9.2	AT+WSCFG: Set the WebSocket Configuration	208
3.9.3	AT+WSHEAD: Set or query WebSocket Request Headers	209
3.9.4	AT+WSOPEN: Query or open a WebSocket Connection	210
3.9.5	AT+WSEND: Send Data to a WebSocket Connection	211
3.9.6	AT+WSDATAFMT: Set WebSocket Received Data Format	212
3.9.7	AT+WSCLOSE: Close a WebSocket Connection	213
3.10	ESP32 Ethernet AT Commands	213
3.10.1	Introduction	213
3.10.2	Prerequisite	213
3.10.3	AT+CIPETHMAC: Query or set the MAC Address of the ESP32 Ethernet	214
3.10.4	AT+CIPETH: Query or set the IP Address of the the ESP32 Ethernet	215
3.11	Signaling Test AT Commands	216
3.11.1	Introduction	216
3.11.2	AT+FACTPLCP: Send with Long or Short PLCP	216
3.12	Web Server AT Commands	216
3.12.1	Introduction	217
3.12.2	AT+WEBSERVER: Enable or disable Wi-Fi connection configuration via Web server	217
3.13	Driver AT Commands	218
3.13.1	Introduction	218
3.13.2	AT+DRVADC: Read ADC Channel Value	218
3.13.3	AT+DRVPWMINIT: Initialize PWM Driver	219
3.13.4	AT+DRVPWMDUTY: Set PWM Duty	220
3.13.5	AT+DRVPWMFADE: Set PWM Fade	220
3.13.6	AT+DRVI2CINIT: Initialize I2C Master Driver	221
3.13.7	AT+DRVI2CRD: Read I2C Data	222
3.13.8	AT+DRVI2CWRDATA: Write I2C Data	222
3.13.9	AT+DRVI2CWRBYTES: Write No More Than 4 Bytes I2C Data	223
3.13.10	AT+DRVSPICONFGPIO: Configure SPI GPIO	224
3.13.11	AT+DRVSPINIT: Initialize SPI Master Driver	224
3.13.12	AT+DRVSPIRD: Read SPI Data	225
3.13.13	AT+DRVSPIWR: Write SPI Data	225

3.14	User AT Commands	226
3.14.1	Introduction	227
3.14.2	AT+USERRAM: Operate user's free RAM	227
3.14.3	AT+USEROTA: Upgrade the Firmware According to the Specified URL	228
3.14.4	AT+USERWKMCFG: Configure How AT Wakes Up MCU	229
3.14.5	AT+USERMCUSLEEP: MCU Indicates Its Sleep State	231
3.14.6	AT+USERDOCS: Query the ESP-AT User Guide for Current Firmware	231
3.15	AT Command Types	232
3.16	AT Commands with Configuration Saved in the Flash	233
3.17	AT Messages	233
3.18	AT Logs	236
4	AT Command Examples	239
4.1	AT Response Message Format Control Examples	239
4.1.1	Enable the system message filter to download files in HTTP passthrough mode	239
4.2	AT+SYSMFG Command Examples	241
4.2.1	Introduction	242
4.2.2	Factory Parameter Configuration	244
4.2.3	PKI Configuration	251
4.2.4	GATTS Configuration	257
4.2.5	Custom Data Storage	262
4.3	TCP/IP AT Examples	265
4.3.1	ESP32 as a TCP client in single connection	265
4.3.2	ESP32 as a TCP server in multiple connections	266
4.3.3	UDP transmission with fixed remote IP address and port	268
4.3.4	UDP transmission with changeable remote IP address and port	270
4.3.5	ESP32 as an SSL client in single connection	272
4.3.6	ESP32 as an SSL server in multiple connections	273
4.3.7	ESP32 as an SSL client to create a single connection with mutual authentication	275
4.3.8	ESP32 as an SSL server to create multiple connections with mutual authentication	278
4.3.9	Network passthrough transmission when the ESP32 works as a TCP client in single connection	280
4.3.10	Network passthrough transmission when the ESP32 works as a TCP server	281
4.3.11	Network passthrough transmission when the ESP32 works as a softAP in UDP transparent transmission	283
4.3.12	ESP32 obtains socket data in passive receiving mode	285
4.3.13	ESP32 enables mDNS service, PC connects to the device's TCP server using a domain name	287
4.4	Bluetooth LE AT Examples	288
4.4.1	Introduction	288
4.4.2	Bluetooth LE client reads and write services	290
4.4.3	Bluetooth LE server read and write services	293
4.4.4	Encrypt Bluetooth LE connection	298
4.4.5	Establish SPP connection between two ESP32 development boards and transmit data in UART-Bluetooth LE Passthrough Mode	303
4.4.6	Establish SPP connection between ESP32 and mobile phone and transmit data in UART-Bluetooth LE passthrough mode	307
4.4.7	Establish Bluetooth LE connection and pairing between ESP32 and mobile phone	309
4.5	MQTT AT Examples	311
4.5.1	MQTT over TCP (with a local MQTT broker)(suitable for a small amount of data)	312
4.5.2	MQTT over TCP (with a local MQTT broker)(suitable for large amounts of data)	313
4.5.3	MQTT over TLS (with a local MQTT broker)	315
4.5.4	MQTT over WSS	316
4.6	MQTT AT Examples for Cloud	318
4.6.1	Obtain certificates and endpoints from AWS IoT	318
4.6.2	Connect to AWS IoT based on mutual authentication with MQTT AT commands	319
4.7	ESP32 Ethernet AT Examples	323
4.7.1	Establish a TCP connection on Ethernet network	323
4.8	Web Server AT Example	324

4.8.1	Wi-Fi Provisioning Using a Browser	324
4.8.2	OTA Firmware Upgrade Using a Browser	330
4.8.3	Wi-Fi Provisioning Using a WeChat Applet	335
4.8.4	OTA Firmware Upgrade Using a WeChat Applet	346
4.8.5	ESP32 Using Captive Portal	346
4.9	HTTP AT Examples	347
4.9.1	The HEAD method of HTTP client	347
4.9.2	The GET method of HTTP client	348
4.9.3	The POST method of HTTP client (suitable for POST small amount of data)	349
4.9.4	The POST method of HTTP client (recommended method)	351
4.9.5	The PUT method of HTTP client (suitable for no data put)	352
4.9.6	The PUT method of HTTP client (recommended method)	353
4.9.7	The DELETE method of HTTP client	355
4.10	WebSocket AT Examples	356
4.10.1	WebSocket Connection over TCP	356
4.10.2	WebSocket Connection over TLS (Mutual Authentication)	358
4.11	ESP32 Classic Bluetooth AT Examples	361
4.11.1	Establish SPP connection between phone (or PC) and ESP32 in Normal Transmission mode with IO capability set to NoInputNoOutput	362
4.11.2	Establish SPP connection between phone (or PC) and ESP32 in Passthrough mode with IO capability set to NoInputNoOutput	363
4.11.3	Establish SPP connection between phone (or PC) and ESP32 with IO capability set to KeyboardOnly	365
4.11.4	Establish SPP connection between two ESP32 development boards	366
4.11.5	Establish A2DP connection and enable A2DP Sink to play music	369
4.11.6	Query and clear Classic Bluetooth encryption device list	370
4.12	Sleep AT Examples	370
4.12.1	Introduction	371
4.12.2	Set Modem-sleep mode in Wi-Fi mode	371
4.12.3	Set Light-sleep mode in Wi-Fi mode	372
4.12.4	Set Light-sleep mode in Bluetooth LE advertising mode	373
4.12.5	Set Light-sleep mode in Bluetooth LE connection mode	374
4.12.6	Set Deep-sleep mode	375
4.13	AT Network Provisioning Examples	375
4.13.1	BluFi Provisioning	376
4.13.2	SmartConfig Provisioning	377
4.13.3	SoftAP Provisioning	378
4.13.4	WPS Provisioning	378
5	ESP-AT Versions	381
5.1	Releases	381
5.2	Which Version Should I Start With?	381
5.3	Versioning Scheme	381
5.3.1	ESP-AT Firmware Release Management	381
5.3.2	ESP-AT Branch Release Management	382
5.4	Support Periods	382
5.5	Check the Current AT Firmware Version	384
5.6	Subscribe to AT Releases	384
5.6.1	Step 1: Log in to Your GitHub Account	384
5.6.2	Step 2: Choose Customized Notifications	384
5.6.3	Step 3: Apply for Release Notifications	384
6	How to Compile and Develop Your Own AT Project	387
6.1	Compile ESP-AT Project Locally	387
6.1.1	Detailed Steps	387
6.1.2	Step 1. Get Started with ESP-IDF	387
6.1.3	Step 2. Get ESP-AT	388
6.1.4	Step 3. Install Environment	388

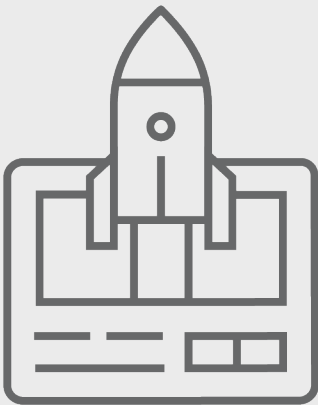
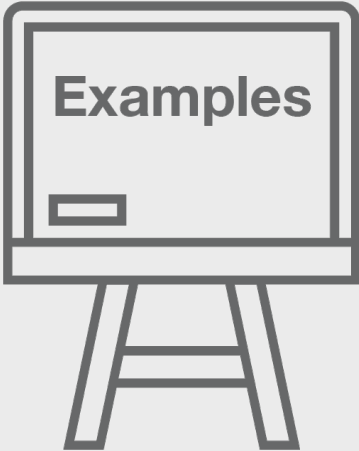
6.1.5	Step 4. Connect Your Device	389
6.1.6	Step 5. Configure	389
6.1.7	Step 6. Build the Project	390
6.1.8	Step 7. Flash onto the Device	390
6.1.9	build.py Advanced Usage	390
6.2	Compile ESP-AT Project on the GitHub Webpage	391
6.2.1	Detailed Steps	391
6.2.2	Step 1. Log in to your GitHub account	391
6.2.3	Step 2. Fork the ESP-AT project	391
6.2.4	Step 3. Enable GitHub Actions	391
6.2.5	Step 4. Configure the secrets required to compile the ESP-AT project	395
6.2.6	Step 5. Use the github.dev editor to modify and submit the code	396
6.2.7	Step 6. Compile the AT firmware using GitHub Actions	405
6.3	How to Set AT Port Pins	405
6.3.1	ESP32 Series	405
6.4	How to Add User-defined AT Commands	406
6.4.1	Customize AT Commands	406
6.4.2	Customize Complex AT Commands	411
6.5	How to Improve ESP-AT Throughput Performance	418
6.5.1	(Simple) Quick Configuration	418
6.5.2	(Recommended) Understand Data Stream and Make the Precise Configuration	419
6.6	How to Update mfg_nvs Partition	422
6.6.1	mfg_nvs Partition	422
6.6.2	Generate mfg_nvs.bin	422
6.7	How to Update Factory Parameters	423
6.7.1	Factory Parameter Configuration	423
6.8	How to Update PKI Configuration	424
6.8.1	Introduction to PKI Configuration	425
6.9	How to Customize Bluetooth® LE Services	426
6.9.1	Bluetooth LE Service Source File	426
6.9.2	Customize Bluetooth LE Services during Compilation	428
6.10	How to Customize Partitions	429
6.10.1	Modify at_customize.csv	429
6.10.2	Generate at_customize.bin	430
6.10.3	Flash at_customize.bin into ESP32 Device	430
6.10.4	Example	431
6.11	How to Enable ESP-AT Ethernet	431
6.11.1	Overview	431
6.11.2	Step 1. Configure and Flash	431
6.11.3	Step 2. Connect the Board and Test	432
6.12	How to Add Support for a Module	432
6.12.1	Step 1: Configure the Factory Parameters for the New Module	432
6.12.2	Step 2: Configure OTA for the Newly Added Module	433
6.12.3	Step 3: Add New Module Configuration	434
6.13	ESP32 SDIO AT Guide	436
6.13.1	Introduction	436
6.13.2	Implement SDIO AT	437
6.13.3	SDIO Data Transimission	437
6.14	How to Implement OTA Upgrade	438
6.14.1	Comparison Among OTA Commands and Their Application Scenarios	438
6.14.2	Perform OTA Upgrade with ESP-AT OTA Commands	439
6.15	How to Update the ESP-IDF Version	445
6.16	ESP-AT Firmware Differences	446
6.17	How to Download the Latest Temporary Version of AT Firmware from GitHub	446
6.18	at.py Tool	451
6.18.1	Detailed Steps	451
6.18.2	Step 1: Install Python	451
6.18.3	Step 2: Download at.py	451

6.18.4	Step 3: Use at.py	451
6.18.5	Step 4: Examples: Modify Firmware Configurations with at.py	451
6.19	AT API Reference	455
6.19.1	Header File	455
6.19.2	Functions	455
6.19.3	Structures	459
6.19.4	Macros	461
6.19.5	Type Definitions	462
6.19.6	Enumerations	462
6.19.7	Header File	465
6.19.8	Functions	465
6.19.9	Macros	467
6.19.10	Enumerations	467
6.20	How to Enable Silence Mode	468
6.20.1	Introduction	468
6.20.2	Configuration	468
6.21	How to Enable More AT Debug Logs	469
6.21.1	AT Wi-Fi Function Debugging	470
6.21.2	AT Network Function Debugging	470
6.21.3	AT BLE Function Debugging	471
6.21.4	AT Interface Function Debugging	472
6.21.5	Debugging Examples	472
7	Customized AT Commands and Firmware For Third-party Open Cloud Platforms	475
8	AT FAQ	477
8.1	AT Documentation	478
8.1.1	How to select the correct AT documentation version?	478
8.2	AT Firmware	478
8.2.1	There is no released firmware for my module. Where can I get the firmware for it?	478
8.2.2	How to get the source code of AT firmware?	479
8.2.3	How to download the AT firmware on Espressif's official website?	479
8.2.4	How to combine all the bin files compiled by ESP-AT?	479
8.2.5	Why does a newly purchased ESP series module print "flash read err" error after power-up?	479
	How to use AT commands?	479
8.2.6	Does the AT firmware shipped in modules support flow control?	479
8.3	AT Commands and Responses	479
8.3.1	Why does AT prompt busy p...?	479
8.3.2	Why does the AT firmware always return the following message after I power up the device and sent the first command?	479
8.3.3	What commands are supported by the default AT firmware on different modules, and from which version are they supported?	480
8.3.4	When the host MCU sends an AT command to the ESP32 device, there is no response. What is the reason?	480
8.3.5	How to handle special characters in AT commands?	480
8.3.6	Can the serial port baudrate be modified in AT Commands? (Default: 115200)	480
8.3.7	How to modify the UART pins of the AT command port?	480
8.3.8	How to solve TCP/SSL data loss during transmission?	480
8.3.9	After ESP32 enters the passthrough mode using AT commands, can ESP32 give a message if the connected hotspot is disconnected?	480
8.3.10	Why is Wi-Fi disconnected (WIFI DISCONNECT)?	481
8.3.11	What are the common Wi-Fi compatibility issues?	481
8.3.12	Do AT commands support ESP-WIFI-MESH?	481
8.3.13	Is it possible to set the ESP32-WROOM-32 module to HID keyboard mode with AT commands?	481
8.3.14	Can AT command set Bluetooth LE transmit power?	481
8.3.15	How to enable the notify and indicate functions on Bluetooth LE clients?	481
8.4	Hardware	482

8.4.1	How big is the chip flash required for AT firmware on different modules?	482
8.4.2	How to view the error log of AT firmware?	482
8.4.3	The UART1 communication pin used by AT on the ESP32 module is inconsistent with the default UART1 pin described in the ESP32 module's datasheet?	482
8.5	Performance	482
8.5.1	How long does it take for AT to connect to Wi-Fi?	482
8.5.2	Is it possible to change the TCP send window size in AT firmware?	482
8.5.3	How to test and optimize the AT throughput?	482
8.5.4	How to modify the default maximum number of TCP segment retransmissions for AT?	483
8.6	Other	483
8.6.1	What interfaces of ESP32 chips can be used to transmit AT commands?	483
8.6.2	How to enable debug log for AT?	483
8.6.3	How to modify the default Wi-Fi mode when AT firmware starts for the first time?	483
8.6.4	How does the AT command implement the functionality of resuming HTTP transfers after interrupts?	483
8.6.5	How to download files from HTTP server and store them to FATFS, or upload files from FATFS file system to HTTP server?	483
8.6.6	How to conduct BQB certification?	483
8.6.7	Does AT support PPP?	484
9	Index of Abbreviations	485
10	Disclaimer	491
10.1	Application Scenario Verification	491
10.2	Compatibility	491
10.3	Server Upgrade Considerations	491
10.4	Wireless Solution Issue Handling	491
10.5	Usage Restrictions	492
10.6	Limitation of Liability	492
10.7	Third-Party Software and Technology	492
10.8	Data Protection	492
10.9	Version Updates and Maintenance	492
10.10	Disclaimer Updates	492
10.11	User Responsibility	492
11	About	493
11.1	Other Espressif Connectivity Solutions	493
Index		495
Index		495

This is the documentation for [ESP-AT Development Framework](#). ESP-AT project was started and powered by [Espressif Systems](#) as an official project, for the [ESP32](#), [ESP32-C2](#), [ESP32-C3](#), [ESP32-C5](#), [ESP32-C6](#), [ESP32-C61](#), and [ESP32-S2 Series SoCs](#) provided for Windows, Linux, and macOS.

This document describes using ESP-AT with the ESP32 SoC.

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

Chapter 1

Get Started

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

1.1 What is ESP-AT

Important: Before using the ESP-AT, please read the [Disclaimer](#) carefully and comply with its terms and precautions.

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

ESP-AT is a project based on ESP-IDF. It makes an ESP32 board work as a slave, and an MCU as a host. The host MCU sends AT commands to the ESP32 chip and receives AT responses back. ESP-AT provides a wide range of AT commands with different functions, such as Wi-Fi commands, TCP/IP commands, Bluetooth LE commands, Bluetooth commands, MQTT commands, HTTP commands, Ethernet commands, etc.

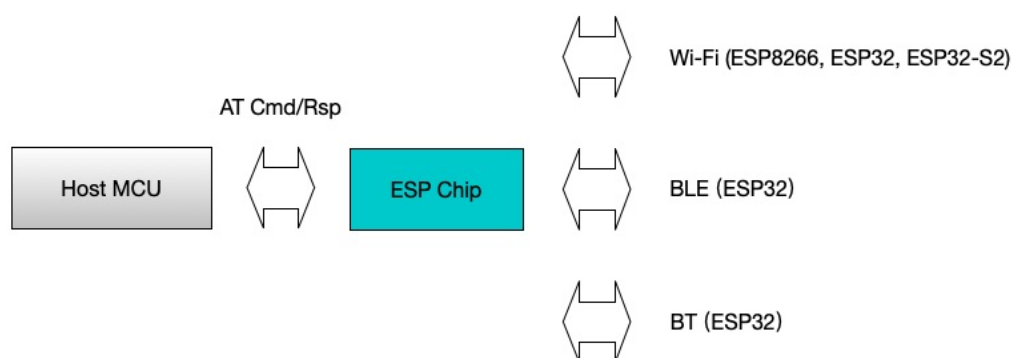


Fig. 1: ESP-AT Overview

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

By default, the host MCU connects to the ESP32 board via UART, and sends/receives AT commands/responses through UART. But you can also use other interfaces, such as SDIO, according to your actual use scenario.

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

1.2 Technology Selection

This document mainly introduces how to choose the right Espressif hardware product, AT software solution, and the initial preparation for your project.

Important: If you have any questions regarding the selection of Espressif hardware products or AT software solutions, please contact [Espressif Sales](#) or [Technical Support](#).

1.2.1 Hardware Selection

Before using ESP-AT, you need to select a suitable Espressif chip to integrate into your product, empowering it with wireless capabilities. Hardware selection is a complex process that requires consideration of various factors such as functionality, power consumption, cost, size, etc. Please read the following content to help you choose the hardware.

- [Product Selector Tool](#) can help you understand the hardware differences of different Espressif products.
- [Datasheets](#) can help you understand the hardware capabilities supported by the chip/module.
- [Board Selection Guide](#) can help you compare the differences between chips, modules, and development boards and provide selection guidance.

Note: The datasheets indicate the maximum hardware capabilities and do not represent the capabilities of the AT software. For example, the ESP32-C6 chip supports Zigbee 3.0 and Thread 1.3, but the existing AT software solutions do not support these two wireless protocols.

1.2.2 AT Software Solution Selection

AT software solution is the AT firmwares provided by Espressif for different chips, which can help you quickly implement wireless functionality.

- If you want to know the detailed AT software capabilities of ESP32 chip, please refer to the [AT Command Set](#).
- If you want to compare the different AT firmware functionalities of ESP32 chip, please refer to the [ESP-AT Firmware Differences](#).

The table below lists the Espressif chips that are currently supported or planned to be supported by the AT firmware, along with a brief comparison. Chips without planned support are excluded.

Chip	Wireless	AT Firmware	Description
ESP32-C6	Wi-Fi 6 + BLE 5.0	v4.1.1.0	
ESP32-C5	5G Wi-Fi + Wi-Fi 6 + BLE 5.0	v5.0.1.0	
ESP32-C61	Wi-Fi 6 + BLE 5.0	v5.0.1.0	
ESP32-C3	Wi-Fi 4 + BLE 5.0	v4.1.1.0	
ESP32-C2	Wi-Fi 4 (or BLE 5.0)	v4.1.1.0	
ESP32	Wi-Fi 4 + BLE v4.2 (+ BT)	v4.1.1.0	
ESP32-S2	Wi-Fi 4	v4.1.1.0	It is recommended to use the more cost-effective ESP32-C series.

- (or BLE 5.0) indicates that Bluetooth LE functionality is supported in the AT software solution, but it is not included in the released firmware.
- (+ BT) indicates that Classic Bluetooth functionality is supported in the AT software solution, but it is not included in the released firmware.

Note: The modules or chips from the factory are not preloaded with AT firmware. If you have mass production requirements, please contact the corresponding business personnel or sales@espressif.com in a timely manner, and we will provide customized production.

Which Type of AT Firmware Shall I Choose?

ESP-AT firmware has the following types. Among them, the workload of downloading or preparing firmware increases from top to bottom, and so does the number of supported module types.

- *Officially Released Firmware (Recommended)*
- *GitHub Temporary Firmware*
- *Firmware with Updated Parameters*
- *Self-Compiled Firmware*

Officially Released Firmware (Recommended) Officially released firmware, also known as “released firmware”, “official firmware” or “default firmware”, has passed testing and periodically released by Espressif’s official team according to the internal development plan. It can be upgraded directly based on the Espressif OTA server. If it fully matches your project needs, it is recommended to choose it over the other types of firmware. If it does not support your module, you can select a firmware that has a similar hardware configuration to your module according to [Hardware Differences](#) for testing and verification.

- How to obtain firmware: [ESP32 AT firmware](#)
- Pros:
 - Stable
 - Reliable
 - Small workload of obtaining firmware
- Cons:
 - Long update cycle
 - A limited number of supported modules
- Reference documentation:
 - [Hardware connection](#)
 - [Firmware Downloading and Flash](#)
 - For which chip series are supported and unsupported by ESP-AT firmware, please refer to ESP-AT GitHub home page [readme.md](#)

GitHub Temporary Firmware **GitHub temporary firmware** is compiled by GitHub whenever code is pushed to GitHub and yet does not reach the firmware release cycle. In other words, it is the firmware in development, including the temporary versions of **officially released firmware** and supported firmware that is not planned for release. The former can be upgraded directly based on the Espressif OTA server.

- How to obtain firmware: Please refer to [How to Download the Latest Temporary Version of AT Firmware from GitHub](#).
- Pros:
 - Real-time integration of new features and bug fixes.
 - Covering some unofficially released firmware, such as firmware based on SDIO communication and firmware based on SPI communication.
 - Small workload of obtaining firmware.
- Cons: As the firmware compiled based on commits that are not for official release has not been thoroughly tested, there may be some risks. You need to do a complete test by yourself.

Firmware with Updated Parameters The **firmware with updated parameters** is generated by updating the parameter area and you do not need to compile the firmware. It is suitable for the case where the firmware function meets the project requirements, but only some parameters do not, such as the UART baud rate and UART GPIO pins. This type of firmware can be directly upgraded based on the Espressif OTA server.

- For how to modify those parameters, please refer to [at.py Tool](#).
- Pros:
 - No need to recompile the firmware.
 - The firmware is stable and reliable.
- Cons: Requiring modification to released firmware, long update cycle, and a limited number of supported modules.

Self-Compiled Firmware When you need to conduct secondary development, you can compile the firmware by yourself. To support the OTA function, you need to deploy your own OTA server.

- For how to compile the firmware by yourself, please refer to [Compile ESP-AT Project Locally](#).
- Pros: You can control functions and cycles.
- Cons: You need to set up the compilation environment.

If stability is your priority, it is recommended to develop your AT firmware based on the latest released version corresponding to the chip. If you want more new features, it is recommended to develop your AT firmware based on the [master branch](#).

1.2.3 Initial Project Preparation

During the initial project preparation phase, it is **strongly recommended** that you choose [Espressif development boards](#) to start your project. In the early stages of the project, it can help you quickly validate prototypes, evaluate hardware and software capabilities, and reduce project risks. In the middle stages of the project, it can help you quickly integrate and verify functions, optimize performance, and improve development efficiency. In the later stages of the project, it can help you quickly simulate and locate issues, and achieve rapid product iteration.

If you are using [Self-Compiled Firmware](#), it is recommended to prioritize using Linux as the development environment.

1.3 Hardware Connection

This document introduces what hardware you need to prepare and how to connect them in order to download AT firmware, send AT commands, and receive AT responses.

The commands supported by each series of AT firmware vary, and its compatibility with modules or chips also differs. For more information, refer to [ESP-AT Firmware Differences](#).

If you don't want to use the default AT pin, you can refer to the [How to Set AT Port Pins](#) document to change the pin.

1.3.1 What You Need

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

Component	Function
ESP32 board	Slave MCU.
USB cable (ESP32 board to PC)	Download/Log output connection.
PC	Act as Host MCU. Download firmware to Slave MCU.
USB cable (PC to serial port converter)	AT command/response connection.
USB to serial port converter	Convert between USB signals and TTL signals.
Jumper wires (serial port converter to ESP32 board)	AT command/response connection.

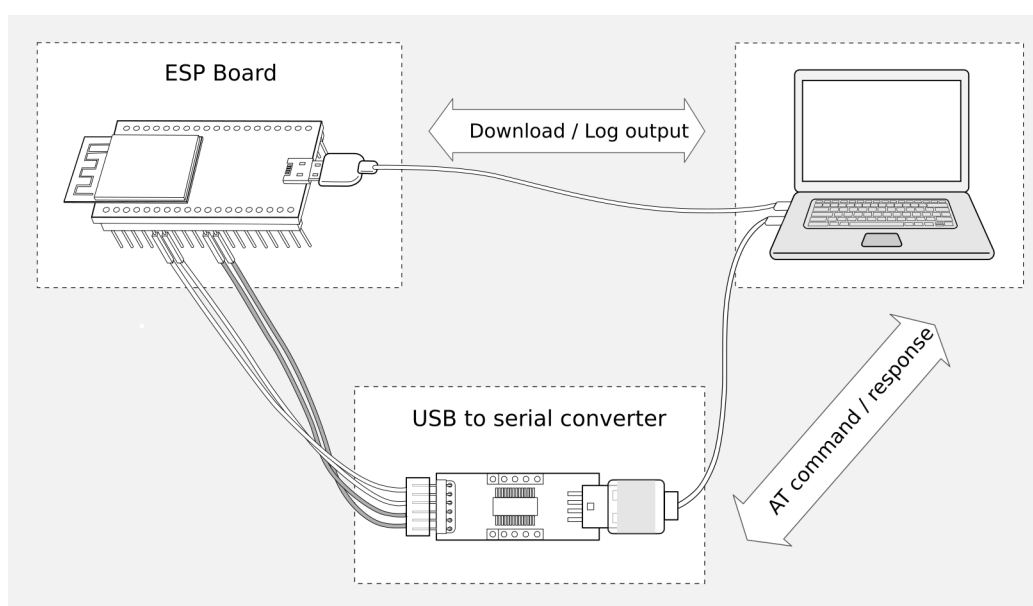


Fig. 2: Connection of Components for ESP-AT Testing

Note:

- In the above picture, four jump wires are used to connect the ESP32 board and USB to serial converter. If you do not use hardware flow control, two wires connecting TX/RX and a simpler converter will be enough.
- If you use an ESP32 module (instead of a development board) and flash firmware via UART, you need to:
 - Reserve the U0RXD and U0TXD pins (refer to [Datasheet > Pin Overview](#))
 - Reserve the strapping pins (refer to [Datasheet > Boot Configurations](#)), enter download mode by controlling the strapping pin level
 - For more details, please refer to [ESP32 Hardware Requirements for Entering Download Mode](#)

1.3.2 ESP32 Series

ESP32 AT uses two UART ports: UART0 is used to download firmware and log output; UART1 is used to send AT commands and receive AT responses. Both UART0 and UART1 use 115200 baud rate for communication by default.

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

For more details of ESP32 modules and boards, please refer to [ESP32 modules](#) and [ESP32 boards](#).

ESP32-WROOM-32 Series

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

Function of Connection	ESP32 Board or Module Pins	Other Device Pins
Download/Log output ¹	UART0 • GPIO3 (RX) • GPIO1 (TX)	PC • TX • RX
AT command/response ²	UART1 • GPIO16 (RX) • GPIO17 (TX) • GPIO15 (CTS) • GPIO14 (RTS)	USB to serial converter • TX • RX • RTS • CTS

Note 1: Connection between individual pins of the ESP32 board and the PC is already established internally on the ESP32 board. You only need to provide USB cable between the board and PC.

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

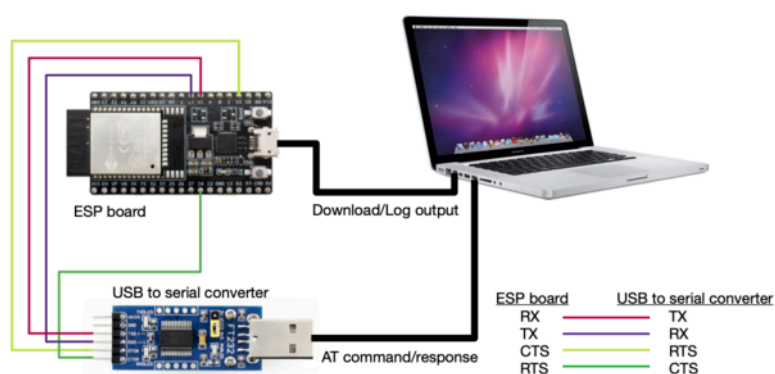


Fig. 3: ESP32-WROOM-32 Series Hardware Connection

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

ESP32-MINI-1 Series

Table 3: ESP32-MINI-1 Series Hardware Connection Pinout

Function of Connection	ESP32 Board or Module Pins	Other Device Pins
Download/Log output ¹	UART0 • GPIO3 (RX) • GPIO1 (TX)	PC • TX • RX
AT command/response ²	UART1 • GPIO19 (RX) • GPIO22 (TX) • GPIO15 (CTS) • GPIO14 (RTS)	USB to serial converter • TX • RX • RTS • CTS

Note 1: Connection between individual pins of the ESP32 board and the PC is already established internally on the ESP32 board. You only need to provide USB cable between the board and PC.

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

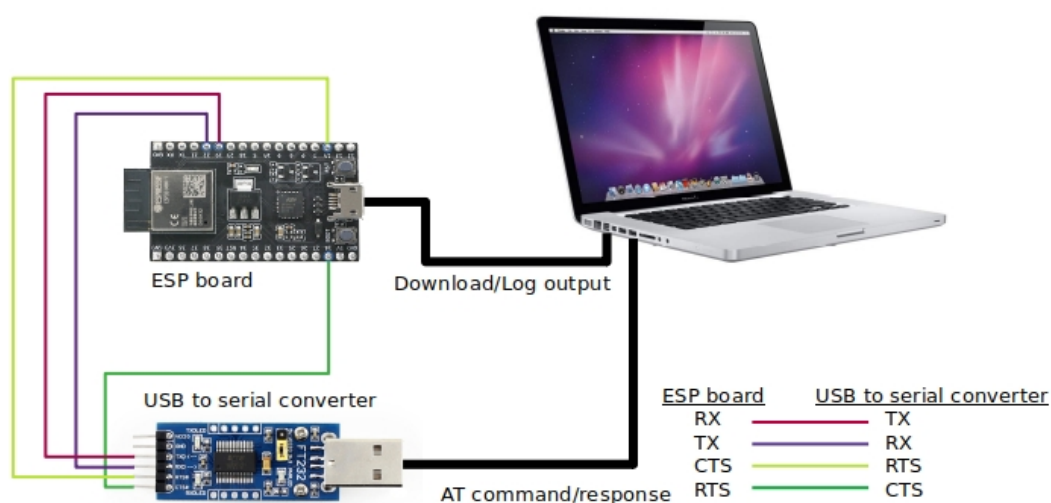


Fig. 4: ESP32-MINI-1 Series Hardware Connection

ESP32-WROVER Series

Table 4: ESP32-WROVER Series Hardware Connection Pinout

Function of Connection	ESP32 Board or Module Pins	Other Device Pins
Download/Log output ¹	UART0 • GPIO3 (RX) • GPIO1 (TX)	PC • TX • RX
AT command/response ²	UART1 • GPIO19 (RX) • GPIO22 (TX) • GPIO15 (CTS) • GPIO14 (RTS)	USB to serial converter • TX • RX • RTS • CTS

Note 1: Connection between individual pins of the ESP32 board and the PC is already established internally on the ESP32 board. You only need to provide USB cable between the board and PC.

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

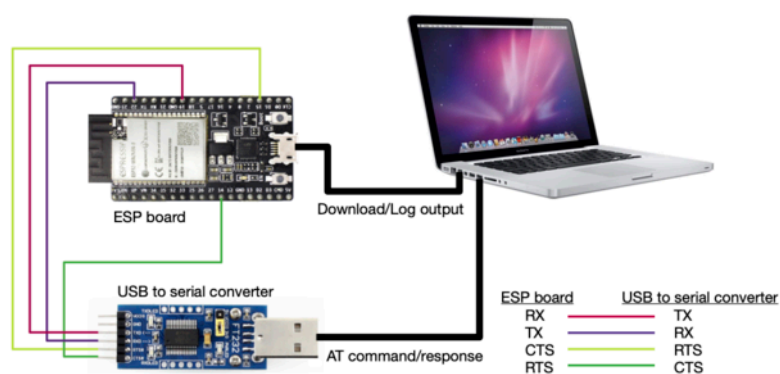


Fig. 5: ESP32-WROVER Series Hardware Connection

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

ESP32-PICO Series

Table 5: ESP32-PICO Series Hardware Connection Pinout

Function of Connection	ESP32 Board or Module Pins	Other Device Pins
Download/Log output ¹	UART0 • GPIO3 (RX) • GPIO1 (TX)	PC • TX • RX
AT command/response ²	UART1 • GPIO19 (RX) • GPIO22 (TX) • GPIO15 (CTS) • GPIO14 (RTS)	USB to serial converter • TX • RX • RTS • CTS

Note 1: Connection between individual pins of the ESP32 board and the PC is already established internally on the ESP32 board. You only need to provide USB cable between the board and PC.

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

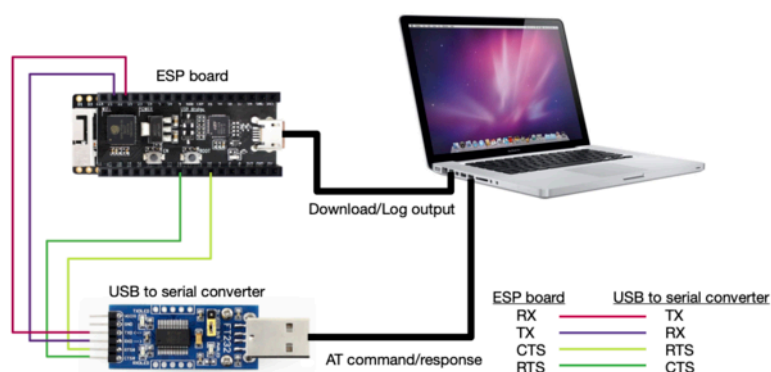


Fig. 6: ESP32-PICO Series Hardware Connection

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

ESP32-SOLO Series

Table 6: ESP32-SOLO Series Hardware Connection Pinout

Function of Connection	ESP32 Board or Module Pins	Other Device Pins
Download/Log output ¹	UART0 • GPIO3 (RX) • GPIO1 (TX)	PC • TX • RX
AT command/response ²	UART1 • GPIO16 (RX) • GPIO17 (TX) • GPIO15 (CTS) • GPIO14 (RTS)	USB to serial converter • TX • RX • RTS • CTS

Note 1: Connection between individual pins of the ESP32 board and the PC is already established internally on the ESP32 board. You only need to provide USB cable between the board and PC.

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

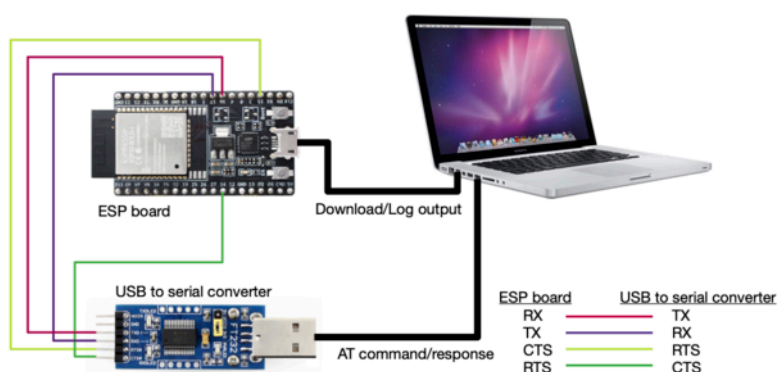


Fig. 7: ESP32-SOLO Series Hardware Connection

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

1.4 Downloading Guide

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

Before you start, please make sure you have already connected your hardware. For more details, see [Hardware Connection](#).

For different series of modules, the commands supported by AT firmware are different. Please refer to [ESP-AT Firmware Differences](#) for more details.

1.4.1 Download AT Firmware

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

- Navigate to [AT Binary Lists](#)
- Find the firmware for your device
- Click the link to download it

Here, we download ESP32-WROOM-32-AT-V3.2.0.0 for ESP32-WROOM-32. The list below describes the structure of this firmware and what each bin file contains. Other AT firmware has similar structure and bin files.

```
.
├── at_customize.bin           // secondary partition table
├── bootloader                 // bootloader
│   └── bootloader.bin
├── customized_partitions      // AT customized binaries
│   └── mfg_nvs.csv           // raw data of manufacturing nvs partition
│       └── mfg_nvs.bin       // manufacturing nvs partition binary
├── download.config            // configuration of downloading
├── esp-at.bin                 // AT application binary
├── esp-at.elf
├── esp-at.map
├── factory                    // A combined bin for factory downloading
│   └── factory_XXX.bin
├── flasher_args.json          // flasher arguments
├── ota_data_initial.bin       // ota data parameters
├── partition_table            // primary partition table
│   └── partition-table.bin
└── sdkconfig                  // compilation configuration for AT firmware
```

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

```
--flash_mode dio --flash_freq 40m --flash_size 4MB
0x1000 bootloader/bootloader.bin
0x8000 partition_table/partition-table.bin
0x10000 ota_data_initial.bin
0x20000 at_customize.bin
0x21000 customized_partitions/mfg_nvs.bin
0x100000 esp-at.bin
```

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

1.4.2 Flash AT Firmware into Your Device

Follow the instructions below for your operating system.

Windows

Before starting to flash, you need to download [Flash Download Tool](#) for Windows. You can refer to the [Flash Download Tool User Guide](#) for instructions.

- Open the ESP32 Flash Download Tool.
- Select chipType. (Here, we select ESP32.)
- Select a workMode according to your need. (Here, we select Developer Mode.)
- Select a loadMode according to your need. (Here, we select uart.)

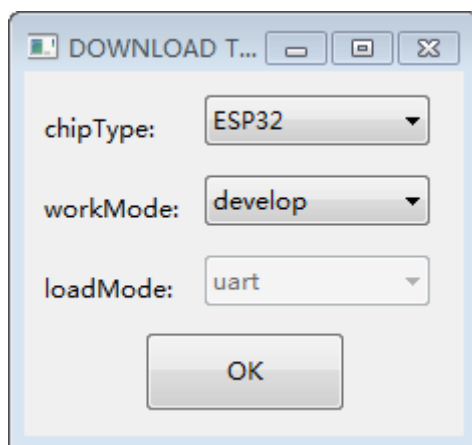


Fig. 8: Firmware Download Configurations

- Flash AT firmware into your device. You can select either of the two ways below.
 - To download one combined factory bin (namely, `factory_XXX.bin` in the `build/factory` directory) to address `0x0`, select “DoNotChgBin” to use the default configuration of the factory bin.
 - To download multiple bins separately to different addresses, set up the configurations according to the file `download.config` and do NOT select “DoNotChgBin”.

In case of flashing issues, please verify what the COM port number of download interface of the ESP32 board is and select it from “COM:” dropdown list. If you do not know the port number, you can refer to [Check port on Windows](#) for details.

When you finish flashing, please [Check Whether AT Works](#).

Linux or macOS

Before you start to flash, you need to install [esptool.py](#).

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

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

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

Replace PORTNAME with your port name. If you do not know it, you can refer to [Check port on Linux and macOS](#) for details.

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

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

```
esptool.py --chip auto --port /dev/tty.usbserial-0001 --baud 115200 --
↳before default_reset --after hard_reset write_flash -z --flash_mode_
↳dio --flash_freq 40m --flash_size 4MB 0x8000 partition_table/
↳partition-table.bin 0x10000 ota_data_initial.bin 0x1000 bootloader/
↳bootloader.bin 0x100000 esp-at.bin 0x20000 at_customize.bin 0x21000_
↳customized_partitions/mfg_nvs.bin
```

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

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

Replace PORTNAME with your port name. If you do not know it, you can refer to [Check port on Linux and macOS](#) for details.

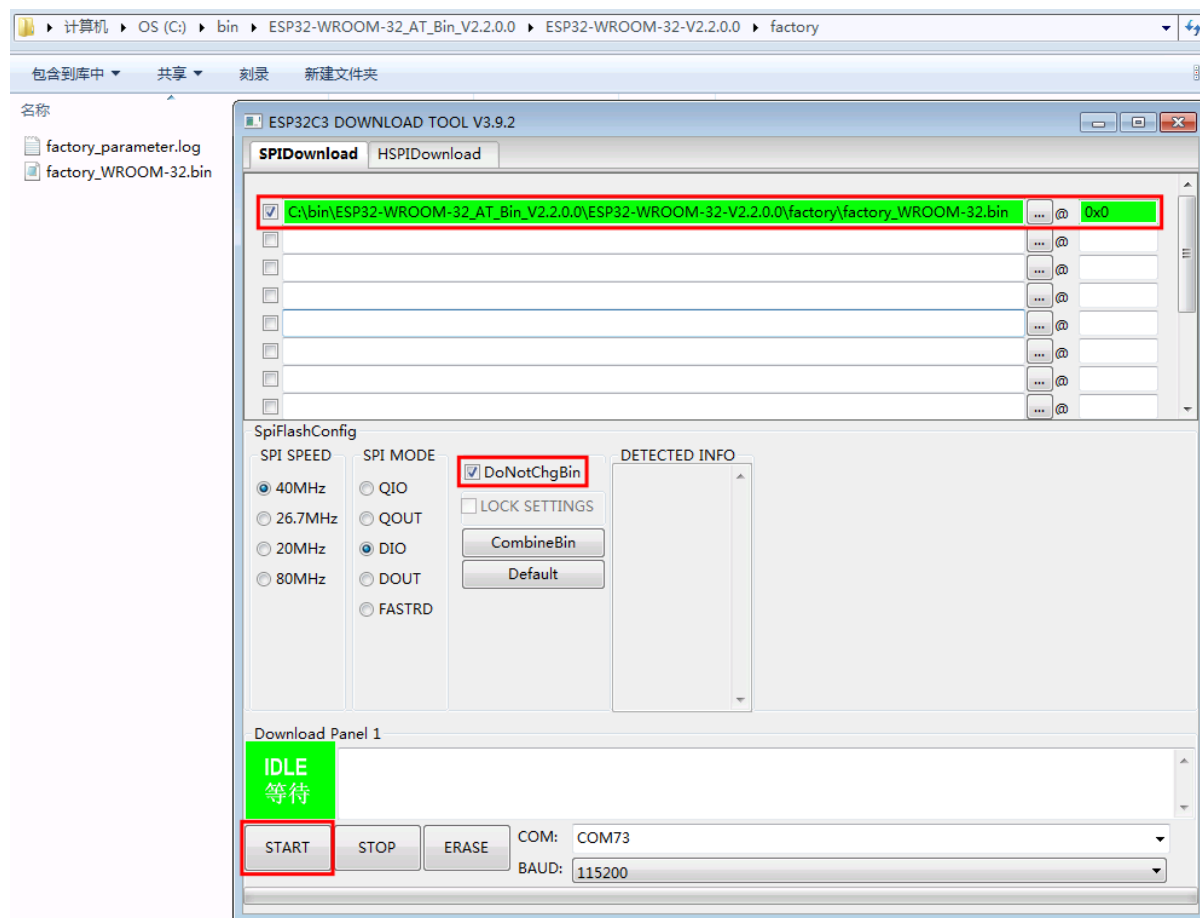


Fig. 9: Download to One Address (click to enlarge)

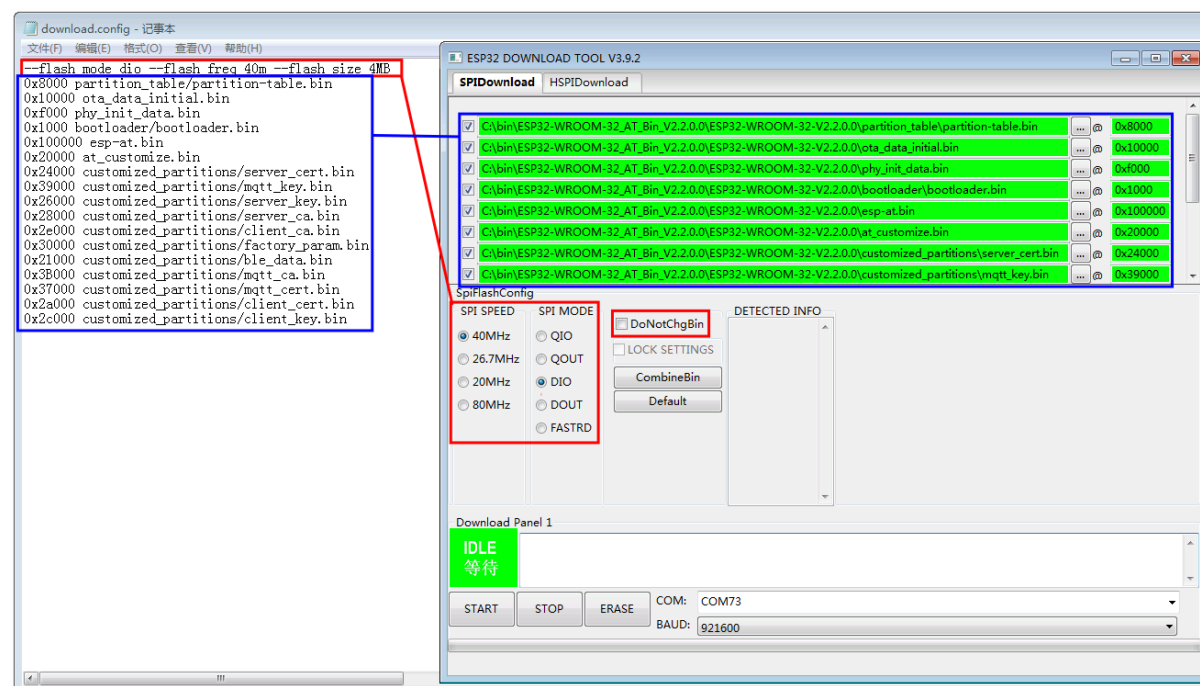


Fig. 10: Download to Multiple Addresses (click to enlarge)

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

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

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

When you finish flashing, please [Check Whether AT Works](#).

1.4.3 Check Whether AT Works

To check whether AT works, do as follows:

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

If the response is OK as shown below, AT works.

```
AT+GMR
AT version:3.2.0.0(s-ec2dec2 - ESP32 - Jul 28 2023 07:05:28)
SDK version:v5.0.2-376-g24b9d38a24-dirty
compile time(6118fc22):Jul 28 2023 09:47:28
Bin version:v3.2.0.0(WROOM-32)

OK
```

Otherwise, you need to check your ESP32 startup log in one of the following ways:

Method 1:

- Open a serial port tool, such as SecureCRT;
- Select the port attached to the “Download/Log output” line. For more information on this line, see [Hardware Connection](#).
- Set Baudrate to 115200;
- Set Data Bits to 8;
- Set Parity to None;
- Set Stop Bits to 1;
- Set Flow Type to None;
- Press the RST key of the board directly. If it is like the log below, it means that ESP-AT firmware have been initialized correctly.

Method 2:

- Open two serial port tools, such as SecureCRT;
- In one serial port tool, select the port attached to the “AT command/response” line. In the other tool, select the port attached to the “Download/Log output” line. For more information on these lines, see [Hardware Connection](#).
- Set Baudrate to 115200;
- Set Data Bits to 8;
- Set Parity to None;
- Set Stop Bits to 1;
- Set Flow Type to None;

- Enter the command **AT+RST** with a new line (CR LF) to the “AT command/response” line. If the serial log from the “Download/Output log” line is like the log below, it means that ESP-AT firmware have been initialized correctly.

ESP32 startup log:

```
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0030,len:5884
ho 0 tail 12 room 4
load:0x40078000,len:15844
load:0x40080400,len:3560
entry 0x40080604
I (29) boot: ESP-IDF v5.0-541-g885e501d99-dirty 2nd stage bootloader
I (29) boot: compile time 08:40:13
I (29) boot: chip revision: v1.0
I (34) boot.esp32: SPI Speed      : 40MHz
I (38) boot.esp32: SPI Mode      : DIO
I (43) boot.esp32: SPI Flash Size : 4MB
I (47) boot: Enabling RNG early entropy source...
I (53) boot: Partition Table:
I (56) boot: ## Label            Usage            Type ST Offset     Length
I (64) boot:  0 phy_init         RF data          01 01 0000f000 00001000
I (71) boot:  1 otadata          OTA data         01 00 00010000 00002000
I (78) boot:  2 nvs               WiFi data        01 02 00012000 0000e000
I (86) boot:  3 at_customize     unknown          40 00 00020000 000e0000
I (93) boot:  4 ota_0            OTA app          00 10 00100000 00180000
I (101) boot:  5 ota_1           OTA app          00 11 00280000 00180000
I (108) boot: End of partition table
I (113) esp_image: segment 0: paddr=00100020 vaddr=3f400020 size=1a854h (108628)
↪map
I (161) esp_image: segment 1: paddr=0011a87c vaddr=3ff80063 size=00008h (    8)
↪load
I (161) esp_image: segment 2: paddr=0011a88c vaddr=3ffbdb60 size=04d5ch ( 19804)
↪load
I (174) esp_image: segment 3: paddr=0011f5f0 vaddr=40080000 size=00a28h ( 2600)
↪load
I (176) esp_image: segment 4: paddr=00120020 vaddr=400d0020 size=11f5c0h (1177024)
↪map
I (609) esp_image: segment 5: paddr=0023f5e8 vaddr=40080a28 size=1e948h (125256)
↪load
I (660) esp_image: segment 6: paddr=0025df38 vaddr=400c0000 size=00064h (   100)
↪load
I (676) boot: Loaded app from partition at offset 0x100000
I (676) boot: Disabling RNG early entropy source...
no external 32k oscillator, disable it now.
at param mode: 1
AT cmd port:uart1 tx:17 rx:16 cts:15 rts:14 baudrate:115200
module_name: WROOM-32
max tx power=78, ret=0
2.5.0
```

To learn more about ESP-AT, please read [What is ESP-AT](#).

To get started with ESP-AT, please read [Technology Selection](#) first to help you select hardware and software, and then read [Hardware Connection](#) to learn what hardware to prepare and how to connect them. Then, you can download and flash AT firmware into your device according to [Downloading Guide](#).

Chapter 2

AT Binary Lists

2.1 Released Firmware

- [Introduction](#)
- [ESP32-WROOM-32 Series](#)
- [ESP32-MINI-1 Series](#)
- [ESP32-WROVER-32 Series](#)
- [ESP32-PICO Series](#)
- [ESP32-SOLO Series](#)
- [Subscribe to AT Releases](#)
- [AT Released Firmware Differences](#)

2.1.1 Introduction

Note:

- Before using the AT firmware below, please read the [Disclaimer](#) carefully and comply with its terms and precautions.
 - If you are unsure which AT firmware your module should use, please read the [AT Released Firmware Differences](#) section first. It compares the various ESP32 AT firmwares in terms of supported command sets, hardware configurations, and module compatibility, helping you determine which firmware best matches your module's hardware configuration.
 - If you need to modify the following configurations, you can generate a new firmware by modifying the AT firmware using the [at.py Tool](#).
 - [Modify UART Configuration](#)
 - [Modify Wi-Fi Configuration](#)
 - [Modify Certificate and Key Configuration](#)
 - [Modify GATTS Configuration](#)
-

2.1.2 ESP32-WROOM-32 Series

- [v4.1.1.0 ESP32-WROOM-32-AT-V4.1.1.0.zip](#) (Recommended)
- [v4.1.0.0 ESP32-WROOM-32-AT-V4.1.0.0.zip](#)
- [v3.4.0.0 ESP32-WROOM-32-AT-V3.4.0.0.zip](#)
- [v3.2.1.0 ESP32-WROOM-32-AT-V3.2.1.0.zip](#)
- [v3.2.0.0 ESP32-WROOM-32-AT-V3.2.0.0.zip](#)
- [v2.4.0.0 ESP32-WROOM-32-AT-V2.4.0.0.zip](#)
- [v2.2.0.0 ESP32-WROOM-32-AT-V2.2.0.0.zip](#)
- [v2.1.0.0 ESP32-WROOM-32-AT-V2.1.0.0.zip](#)
- [v2.0.0.0 ESP32-WROOM-32-AT-V2.0.0.0.zip](#)
- [v1.1.2.0 ESP32-WROOM-32-AT-V1.1.2.0.zip](#)
- [v1.1.1.0 ESP32-WROOM-32-AT-V1.1.1.0.zip](#)
- [v1.1.0.0 ESP32-WROOM-32-AT-V1.1.0.0.zip](#)
- [v1.0.0.0 ESP32-WROOM-32-AT-V1.0.0.0.zip](#)
- [v0.10.0.0 ESP32-WROOM-32-AT-V0.10.0.0.zip](#)

2.1.3 ESP32-MINI-1 Series

- [v4.1.1.0 ESP32-MINI-1-AT-V4.1.1.0.zip](#) (Recommended)
- [v4.1.0.0 ESP32-MINI-1-AT-V4.1.0.0.zip](#)
- [v3.4.0.0 ESP32-MINI-1-AT-V3.4.0.0.zip](#)
- [v3.2.1.0 ESP32-MINI-1-AT-V3.2.1.0.zip](#)
- [v3.2.0.0 ESP32-MINI-1-AT-V3.2.0.0.zip](#)
- [v2.4.0.0 ESP32-MINI-1-AT-V2.4.0.0.zip](#)
- [v2.2.0.0 ESP32-MINI-1-AT-V2.2.0.0.zip](#)

2.1.4 ESP32-WROVER-32 Series

Note:

- This series is no longer releasing new AT firmware. It is recommended to use the ESP32-MINI-1 series instead.
 - It is not recommended to use the ESP32-WROVER-B module due to hardware limit. Please use other WROVER series modules.
-

- [v2.4.0.0 ESP32-WROVER-32-AT-V2.4.0.0.zip](#)
- [v2.2.0.0 ESP32-WROVER-32-AT-V2.2.0.0.zip](#)
- [v2.1.0.0 ESP32-WROVER-32-AT-V2.1.0.0.zip](#)
- [v2.0.0.0 ESP32-WROVER-32-AT-V2.0.0.0.zip](#)
- [v0.10.0.0 ESP32-WROVER-32-AT-V0.10.0.0.zip](#)

2.1.5 ESP32-PICO Series

- [v4.1.1.0 ESP32-PICO-D4-AT-V4.1.1.0.zip](#) (Recommended)
- [v4.1.0.0 ESP32-PICO-D4-AT-V4.1.0.0.zip](#)
- [v3.4.0.0 ESP32-PICO-D4-AT-V3.4.0.0.zip](#)
- [v3.2.1.0 ESP32-PICO-D4-AT-V3.2.1.0.zip](#)
- [v3.2.0.0 ESP32-PICO-D4-AT-V3.2.0.0.zip](#)
- [v2.4.0.0 ESP32-PICO-D4-AT-V2.4.0.0.zip](#)
- [v2.2.0.0 ESP32-PICO-D4-AT-V2.2.0.0.zip](#)
- [v2.1.0.0 ESP32-PICO-D4-AT-V2.1.0.0.zip](#)
- [v2.0.0.0 ESP32-PICO-D4-AT-V2.0.0.0.zip](#)

2.1.6 ESP32-SOLO Series

- v4.1.1.0 [ESP32-SOLO-AT-V4.1.1.0.zip](#) (Recommended)
- v4.1.0.0 [ESP32-SOLO-AT-V4.1.0.0.zip](#)
- v3.4.0.0 [ESP32-SOLO-AT-V3.4.0.0.zip](#)
- v3.2.1.0 [ESP32-SOLO-AT-V3.2.1.0.zip](#)
- v3.2.0.0 [ESP32-SOLO-AT-V3.2.0.0.zip](#)
- v2.4.0.0 [ESP32-SOLO-AT-V2.4.0.0.zip](#)
- v2.2.0.0 [ESP32-SOLO-AT-V2.2.0.0.zip](#)
- v2.1.0.0 [ESP32-SOLO-AT-V2.1.0.0.zip](#)
- v2.0.0.0 [ESP32-SOLO-AT-V2.0.0.0.zip](#)

2.1.7 Subscribe to AT Releases

Please refer to the [Subscribe to AT Releases](#) documentation to subscribe to our version release notifications and stay updated on the latest releases.

2.1.8 AT Released Firmware Differences

This section compares the ESP32 AT released firmwares in supported command sets, hardware, and supported modules.

- ESP32-WROOM-32-AT-Vx.x.x.x.zip (referred to as **WROOM Bin** in this section);
- ESP32-WROVER-32-AT-Vx.x.x.x.zip (referred to as **WROVER Bin** in this section);
- ESP32-PICO-D4-AT-Vx.x.x.x.zip (referred to as **PICO-D4 Bin** in this section);
- ESP32-SOLO-1-AT-Vx.x.x.x.zip (referred to as **SOLO-1 Bin** in this section);
- ESP32-MINI-1-AT-Vx.x.x.x.zip (referred to as **MINI-1 Bin** in this section);
- ESP32-D2WD-AT-Vx.x.x.x.zip (referred to as **D2WD Bin** in this section);

Supported Command Set

The table lists which command set is supported by default in the official AT firmware applicable to ESP32 series of modules (marked with ) , which is not supported by default but can be supported after configuration of the ESP-AT project (marked with ) , and which is not supported at all (marked with ) . Note that the command set that is not shown in this table is not supported either. Applicable firmware that has not been *officially released* requires compilation by yourself. Those self-compiled firmware cannot be upgraded OTA from Espressif official server.

Command Set	WROOM Bin	WROVER Bin	PICO-D4 Bin	SOLO-1 Bin	MINI-1 Bin	D2WD Bin
base	✓	✓	✓	✓	✓	✓
user	✓	✓	✓	✓	✓	✓
Wi-Fi	✓	✓	✓	✓	✓	✓
TCP-IP	✓	✓	✓	✓	✓	✓
mDNS	✓	✓	✓	✓	✓	✓
WPS	✓	✓	✓	✓	✓	✓
SmartConfig	✓	✓	✓	✓	✓	✓
ping	✓	✓	✓	✓	✓	✓
MQTT	✓	✓	✓	✓	✓	✓
HTTP	✓	✓	✓	✓	✓	✓
Bluetooth LE	✓	✓	✓	✓	✓	✓
Bluetooth LE HID	✓	✓	✓	✓	✓	✓
BluFi	✓	✓	✓	✓	✓	✓
Bluetooth SPP	✓	✓	✓	✓	✓	✓
Bluetooth A2DP	✓	✓	✓	✓	✓	✓
ethernet	✓	✓	✓	✓	✓	✓
FileSystem	✓	✓	✓	✓	✓	✓
driver	✓	✓	✓	✓	✓	✓
WPA2 enterprise	✓	✓	✓	✓	✓	✓
Web server	✓	✓	✓	✓	✓	✓
WebSocket	✓	✓	✓	✓	✓	✓
OTA	✓	✓	✓	✓	✓	✗

Hardware Differences

Hardware	WROOM Bin	WROVER Bin	PICO-D4 Bin	SOLO-1 Bin	MINI-1 Bin	D2WD Bin
Flash	4 MB	4 MB	4 MB	4 MB	4 MB	2 MB
PSRAM	✗	8 MB	✗	✗	✗	✗
UART Pins ¹						
	TX: 17	TX: 22	TX: 22	TX: 17	TX: 22	TX: 22
	RX: 16	RX: 19	RX: 19	RX: 16	RX: 19	RX: 19
	CTS: 15	CTS: 15	CTS: 15	CTS: 15	CTS: 15	CTS: 15
	RTS: 14	RTS: 14	RTS: 14	RTS: 14	RTS: 14	RTS: 14

Supported Modules

The table below lists the modules or chips that are default supported by the officially released ESP32 series AT firmware (indicated by ✓), the modules that are not supported by default but can be modified to be supported using the [at.py Tool](#) (indicated by ✓), and the modules that are not supported at all (indicated by ✗). For modules that are not supported at all, you can refer to [Compile ESP-AT Project Locally](#) to modify the configuration as needed for support.

¹ UART pins can be customized. See [How to Set AT Port Pins](#) for details.

Module/Chip	WROOM Bin	WROVER Bin	PICO-D4 Bin	SOLO-1 Bin	MINI-1 Bin	D2WD Bin
ESP32-WROOM-32	✓	✗	✓	✓	✓	✓
ESP32-WROOM-32D	✓	✗	✓	✓	✓	✓
ESP32-WROOM-32E	✓	✗	✓	✓	✓	✓
ESP32-WROOM-32U	✓	✗	✓	✓	✓	✓
ESP32-WROOM-32UE	✓	✗	✓	✓	✓	✓
ESP32-WROOM-32DA	✓	✗	✓	✓	✓	✓
ESP32-WROOM-32SE	✓	✗	✓	✓	✓	✓
ESP32-WROVER-E	✓	✓	✓	✓	✓	✓
ESP32-WROVER-IE	✓	✓	✓	✓	✓	✓
ESP32-WROVER-B	✓	✓	✓	✓	✓	✓
ESP32-WROVER-IB	✓	✓	✓	✓	✓	✓
ESP32-WROVER	✓	✓	✓	✓	✓	✓
ESP32-SOLO-1	✓	✗	✓	✓	✓	✓
ESP32-D2WD	✗	✗	✗	✗	✗	✓
ESP32-MINI-1	✓	✗	✓	✓	✓	✓
ESP32-MINI-1U	✓	✗	✓	✓	✓	✓
ESP32-PICO-D4	✓	✗	✓	✓	✓	✓
ESP32-PICO-V3-ZERO	✗	✗	✗	✗	✗	✗
ESP32-PICO-MINI-02	✓	✗	✓	✓	✓	✓
ESP32-PICO-MINI-02U	✓	✗	✓	✓	✓	✓

This document covers the following sections:

- [Download ESP32 AT Released Firmware](#)
- [Brief Introduction to AT Firmware](#): What binary files the AT firmware contains and their functions.

Note: To download AT firmware for other chip series, please go to the drop-down list on the upper left corner of this page and select a chip series to navigate to the documentation of that chip for downloading.

Currently, there is no AT firmware available for the ESP32-H series, ESP32-P series, and ESP32-S3 series. It is recommended to use the ESP32-C series.

2.2 Brief Introduction to AT Firmware

The ESP-AT firmware package contains several binary files for specific functionalities:

```

build
├─ at_customize.bin          // Secondary partition table (user partition table,
└─ listing the start address and size of the mfg_nvs partition and fs_storage
└─ partition)
├─ bootloader
│   └─ bootloader.bin        // Bootloader
├─ customized_partitions
│   └─ mfg_nvs.bin           // Factory configuration parameters, parameter values
└─ are listed in the mfg_nvs.csv file in the same directory
├─ esp-at.bin                // AT application firmware
├─ factory
│   └─ factory_xxx.bin       // Collection of binary files for specific
└─ functionalities. You can burn only this file to the flash space with a starting

```

(continues on next page)

(continued from previous page)

```
↪address of 0, or burn several binary files to the flash space corresponding to↪
↪the starting address according to the information in the download.config file.
├─ partition_table
│   └─ partition-table.bin // Primary partition table (system partition table)
└─ ota_data_initial.bin    // OTA data initialization file
```

Chapter 3

AT Command Set

Here is a list of AT commands.

3.1 Basic AT Commands

- *Introduction*
- *AT*: Test AT startup.
- *AT+RST*: Restart a module.
- *AT+GMR*: Check version information.
- *AT+CMD*: List all AT commands and types supported in current firmware.
- *AT+GSLP*: Enter Deep-sleep mode.
- *ATE*: Configure AT commands echoing.
- *AT+RESTORE*: Restore factory default settings of the module.
- *AT+SAVETRANSLINK*: Set whether to enter Network or Bluetooth *Passthrough Mode* on power-up.
- *AT+TRANSINTVL*: Set the data transmission interval in the *Passthrough Mode*.
- *AT+UART_CUR*: Current UART configuration, not saved in flash.
- *AT+UART_DEF*: Default UART configuration, saved in flash.
- *AT+SLEEP*: Set the sleep mode.
- *AT+SYSRAM*: Query the heap memory status.
- *AT+SYMSG*: Query or set System Prompt Information.
- *AT+SYMSGFILTER*: Enable or disable the *system message* filter.
- *AT+SYMSGFILTERCFG*: Query or set the *system message* filters.
- *AT+SYSFLASH*: Query or set User Partitions in Flash.
- *AT+SYSMFG*: Query or set *manufacturing nvs* User Partitions.
- *AT+RFPOWER*: Query or set RF TX Power.
- *AT+RFCAL*: RF full calibration.
- *AT+SYSROLLBACK*: Roll back to the previous firmware.
- *AT+SYSTIMESTAMP*: Query or set local time stamp.
- *AT+SYSLOG*: Enable or disable the AT error code prompt.
- *AT+SLEEPWKCFG*: Query or set the light-sleep wakeup source and awake GPIO.
- *AT+SYSSTORE*: Query or set parameter store mode.
- *AT+SYSREG*: Read or write the register.

3.1.1 Introduction

Important: The default AT firmware supports all the AT commands mentioned on this page.

3.1.2 AT: Test AT Startup

Execute Command

Command:

```
AT
```

Response:

```
OK
```

3.1.3 AT+RST: Restart a Module

Execute Command

Command:

```
AT+RST
```

Response:

```
OK
```

3.1.4 AT+GMR: Check Version Information

Execute Command

Command:

```
AT+GMR
```

Response:

```
<AT version info>
<SDK version info>
<compile time>
Bin version:<Bin version>(<module_name>)

OK
```

Parameters

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

- **<Bin version>**: esp-at firmware version. Version information can be modified in menuconfig. `python build.py menuconfig>Application manager>Project version`. Maximum length: 32 bytes.
- **<module_name>**: the module name, which is defined in file: `esp-at/components/customized_partitions/raw_data/factory_param/factory_param_data.csv`.

Note

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

Example

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

OK
```

3.1.5 AT+CMD: List all AT commands and types supported in current firmware**Query Command****Command:**

```
AT+CMD?
```

Response:

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

OK
```

Parameters

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

3.1.6 AT+GSLP: Enter Deep-sleep Mode**Set Command****Command:**

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

Response:

```
<time>
```

```
OK
```

Parameter

- **<time>**: The duration when the device stays in Deep-sleep. Unit: millisecond. When the time is up, the device automatically wakes up, calls Deep-sleep wake stub, and then proceeds to load the application.
 - 0 means restarting right now
 - The maximum Deep-sleep time is about 28.8 days ($2^{31}-1$ milliseconds)

Notes

- The theoretical and actual time of Deep-sleep may be different due to external factors.

3.1.7 ATE: Configure AT Commands Echoing

Execute Command

Command:

```
ATE0
```

or

```
ATE1
```

Response:

```
OK
```

Parameters

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

3.1.8 AT+RESTORE: Restore Factory Default Settings

Execute Command

Command:

```
AT+RESTORE
```

Response:

```
OK
```

Notes

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

3.1.9 AT+SAVETRANSLINK: Set Whether to Enter Network or Bluetooth LE Passthrough Mode on Power-up

- *Set Whether to Enter TCP/SSL Client Passthrough Mode on Power-up*
- *Set Whether to Enter TCP/SSL Server Passthrough Mode on Power-up*
- *Set Whether to Enter UDP Passthrough Mode on Power-up*
- *Set Whether to Enter Bluetooth LE Passthrough Mode on Power-up*

Set Whether to Enter TCP/SSL Client Passthrough Mode on Power-up

Set Command Command:

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

Response:

```
OK
```

Parameters

- **<mode>**:
 - 0: ESP32 will NOT enter TCP/SSL client *Passthrough Mode* on power-up.
 - 1: ESP32 will enter TCP/SSL client *Passthrough Mode* on power-up.
- **<" remote host" >**: IPv4 address, IPv6 address, or domain name of remote host. The maximum length is 64 bytes.
- **<remote port>**: the remote port number.
- **<" type" >**: string parameter showing the type of transmission: "TCP" , "TCPv6" , "SSL" , or "SSLv6" . Default: "TCP" .
- **<keep_alive>**: It configures the `SO_KEEPALIVE` option for socket. Unit: second.
 - Range: [0,7200].
 - * 0: Disable keep-alive function (default).
 - * 1 ~ 7200: Enable keep-alive function. `TCP_KEEPIPLE` value is **<keep_alive>**, `TCP_KEEPIPLE` value is 1, and `TCP_KEEPCNT` value is 3.
 - This parameter of this command is the same as the **<keep_alive>** parameter of `AT+CIPTCPOPT` command. It always takes the value set later by either of the two commands. If it is omitted or not set, the last configured value is used by default.

Notes

- This command will save the *Passthrough Mode* configuration in the NVS partition. If **<mode>** is set to 1, ESP32 will enter the *Passthrough Mode* in the next power on. The configuration will take effect after ESP32 reboots.

Example

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

Set Whether to Enter TCP/SSL Server Passthrough Mode on Power-up

Set Command Command:

```
AT+SAVETRANSLINK=<mode>[,<port>][,<"type">][,<CA enable>][,<netif>]
```

Response:

```
OK
```

Parameters

- **<mode>**:
 - 0: ESP32 will NOT enter TCP/SSL server *Passthrough Mode* on power-up.
 - 3: ESP32 will enter TCP/SSL server *Passthrough Mode* on power-up.
- **<port>**: The port that the server listens to. Default: 333.
- **<"type">**: Server type: "TCP", "TCPv6", "SSL", or "SSLv6". Default: "TCP".
- **<CA enable>**: This parameter is only valid when <type> is "SSL" or "SSLv6".
 - 0: Do not use CA certificate.
 - 1: Use CA certificate.
- **<netif>**: Specifies the network interface for the server to listen on. Default: 0.
 - 0: All network interfaces
 - 1: Wi-Fi Station interface
 - 2: Wi-Fi SoftAP interface
 - 3: Ethernet interface

Notes

- This command will save the *Passthrough Mode* configuration in the NVS partition. If <mode> is set to 1, ESP32 will enter the *Passthrough Mode* in the next power on. The configuration will take effect after ESP32 reboots.
- When ESP32 works as a Wi-Fi Station, please configure the Wi-Fi connection information in advance and enable the reconnection mechanism; when it works as a SoftAP, please configure the SoftAP parameters in advance to ensure it can work normally after power-on.
- In the TCP/SSL server passthrough mode, only a single passthrough connection is supported: once a client establishes a passthrough connection, the server will enter passthrough mode and reject subsequent passthrough connection requests.
- When the underlying network interface (netif) is stopped or disconnected, existing connections will be closed, and the server will return to listening state, ready to accept new normal connections.
- After exiting passthrough mode, the TCP/SSL server will continue running, but in normal (non-passthrough) mode with a maximum of 1 concurrent connection.

Example

```
// Set Wi-Fi SoftAP information, SSID is esp-ap-001, no password, channel 11, open_
↪ encryption, maximum 3 connections
AT+CWSAP="esp-ap-001","",11,0,3

// Set to enter TCP server passthrough mode on power-up, listening on port 1002
AT+SAVETRANSLINK=3,1002,"TCP",2
```

(continues on next page)

(continued from previous page)

```
// Restart the module
AT+RST

// PC connects to esp-ap-001 and uses a TCP client tool to connect to ESP32's port_
↪1002
nc 192.168.4.1 1002

// Transmit data in passthrough mode
```

Set Whether to Enter UDP Passthrough Mode on Power-up

Set Command Command:

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

Response:

```
OK
```

Parameters

- **<mode>**:
 - 0: ESP32 will NOT enter UDP *Passthrough Mode* on power-up.
 - 1: ESP32 will enter UDP *Passthrough Mode* on power-up.
- **<" remote host" >**: IPv4 address, IPv6 address, or domain name of remote host. The maximum length is 64 bytes.
- **<remote port>**: the remote port number.
- **<" type" >**: string parameter showing the type of transmission: "UDP" or "UDPv6". Default: "TCP".
- **<local port>**: local port when UDP passthrough is enabled on power-up.

Notes

- This command will save the *Passthrough Mode* configuration in the NVS partition. If <mode> is set to 1, ESP32 will enter the *Passthrough Mode* in the next power on. The configuration will take effect after ESP32 reboots.
- To establish an UDP transmission based on an IPv6 network, do as follows:
 - Make sure that the AP supports IPv6
 - Set *AT+CIPV6=1*
 - Obtain an IPv6 address through the *AT+CWJAP* command
 - (Optional) Check whether ESP32 has obtained an IPv6 address using the *AT+CIPSTA?* command

Example

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

Set Whether to Enter Bluetooth LE Passthrough Mode on Power-up

Set Command Command:

```
AT+SAVETRANSLINK=<mode>,<role>,<tx_srv>,<tx_char>,<rx_srv>,<rx_char>,<"peer_addr">
```

Response:

OK

Parameters

- **<mode>**:
 - 0: ESP32 will NOT enter BLE *Passthrough Mode* on power-up.
 - 2: ESP32 will enter BLE *Passthrough Mode* on power-up.
- **<role>**:
 - 1: client role.
 - 2: server role.
- **<tx_srv>**: tx service' s index. It can be queried with command [AT+BLEGATTCPRIMSRV=<conn_index>](#) if AT works as GATTC role or with command [AT+BLEGATTSSRV?](#) if AT works as GATTS role.
- **<tx_char>**: tx characteristic' s index. It can be queried with command [AT+BLEGATTCHAR=<conn_index>,<srv_index>](#) if AT works as GATTC role or with command [AT+BLEGATTCHAR?](#) if AT works as GATTS role.
- **<rx_srv>**: rx service' s index. It can be queried with command [AT+BLEGATTCPRIMSRV=<conn_index>](#) if AT works as GATTC role or with command [AT+BLEGATTSSRV?](#) if AT works as GATTS role.
- **<rx_char>**: rx characteristic' s index. It can be queried with command [AT+BLEGATTCHAR=<conn_index>,<srv_index>](#) if AT works as GATTC role or with command [AT+BLEGATTCHAR?](#) if AT works as GATTS role.
- **<" peer_addr" >**: remote Bluetooth LE address.

Notes

- This command will save the BLE *Passthrough Mode* configuration in the NVS partition. If <mode> is set to 2, ESP32 will enter the Bluetooth LE *Passthrough Mode* in the next power on. The configuration will take effect after ESP32 reboots.

Example

```
AT+SAVETRANSLINK=2,2,1,7,1,5,"26:a2:11:22:33:88"
```

3.1.10 AT+TRANSINTVL: Set the Data Transmission Interval in Passthrough Mode

Query Command

Command:

```
AT+TRANSINTVL?
```

Response:

```
+TRANSINTVL:<interval>
```

OK

Set Command

Command:

```
AT+TRANSINTVL=<interval>
```

Response:

OK

Parameter

- **<interval>**: Data transmission interval. Unit: milliseconds. Default value: 20. Range: [0,1000].

Note

- In passthrough mode, if the data received by ESP32 reaches or exceeds 2920 bytes, the data will be immediately sent in chunks of 2920 bytes. Otherwise, it will wait for <interval> milliseconds before being sent.
- To optimize data transmission in cases where the data size is small and the data transmission interval is short, adjusting <interval> can be useful. A smaller <interval> reduces the delay in sending data to the protocol stack, but this may increase the number of times the protocol stack sends data to the network, thereby potentially decreasing the throughput performance to some extent.

Example

```
// Set to send immediately upon receiving data
AT+TRANSINTVL=0
```

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

Query Command

Command:

AT+UART_CUR?

Response:

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

Set Command

Command:

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

Response:

OK

Parameters

- **<baudrate>**: UART baud rate
 - For ESP32 devices, the supported range is 80 ~ 5000000.
- **<databits>**: data bits
 - 5: 5-bit data
 - 6: 6-bit data
 - 7: 7-bit data

- 8: 8-bit data
- **<stopbits>**: stop bits
 - 1: 1-bit stop bit
 - 2: 1.5-bit stop bit
 - 3: 2-bit stop bit
- **<parity>**: parity bit
 - 0: None
 - 1: Odd
 - 2: Even
- **<flow control>**: flow control
 - 0: Flow control is not enabled.
 - 1: Enable RTS.
 - 2: Enable CTS.
 - 3: Enable both RTS and CTS.

Notes

- The Query Command will return actual values of UART configuration parameters, which may have minor differences from the set value because of the clock division.
- The configuration changes will NOT be saved in flash.
- To use hardware flow control, you need to connect CTS/RTS pins of your ESP32. For more details, please refer to [Hardware Connection](#) or components/customized_partitions/raw_data/factory_param/factory_param_data.csv.

Example

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

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

Query Command

Command:

```
AT+UART_DEF?
```

Response:

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

```
OK
```

Set Command

Command:

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

Response:

```
OK
```

Parameters

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

Notes

- The configuration changes will be saved in the NVS partition, and will still be valid when the chip is powered on again.
- To use hardware flow control, you need to connect CTS/RTS pins of your ESP32. For more details, please refer to [Hardware Connection](#) or components/customized_partitions/raw_data/factory_param/factory_param_data.csv.

Example

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

3.1.13 AT+SLEEP: Set the Sleep Mode

Query Command

Command:

```
AT+SLEEP?
```

Response:

```
+SLEEP:<sleep mode>
```

```
OK
```

Set Command

Command:

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

Response:

```
OK
```

Parameter

- **<sleep mode>:**
 - 0: Disable the sleep mode.
 - 1: Wi-Fi Modem-sleep mode. The RF module will be periodically closed according to the AP's DTIM. This setting only takes effect when Wi-Fi mode is Station; in the Null Wi-Fi mode, the setting is allowed but will not take effect (for compatibility with older AT firmware); in other cases, it cannot be set.
 - 2: Light-sleep mode. This parameter cannot be set when the Wi-Fi mode is SoftAP or Station+SoftAP.
 - * Null Wi-Fi mode.
 - CPU will automatically sleep and RF will be closed.
 - * Only Wi-Fi mode.
 - CPU will automatically sleep and RF will be periodically closed according to listen interval set by [AT+CWJAP](#) or [AT+CWCONFIG](#).
 - Only Bluetooth mode.
 - * When Bluetooth LE is advertising, CPU will automatically sleep and RF will be periodically closed according to advertising interval of Bluetooth.
 - * When Bluetooth LE is connected, CPU will automatically sleep and RF will be periodically closed according to connection interval of Bluetooth.
 - Wi-Fi and Bluetooth coexistence mode.
 - * CPU will automatically sleep and RF will be periodically closed according to power management module.
 - 3: Wi-Fi Modem-sleep listen interval mode. The RF module will be periodically closed according to listen interval set by [AT+CWJAP](#) or [AT+CWCONFIG](#). This setting only takes effect when Wi-Fi mode is Station; in the Null Wi-Fi mode, the setting is allowed but will not take effect (for compatibility with older AT firmware); in other cases, it cannot be set.

Note

- Before setting the Light-sleep mode, it is recommended to set the wakeup source in advance through the command [AT+SLEEPWKCFCG](#), otherwise ESP32 can't wake up and will always be in sleep mode.
- After setting the Light-sleep mode, if the Light-sleep wakeup condition is not met, ESP32 will automatically enter the sleep mode. When the Light-sleep wakeup condition is met, ESP32 will automatically wake up from sleep mode.
- In only BLE mode, only BLE Light-sleep and BLE Modem-sleep modes are available. Use the `AT+SLEEP=2` command to enable BLE Light-sleep, and use the `AT+SLEEP=0` command to enable BLE Modem-sleep.
- For Light-sleep mode in BLE mode, users must ensure external 32KHz crystal oscillator, otherwise the Light-sleep mode will work in Modem-sleep mode.
- For more examples, please refer to [Sleep AT Examples](#).

Example

```
AT+SLEEP=0
```

3.1.14 AT+SYSRAM: Query the Heap Memory Status

Query Command

Command:

```
AT+SYSRAM?
```

Response:

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

Parameters

- **<remaining RAM size>**: Current available heap size. Unit: byte. When there is a memory leak, this value will gradually decrease.
- **<minimum heap size>**: Minimum available heap size recorded since the ESP32 was powered on. Unit: byte. When this value is less than or close to 10 KB, the Wi-Fi and BLE functions of ESP32 may be affected. This value only updates when a new, lower minimum is detected; it never increases.

Note

- When `<minimum heap size>` is less than or close to 10 KB, or you see logs like `alloc failed, size:<requested_size>, caps:<requested_caps>` on the [AT log port](#), consider optimizing memory:
 - Disable unnecessary AT features under `python build.py menuconfig>Component config>AT`.
 - Refer to the [Minimizing RAM Usage](#) documentation for relevant methods.

Example

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

Set Command**Function:**

Query the memory usage under given capabilities.

Command:

```
AT+SYSRAM=<caps>
```

Response:

```
+SYSRAM:<caps_largest_free_block_size>,<caps_free_size>,<caps_minimum_free_size>,  
↪<caps_total_size>  
OK
```

Parameters

- **<caps>**: Capability value. See [Different capability definitions](#) for details. Multiple values can be combined, e.g., `AT+SYSRAM=0x1800` represents `MALLOC_CAP_INTERNAL | MALLOC_CAP_DEFAULT`.
- **<caps_largest_free_block_size>**: The maximum size of a free block that can be allocated under the given caps. Unit: byte.
- **<caps_free_size>**: The total size of all free blocks under the given caps. Unit: byte.
- **<caps_minimum_free_size>**: The minimum total size of all free blocks under the given caps since power-on. Unit: byte.
- **<caps_total_size>**: The total memory size under the given caps. Unit: byte.

Note

- During system operation, if there is insufficient memory, the *AT log port* will output `alloc failed, size:<requested_size>, caps:<requested_caps>`. You can send `AT+SYSRAM=<requested_caps>` to check the memory usage under the current caps. The `<caps_largest_free_block_size>` determines whether a memory block of size `<requested_size>` can be allocated.

3.1.15 AT+SYMSG: Query or Set System Prompt Information

Query Command

Function:

Query the current system prompt information state.

Command:

```
AT+SYMSG?
```

Response:

```
+SYMSG:<state>
OK
```

Set Command

Function:

Configure system prompt information. If you need more fine-grained management of AT messages, please use the [AT+SYMSGFILTER](#) command.

Command:

```
AT+SYMSG=<state>
```

Response:

```
OK
```

Parameter

- **<state>**:
 - bit 0: Prompt information when quitting Network *Passthrough Mode*, Bluetooth LE SPP and Bluetooth SPP.

- * 0: Print no prompt information when quitting Network *Passthrough Mode*, Bluetooth LE SPP and Bluetooth SPP.
- * 1: Print +QUITT when quitting Network *Passthrough Mode*, Bluetooth LE SPP and Bluetooth SPP.
- bit 1: Connection prompt information type.
 - * 0: Use simple prompt information, such as XX, CONNECT.
 - * 1: Use detailed prompt information, such as +LINK_CONN:status_type,link_id,ip_type,terminal_type,remote_ip,remote_port,local_port.
- bit 2: Connection status prompt information for Network *Passthrough Mode*, Bluetooth LE SPP and Bluetooth SPP.
 - * 0: Print no prompt information.
 - * 1: Print one of the following prompt information when Wi-Fi, socket, Bluetooth LE or Bluetooth status is changed:

```

- "CONNECT\r\n" or the message prefixed with "+LINK_CONN:"
- "CLOSED\r\n"
- "WIFI CONNECTED\r\n"
- "WIFI GOT IP\r\n"
- "WIFI GOT IPv6 LL\r\n"
- "WIFI GOT IPv6 GL\r\n"
- "WIFI DISCONNECT\r\n"
- "+ETH_CONNECTED\r\n"
- "+ETH_DISCONNECTED\r\n"
- the message prefixed with "+ETH_GOT_IP:"
- the message prefixed with "+STA_CONNECTED:"
- the message prefixed with "+STA_DISCONNECTED:"
- the message prefixed with "+DIST_STA_IP:"
- the message prefixed with "+BLECONN:"
- the message prefixed with "+BLEDISCONN:"
- "+BLE_TT_MODE\r\n"
- in passthrough mode, when enabling notify or indicate, the
  ↳message prefixed with "+WRITE:" will be printed

```

Notes

- The configuration changes will be saved in the NVS partition if AT+SYSSTORE=1.
- If you set bit 0 to 1, it will prompt "+QUITT" when you quit *Passthrough Mode*.
- If you set bit 1 to 1, it will impact the information of command *AT+CIPSTART* and *AT+CIPSERVER*. It will supply "+LINK_CONN:status_type,link_id,ip_type,terminal_type,remote_ip,remote_port,local_port" instead of "XX,CONNECT" .

Example

```

// print no prompt info when quitting passthrough mode
// print detailed connection prompt info
// print no prompt info when the connection status is changed
AT+SYMSMSG=2

```

or

```

// In the transparent transmission mode, a prompt message will be printed when the
  ↳Wi-Fi, socket, Bluetooth LE or Bluetooth status changes
AT+SYMSMSG=4

```

3.1.16 AT+SYMSMSGFILTER: Enable or Disable the system message Filter

Query Command

Function:

Query the current *system message* filtering state.

Command:

```
AT+SYSMSGFILTER?
```

Response:

```
+SYSMSGFILTER:<enable>
OK
```

Set Command

Function:

Enable or disable the *system message* filter.

Command:

```
AT+SYSMSGFILTER=<enable>
```

Response:

```
OK
```

Parameter

- **<enable>:**
 - 0: Disable system message filtering. System default value. When disabled, system messages will not be filtered by the set filter.
 - 1: Enable system message filtering. When it is enabled, the data matching the regular expression will be filtered out by AT, and MCU will not receive it, whereas the unmatched data will be sent to the MCU as it is.

Notes

- Please use *AT+SYSMSGFILTERCFG* to set up system message filters. Then, use this command to enable the system message filtering to achieve more sophisticated system message management.
- Please use the *AT+SYSMSGFILTER=1* command with caution. It is recommended that you disable the system message filtering promptly after enabling it to prevent the over-filtering of AT system messages.
- Before entering the *Passthrough Mode*, it is strongly recommended to use the *AT+SYSMSGFILTER=0* command to disable system message filtering.
- If you are working on secondary development of AT project, please use the following APIs to transmit data via the AT command port.

```
// Data transmission via native AT command port. Data will not be filtered by
↪ AT+SYSMSGFILTER command, and AT will not actively wake up MCU before sending
↪ data (MCU wake-up function is set by AT+USERWKMCFG).
int32_t esp_at_port_write_data_without_filter(uint8_t data, int32_t len);

// Data transmission via AT command port with a filtering function. Data will be
↪ filtered by AT+SYSMSGFILTER command (if enabled), and AT will not actively wake
↪ up MCU before sending data (MCU wake-up function is set by AT+USERWKMCFG).
```

(continues on next page)

(continued from previous page)

```

↪command).
int32_t esp_at_port_write_data(uint8_t data, int32_t len);

// Data transmission via AT command port with wake-up MCU function. Data will not
↪be filtered by AT+SYMSGFILTER command, and AT will actively wake up MCU before
↪sending data (MCU wake-up function is set by AT+USERWKMUCCFG command).
int32_t esp_at_port_active_write_data_without_filter(uint8_t data, int32_t len);

// Data transmission via AT command port with wake-up MCU function and filtering
↪function. Data will be filtered by AT+SYMSGFILTER command (if enabled), and AT
↪will actively wake up MCU before sending data (MCU wake-up function is set by
↪AT+USERWKMUCCFG command).
int32_t esp_at_port_active_write_data(uint8_t data, int32_t len);

```

Example For detailed examples, refer to [system message filtering example](#).

3.1.17 AT+SYMSGFILTERCFG: Query or Set the system message Filters

- [Query the Filters](#)
- [Clear all the Filters](#)
- [Add a Filter](#)
- [Delete a Filter](#)

Query the Filters

Query Command **Command:**

```
AT+SYMSGFILTERCFG?
```

Response:

```

+SYMSGFILTERCFG:<index>,<"head_regexp">,<"tail_regexp">
OK

```

Parameters

- **<index>**: The index of filters.
- **<" head_regexp" >**: The regular expression of header.
- **<" tail_regexp" >**: The regular expression of tail.

Clear all the Filters

Set Command **Command:**

```
AT+SYMSGFILTERCFG=<operator>
```

Response:

```
OK
```

Parameter

- **<operator>**:
 - 0: Clear all the filters. After clearing, you can free some heap size occupied by the filters.

Example

```
// Clear all the filters
AT+SYMSGFILTERCFG=0
```

Add a Filter**Set Command Command:**

```
AT+SYMSGFILTERCFG=<operator>,<head_regexp_len>,<tail_regexp_len>[,<cflags>]
```

Response:

```
OK
```

```
>
```

The symbol > indicates that AT is ready for receiving regular expressions from AT command port. You should enter the head regular expression and the tail regular expression. When the length reaches the <head_regexp_len> + <tail_regexp_len> value, the regular expression integrity check starts.

If the regular expression integrity check fails or the addition of filter fails, AT returns:

```
ERROR
```

If the integrity of the regular expression is verified successfully and the filter is added successfully, AT returns:

```
OK
```

Parameters

- **<operator>**:
 - 1: Add a filter. A filter contains a header regular expression and a tail regular expression.
- **<head_regexp_len>**: The length of the header regular expression. Range: [0,64]. If it is set to 0, the matching of the regular expression in the header is ignored, and <tail_regexp_len> cannot be 0.
- **<tail_regexp_len>**: The length of the tail regular expression. Range: [0,64]. If it is set to 0, the matching of the regular expression in the tail is ignored, and <head_regexp_len> cannot be 0.
- **<cflags>**: Optional parameter, the compilation flags for regular expressions. Please refer to [cflags description](#). Default: 0.
 - bit 0: REG_EXTENDED, use POSIX Extended Regular Expression syntax.
 - bit 1: REG_ICASE, ignore case when matching.
 - bit 2: REG_NEWLINE, change the behavior of ^ and \$ to match the beginning and end of lines, rather than the beginning and end of the entire string.

Notes

- Please use this command to set up system message filters. Then, use [AT+SYMSGFILTER](#) to enable the system message filtering to achieve more sophisticated system message management.
- For more details about header and tail regular expression format, refer to [POSIX Basic Regular Expression \(BRE\)](#).
- In order to avoid [system message](#) (TX data of AT command port) being filtered incorrectly, it is **strongly recommended** that the header regular expression starts with ^ and the tail regular expression ends with \$.
- Only when the system message matches both the header regular expression and the tail regular expression **at the same time** is the system message filtered. After filtering, the data matching the regular expression will be filtered out by AT, and MCU will not receive it, whereas the unmatched data will be sent to the MCU as it is. For example: if data is received with a length of n bytes, and the header regular expression matches data[0] ~ data[i], and the tail regular expression matches data[j] ~ data[n-1], then the data received by MCU will be data[i+1] ~ data[j-1].

- When the system message matches one filter, it will not continue to match other filters.
- When the system message matches the filter, the system message will not be cached, that is, the previous system message and the current system message will not be combined for matching.
- For devices with large throughput, it is **strongly recommended** that you limit the number of filters and disable system message filtering using the `AT+SYMSGFILTER=0` command in time.

Example

```
// Set the filter to filter out the "WIFI CONNECTED" system message report
AT+SYMSGFILTERCFG=1,17,0
// After the command returns OK and >, enter "^WIFI CONNECTED\r\n" (Note: \r\n are
↳2 bytes, corresponding to 0D 0A in ASCII code)

// Enable system message filtering
AT+SYMSGFILTER=1

// Test filtering function
AT+CWMODE=1
AT+CWJAP="ssid", "password"
// AT no longer outputs "WIFI CONNECTED" system message report
```

For more examples of filtering system messages, refer to [system message filter example](#).

Delete a Filter

Set Command Command:

```
AT+SYMSGFILTERCFG=<operator>,<head_regexp_len>,<tail_regexp_len>[,<cflags>]
```

Response:

```
OK
>
```

The symbol > indicates that AT is ready for receiving regular expressions from AT command port. You should enter the head regular expression and the tail regular expression. When the length reaches the <head_regexp_len> + <tail_regexp_len> value, the regular expression integrity check starts.

If the regular expression integrity check fails or the addition of filter fails, AT returns:

```
ERROR
```

If the integrity of the regular expression is verified successfully and the filter is added successfully, AT returns:

```
OK
```

Parameters

- **<operator>**:
 - 2: Delete a filter.
- **<head_regexp_len>**: The length of the header regular expression. Range: [0,64]. If it is set to 0, the <tail_regexp_len> cannot be 0.
- **<tail_regexp_len>**: The length of the header regular expression. Range: [0,64]. If it is set to 0, the <head_regexp_len> cannot be 0.
- **<cflags>**: Optional parameter, the compilation flags for regular expressions. Please refer to [cflags description](#). Default: 0.
 - bit 0: REG_EXTENDED, use POSIX Extended Regular Expression syntax.
 - bit 1: REG_ICASE, ignore case when matching.

- bit 2: REG_NEWLINE, change the behavior of ^ and \$ to match the beginning and end of lines, rather than the beginning and end of the entire string.

Notes

- The filter to be deleted should be in the added filters.

Example

```
// Delete the filter added above
AT+SYMSGFILTERCFG=2,17,0
// After the command returns OK and >, enter "^WIFI CONNECTED\r\n" (Note: \r\n are
↪2 bytes, corresponding to 0D 0A in ASCII code)

// Test filtering function
AT+CWMODE=1
AT+CWJAP="ssid", "password"
// AT will output "WIFI CONNECTED" system message report again
```

3.1.18 AT+SYSFLASH: Query or Set User Partitions in Flash

Query Command

Function:

Query user partitions in flash.

Command:

```
AT+SYSFLASH?
```

Response:

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

Set Command

Function:

Read, write, and erase the user partitions in flash.

Command:

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

Response:

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

Parameters

- **<operation>**:
 - 0: Erase sector.
 - 1: Write data into the user partition.
 - 2: Read data from the user partition.
- **<partition>**: name of user partition

- **<offset>**: offset of user partition
- **<length>**: data length
- **<type>**: type of user partition
- **<subtype>**: subtype of user partition
- **<addr>**: address of user partition
- **<size>**: size of user partition

Notes

- Please make sure that you have downloaded `at_customize.bin` before using this command. For more details, please refer to [How to Customize Partitions](#).
- When erasing a partition, please erase the target partition in its entirety. This can be done by omitting the parameters `<offset>` and `<length>`. For example, command `AT+SYSFLASH=0, "mfg_nvs"` can erase the entire “mfg_nvs” user partition.
- The introduction to partitions is in [ESP-IDF Partition Tables](#).
- If the operator is `write`, wrap return `>` after the write command, then you can send the data that you want to write. The length should be parameter `<length>`.
- If the operator is `write`, please make sure that you have already erased this partition.
- If you want to modify some data in the “mfg_nvs” partition, please use the `AT+SYSMFG` command (key-value pairs operation). If you want to modify total “mfg_nvs” partition, please use the `AT+SYSFLASH` command (partition operation).
- When writing to a partition, the MCU should write data in multiple chunks to avoid memory exhaustion caused by writing too much data at once. For example, write 4 KB of data each time until the write is complete.

Example

```
// erase the "mfg_nvs" partition in its entirety.
AT+SYSFLASH=0, "mfg_nvs", 4096, 8192

// write a new "mfg_nvs" partition (size: 0x1C000) at offset 0 of the "mfg_nvs"
↪partition.
AT+SYSFLASH=1, "mfg_nvs", 0, 0x1C000
```

3.1.19 AT+SYSMFG: Query or Set manufacturing nvs User Partitions

Query Command

Function:

Query all namespaces of *manufacturing nvs* user partitions.

Command:

```
AT+SYSMFG?
```

Response:

```
+SYSMFG:<"namespace">
OK
```

Erase a namespace or key-value pair

Set Command Command:

```
AT+SYSMFG=<operation>,<"namespace">[,<"key">]
```

Response:

```
OK
```

Parameters

- **<operation>**:
 - 0: Erase operation
 - 1: Read operation
 - 2: Write operation
- **<" namespace" >**: namespace name.
- **<" key" >**: key name. If this parameter is omitted, all key-value pairs of current <"namespace"> will be erased. Otherwise, only the current key-value pair is erased.

Note

- Please refer to the [Non-Volatile Storage \(NVS\)](#) documentation to understand the concept of namespace and key-value pairs.

Example

```
// Erase all key-value pairs of client_cert namespace (That is, erase all client_
↪certificates)
AT+SYSMFG=0,"client_cert"

// Erase the client_cert.0 key-value pair of client_cert namespace (That is, erase_
↪the first client certificate)
AT+SYSMFG=0,"client_cert","client_cert.0"
```

Read a namespace or key-value pair**Set Command Command:**

```
AT+SYSMFG=<operation>[,<"namespace">][,<"key">][,<offset>,<length>]
```

Response:

When <"namespace"> and subsequent parameters are omitted, it returns:

```
+SYSMFG:<"namespace">
```

```
OK
```

When <"key"> and subsequent parameters are omitted, it returns:

```
+SYSMFG:<"namespace">,<"key">,<type>
```

```
OK
```

In other cases, it returns:

```
+SYSMFG:<"namespace">,<"key">,<type>,<length>,<value>
```

```
OK
```


Parameters

- **<operation>**:
 - 0: Erase operation
 - 1: Read operation
 - 2: Write operation
- **<" namespace" >**: Namespace name.
- **<" key" >**: key name.
- **<offset>**: The offset of the value.
- **<length>**: The length of the value.
- **<type>**: The type of the value.
 - 1: u8
 - 2: i8
 - 3: u16
 - 4: i16
 - 5: u32
 - 6: i32
 - 7: string
 - 8: binary
- **<value>**: The data of the value.

Note

- Please refer to the [Non-Volatile Storage \(NVS\)](#) documentation to understand the concept of namespace and key-value pairs.

Example

```
// Read all namespaces
AT+SYSFMFG=1

// Read all key-value pairs of client_cert namespace
AT+SYSFMFG=1,"client_cert"

// Read the value of client_cert.0 key in client_cert namespace
AT+SYSFMFG=1,"client_cert","client_cert.0"

// Read the value of client_cert.0 key in client_cert namespace, from offset: 100_
↳place, read 200 bytes
AT+SYSFMFG=1,"client_cert","client_cert.0",100,200
```

Write a key-value pair to a namespace

Set Command Command:

```
AT+SYSFMFG=<operation>,<"namespace">,<"key">,<type>,<value>
```

Response:

```
OK
```

Parameters

- **<operation>**:
 - 0: erase operation
 - 1: read operation
 - 2: write operation

- **<" namespace" >**: namespace name.
- **<" key" >**: key name.
- **<type>**: The type of the value.
 - 1: u8
 - 2: i8
 - 3: u16
 - 4: i16
 - 5: u32
 - 6: i32
 - 7: string
 - 8: binary
- **<value>**: It means differently depending on the parameter **<type>**:
 - If **<type>** is between 1-6, **<value>** represents the real value.
 - If **<type>** is between 7-8, **<value>** represents the length of the value. After you send the command, AT will return **>**. This symbol indicates that AT is ready for receiving data. You should enter the data of designated length. When the data length reaches the **<value>** value, the key-value pair will be written to the namespace immediately.

Note

- Please refer to the [Non-Volatile Storage \(NVS\)](#) documentation to understand the concept of namespace and key-value pairs.
- Before writing, you do not need to manually erase namespaces or key-value pairs (NVS will automatically erase key-value pairs as needed).
- If you want to modify some data in the “mfg_nvs” partition, please use the [AT+SYSMFG](#) command (key-value pairs operation). If you want to modify total “mfg_nvs” partition, please use the [AT+SYSFLASH](#) command (partition operation).
- For more examples of using the AT+SYSMFG command, please refer to [AT+SYSMFG Command Examples](#).
- When modifying configurations, this command does not perform functional validation on the configuration values. Even if you configure incorrect values, they will be stored. Therefore, you need to ensure the correctness of the configuration data yourself.

Example

```
// Write a new value for client_cert.0 key into client_cert namespace (That is, ↵
↵update the 0th client certificate)
AT+SYSMFG=2,"client_cert","client_cert.0",8,1164

// Wait until AT command port returns ``>``, and then write 1164 bytes
```

3.1.20 AT+RFPOWER: Query or Set RF TX Power

Query Command

Function:

Query the RF TX Power.

Command:

```
AT+RFPOWER?
```

Response:

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

Set Command

Command:

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

Response:

```
OK
```

Parameters

- **<wifi_power>**: the unit is 0.25 dBm. For example, if you set the value to 78, the actual maximum RF Power value is $78 * 0.25 \text{ dBm} = 19.5 \text{ dBm}$. After you configure it, please confirm the actual value by entering the command AT+RFPOWER?.
 - For ESP32 devices, the range is [40,84]:

set value	get value	actual value	actual dBm
[40,43]	34	34	8.5
[44,51]	44	44	11
[52,55]	52	52	13
[56,59]	56	56	14
[60,65]	60	60	15
[66,71]	66	66	16.5
[72,77]	72	72	18
[78,84]	78	78	19.5

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

3.1.21 Note

- When Wi-Fi is turned off or not initialized, the [AT+RFPOWER](#) command cannot set or query the RF TX Power of Wi-Fi. Similarly, when Bluetooth LE is not initialized, the command cannot set or query that of Bluetooth LE, either.
- Since the RF TX Power is actually divided into several levels, and each level has its own value range, the wifi_power value queried by the esp_wifi_get_max_tx_power may differ from the value set by esp_wifi_set_max_tx_power and is no larger than the set value.
- It is recommended to set the two parameters <ble_scan_power> and <ble_conn_power> to the same value as the <ble_adv_power> parameter. Otherwise, they will be automatically adjusted to the value of <ble_adv_power>.

3.1.22 AT: RF Full Calibration

Execute Command

Command:

```
AT+RFCAL
```

Response:

```
OK
```

3.1.23 Note

- ESP32 will automatically perform RF full calibration on the first startup, and partial calibration on subsequent startups. For more details, please refer to [RF Calibration](#).
- It is recommended to perform RF full calibration after firmware upgrade, changes to the device environment, or prolonged periods of device inactivity.

3.1.24 AT+SYSROLLBACK: Roll Back to the Previous Firmware

Query Command**Function:**

Query the address and version of the current running firmware and the rollback firmware.

Command:

```
AT+SYSROLLBACK?
```

Response:

```
+SYSROLLBACK:<running_app_addr>,<"running_app_version">,<rollback_app_addr>,<  
↪"rollback_app_version">  
OK
```

Execute Command**Function:**

Roll back to the previous firmware.

Command:

```
AT+SYSROLLBACK
```

Response:

```
OK
```

Parameters

- **<running_app_addr>**: the address of the current running firmware.
- **<" running_app_version" >**: the version of the current running firmware.
- **<rollback_app_addr>**: the address of the rollback firmware.
- **<" rollback_app_version" >**: the version of the rollback firmware.

Note

- When performing a firmware rollback, ESP32 first verifies the integrity of the target firmware. The rollback proceeds only if verification succeeds; otherwise, an error is returned.
- The rollback function operates independently of the OTA upgrade process and can directly switch execution to the firmware version stored in the other OTA partition.

3.1.25 AT+SYSTIMESTAMP: Query or Set Local Time Stamp

Query Command**Function:**

Query the time stamp.

Command:

```
AT+SYSTIMESTAMP?
```

Response:

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

Set Command**Function:**

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

Command:

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

Response:

```
OK
```

Parameter

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

Example

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

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

Query Command**Function:**

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

Command:

```
AT+SYSLOG?
```

Response:

```
+SYSLOG:<status>

OK
```

Set Command**Function:**

Enable or disable the AT error code prompt.

Command:

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

Response:

```
OK
```

Parameter

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

Example

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

OK
AT+FAKE
ERR CODE:0x01090000

ERROR
```

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

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

ERROR
```

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

category	subcategory	extension
bit 32 – bit 24	bit 23 – bit 16	bit 15 – bit 0

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

Table 1: Subcategory of Error Code

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

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

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

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

Set Command

Command:

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

Response:

```
OK
```

Parameters

- **<wakeup source>:**
 - 0: Reserved, not supported now.
 - 1: Reserved, not supported now.
 - 2: Wakeup by GPIO.
- **<param1>:**
 - If the wakeup source is GPIO, it means the GPIO number.
 - Reserved, not supported now.
- **<param2>:**
 - If the wakeup source is GPIO, it means the wakeup level:
 - * 0: Low level.
 - * 1: High level.

Example

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

Note

- The wake-up pin must be driven to a valid logic level and should not be left floating. It must be either high or low.

3.1.28 AT+SYSSTORE: Query or Set Parameter Store Mode**Query Command****Function:**

Query the AT parameter store mode.

Command:

```
AT+SYSSTORE?
```

Response:

```
+SYSSTORE:<store_mode>
OK
```

Set Command**Command:**

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

Response:

```
OK
```

Parameter

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

Note

- This command affects set commands only. Query commands are always fetched from RAM.
- Affected commands:

- [AT+SYMSMSG](#)
- [AT+CWMODE](#)
- [AT+CWBANDWIDTH](#)
- [AT+CIPV6](#)
- [AT+CWCONFIG](#)

- *AT+CWIAP*
- *AT+CWSAP*
- *AT+CWRECONNCFG*
- *AT+CIPAP*
- *AT+CIPSTA*
- *AT+CIPAPMAC*
- *AT+CIPSTAMAC*
- *AT+CIPDNS*
- *AT+CIPSSLCONF*
- *AT+CIPRECONNINTV*
- *AT+CIPTCPOPT*
- *AT+CWDHCPS*
- *AT+CWDHCP*
- *AT+CWSTAPROTO*
- *AT+CWAPPROTO*
- *AT+CWJEAP*
- *AT+CIPETH*
- *AT+CIPETHMAC*
- *AT+BLENAM*
- *AT+BTNAME*
- *AT+BLEADVPARAM*
- *AT+BLEADVDATA*
- *AT+BLEADVDATAEX*
- *AT+BLESCANRSPDATA*
- *AT+BLESCANPARAM*
- *AT+BTSCANMODE*

Examples

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

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

3.1.29 AT+SYSREG: Read or Write the Register

Set Command

Command:

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

Response:

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

Parameters

- **<direct>**: read or write register.
 - 0: Read register.
 - 1: Write register.

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

Note

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

3.2 Wi-Fi AT Commands

- *Introduction*
- *AT+CWINIT*: Initialize or deinitialize Wi-Fi driver.
- *AT+CWMODE*: Set the Wi-Fi mode (Station/SoftAP/Station+SoftAP).
- *AT+CWBANDWIDTH*: Query or set Wi-Fi bandwidth.
- *AT+CWSTATE*: Query the Wi-Fi state and Wi-Fi information.
- *AT+CWCONFIG*: Query or set Wi-Fi inactive time and listen interval time.
- *AT+CWLAP*: Connect to an AP.
- *AT+CWRECONNCFG*: Query or set the Wi-Fi reconnecting configuration.
- *AT+CWLAPOPT*: Set the configuration for the command *AT+CWLAP*.
- *AT+CWLAP*: List available APs.
- *AT+CWLQAP*: Disconnect from an AP.
- *AT+CWSAP*: Query or set the configuration of an ESP32 SoftAP.
- *AT+CWLIF*: Obtain IP address of the station that connects to an ESP32 SoftAP.
- *AT+CWLQIF*: Disconnect stations from an ESP32 SoftAP.
- *AT+CWDHCP*: Enable or disable DHCP.
- *AT+CWDHCP*: Query or set the IPv4 addresses allocated by an ESP32 SoftAP DHCP server.
- *AT+CWAUTOCONN*: Connect to an AP automatically when powered on.
- *AT+CWAPPROTO*: Query or set the Wi-Fi protocol standard of SoftAP mode.
- *AT+CWSTAPROTO*: Query or set the Wi-Fi protocol standard of station mode.
- *AT+CIPSTAMAC*: Query or set the MAC address of an ESP32 station.
- *AT+CIPAPMAC*: Query or set the MAC address of an ESP32 SoftAP.
- *AT+CIPSTA*: Query or set the IP address of an ESP32 station.
- *AT+CIPAP*: Query or set the IP address of an ESP32 SoftAP.
- *AT+CWSTARTSMART*: Start SmartConfig.
- *AT+CWSTOPSMART*: Stop SmartConfig.
- *AT+WPS*: Enable the WPS function.
- *AT+CWJEAP*: Connect to a WPA2 Enterprise AP.
- *AT+CWHOSTNAME*: Query or set the host name of an ESP32 station.
- *AT+CWCOUNTRY*: Query or set the Wi-Fi Country Code.

3.2.1 Introduction

Important: The default AT firmware supports all the AT commands except *AT+CWJEAP* mentioned on this page. If you need to modify the commands supported by ESP32 by default, please compile the ESP-AT project by following the steps in *Compile ESP-AT Project Locally* documentation. In the project configuration during the fifth step, make the following selections (Each item below is independent. Choose it according to your needs):

- Enable EAP commands (*AT+CWJEAP*): Component config>AT>AT WPA2 Enterprise command support
- Disable WPS commands (*AT+WPS*): Component config>AT>AT WPS command support
- Disable smartconfig commands (*AT+CWSTARTSMART* and *AT+CWSTOPSMART*): Component config>AT>AT smartconfig command support

- Disable all Wi-Fi commands (Not recommended. Once disabled, all Wi-Fi and above functions will be unusable, and you will need to implement these AT commands yourself): `Component config > AT > AT wifi command support`
-

3.2.2 AT+CWINIT: Initialize or Deinitialize Wi-Fi Driver

Query Command

Function:

Query the Wi-Fi initialization status of ESP32 device.

Command:

```
AT+CWINIT?
```

Response:

```
+CWINIT:<init>
OK
```

Set Command

Function:

Initialize or deinitialize Wi-Fi driver of ESP32 device.

Command:

```
AT+CWINIT=<init>
```

Response:

```
OK
```

Parameters

- **<init>:**
 - 0: Deinitialize Wi-Fi driver of ESP32 device.
 - 1: Initialize Wi-Fi driver of ESP32 device. (Default value)

Note

- This setting is not saved to flash and will revert to the default value of 1 after restarting.
- When you run out of RAM resources and Wi-Fi is not used, you can use this command to clean up the Wi-Fi driver to free up RAM resources.

Example

```
// Deinitialize Wi-Fi driver
AT+CWINIT=0
```

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

Query Command

Function:

Query the Wi-Fi mode of ESP32.

Command:

```
AT+CWMODE?
```

Response:

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

Set Command

Function:

Set the Wi-Fi mode of ESP32.

Command:

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

Response:

```
OK
```

Parameters

- **<mode>**:
 - 0: Null mode. Wi-Fi RF will be disabled.
 - 1: Station mode.
 - 2: SoftAP mode.
 - 3: SoftAP+Station mode.
- **<auto_connect>**: Enable or disable automatic connection to an AP when you change the mode of the ESP32 from the SoftAP mode or null mode to the station mode or the SoftAP+Station mode. Default: 1. If you omit the parameter, the default value will be used, i.e. automatically connecting to an AP.
 - 0: The ESP32 will not automatically connect to an AP.
 - 1: The ESP32 will automatically connect to an AP if the configuration to connect to the AP has already been saved in flash before.

Note

- The configuration changes will be saved in the NVS partition if [AT+SYSSSTORE=1](#).
- If you have previously used the Bluetooth function, it is recommended to send the following commands to deinitialize the initialized functions before using the SoftAP or SoftAP+Station functions:
 - [AT+BTINIT=0](#) (Deinitialize Classic Bluetooth)
 - [AT+BLEINIT=0](#) (Deinitialize Bluetooth LE)
 - [AT+BLUFI=0](#) (Disable BluFi)

For more details, please refer to the [RF Coexistence](#) documentation.

Example

```
AT+CWMODE=3
```

3.2.4 AT+CWBANDWIDTH: Query or Set Wi-Fi Bandwidth

Query Command

Command:

```
AT+CWBANDWIDTH?
```

Response:

```
+CWBANDWIDTH:<netif>,<bandwidth_2ghz>,<bandwidth_5ghz>
```

```
OK
```

Set Command

Command:

```
AT+CWBANDWIDTH=<netif>,<bandwidth_2ghz>[,<bandwidth_5ghz>]
```

Response:

```
OK
```

Parameters

- **<netif>**:
 - 0: Station interface
 - 1: SoftAP interface
- **<bandwidth_2ghz>**: 2.4 GHz bandwidth
 - 0: When used in a set command, it means the default value is kept. When used in a query command, it indicates that the current value is invalid
 - 1: 20 MHz
 - 2: 40 MHz
- **<bandwidth_5ghz>**: 5 GHz bandwidth
 - 0: When used in a set command, it means the default value is kept. When used in a query command, it indicates that the current value is invalid
 - 1: 20 MHz
 - 2: 40 MHz

Notes

- If *AT+SYSSSTORE=1*, this setting will be saved in the NVS partition.
- By default, the Wi-Fi bandwidth of ESP32 devices is 40 MHz.
- Note: Wi-Fi bandwidth settings are limited by hardware capabilities and Wi-Fi protocol standards.

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

Query Command

Function:

Query the Wi-Fi state and Wi-Fi information of ESP32.

Command:

```
AT+CWSTATE?
```

Response:

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

```
OK
```

Parameters

- **<state>**: current Wi-Fi state.
 - 0: ESP32 station has not started any Wi-Fi connection.
 - 1: ESP32 station has connected to an AP, but does not get an IPv4 address yet.
 - 2: ESP32 station has connected to an AP, and got an IPv4 address.
 - 3: ESP32 station is in Wi-Fi connecting or reconnecting state.
 - 4: ESP32 station is in Wi-Fi disconnected state.
- **<"ssid">**: the SSID of the target AP.

Note

- When ESP32 station is not connected to an AP, it is recommended to use this command to query Wi-Fi information; after ESP32 station is connected to an AP, it is recommended to use [AT+CWJAP](#) to query Wi-Fi information.

3.2.6 AT+CWCONFIG: Query or Set Wi-Fi Inactive Time and Listen Interval Time

Query Command

Command:

```
AT+CWCONFIG?
```

Response:

```
+CWCONFIG:<ap_inactive_time>,<sta_inactive_time>,<listen_interval>
```

```
OK
```

Set Command

Command:

```
AT+CWCONFIG=[<ap_inactive_time>][,<sta_inactive_time>][,<listen_interval>]
```

Response:

OK

Parameters

- **<ap_inactive_time>**: Inactive time in SoftAP mode. Unit: seconds. Default: 300. Range: [10,3600]. If the SoftAP does not receive any data from a connected station within the inactive time, the SoftAP will forcibly disconnect that Station's Wi-Fi connection.
- **<sta_inactive_time>**: Inactive time in Station mode. Unit: seconds. Default: 6. Range: [3,600]. If the Station does not receive any beacon frames from the connected SoftAP within the inactive time, the Wi-Fi connection of ESP32 will be forcibly disconnected.
- **<listen_interval>**: Interval for listening to AP beacons, in units of AP beacon intervals. Default: 3. Range: [1,100]. This parameter is the same as **<listen_interval>** in [AT+CWJAP](#), but this configuration is global.

Note

- The configuration changes will be saved in the NVS partition if [AT+SYSSTORE=1](#).

3.2.7 AT+CWJAP: Connect to an AP

Query Command

Function:

Query the AP to which the ESP32 Station is already connected.

Command:

AT+CWJAP?

Response:

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

Set Command

Function:

Connect an ESP32 station to a targeted AP.

Command:

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

Response:

```
WIFI CONNECTED
WIFI GOT IP

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

or

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

Execute Command

Function:

Connect an ESP32 station to a targeted AP with last Wi-Fi configuration.

Command:

```
AT+CWJAP
```

Response:

```
WIFI CONNECTED
WIFI GOT IP

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

or

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

Parameters

- **<"ssid">**: the SSID of the target AP.
 - Escape character syntax is needed if SSID or password contains special characters, such as , , " , or \.
 - Chinese SSID is supported. Chinese SSID of some routers or hotspots is not encoded in UTF-8 encoding format. You can scan SSID first, and then connect using the scanned SSID.
- **<"pwd">**: password, MAX: 64-byte ASCII.
- **<"bssid">**: the MAC address of the target AP. It cannot be omitted when multiple APs have the same SSID.
- **<channel>**: channel.
- **<rssi>**: signal strength.
- **<authmode>**: Wi-Fi authentication mode threshold. ESP-AT only connects to APs with an authentication mode equal to or higher than this threshold.
 - 0: OPEN (when <"pwd"> is not set or is empty) or WEP (when <"pwd"> length is greater than or equal to 8 bytes)
 - 1: WPA_PSK
 - 2: WPA_PSK
 - 3: WPA2_PSK
 - 4: WPA_WPA2_PSK
 - 5: WPA2_ENTERPRISE
 - 6: WPA3_PSK
 - 7: WPA2_WPA3_PSK
 - 8: WAPI_PSK
 - 9: OWE
 - 10: WPA3_ENT_192
 - 11: WPA3_EXT_PSK
 - 12: WPA3_EXT_PSK_MIXED_MODE
 - 13: DPP
 - 14: WPA3_ENTERPRISE
 - 15: WPA2_WPA3_ENTERPRISE
 - 16: WPA_ENTERPRISE
- **<reconn_interval>**: the interval between Wi-Fi reconnections. Unit: second. Default: 1. Maximum: 7200.

- 0: The ESP32 station will not reconnect to the AP when disconnected.
- [1,7200]: The ESP32 station will reconnect to the AP at the specified interval when disconnected.
- **<listen_interval>**: the interval of listening to the AP's beacon. Unit: AP beacon intervals. Default: 3. Range: [1,100].
- **<scan_mode>**:
 - 0: Fast scan. It will end after finding the targeted AP. The ESP32 station will connect to the first scanned AP.
 - 1: All-channel scan. It will end after all the channels are scanned. The device will connect to the scanned AP with the strongest signal.
- **<jap_timeout>**: maximum timeout for [AT+CWJAP](#) command. Unit: second. Default: 15. Range: [3,600].
- **<pmf>**: Protected Management Frames. Default: 1.
 - 0 means disable PMF.
 - bit 0: PMF capable, advertises support for protected management frame. Device will prefer to connect in PMF mode if other device also advertises PMF capability.
 - bit 1: PMF required, advertises that protected management frame is required. Device will not associate to non-PMF capable devices.
- **<error code>**: (for reference only)
 - 1: Connection timeout.
 - 2: Wrong password.
 - 3: Cannot find the target AP.
 - 4: Connection failed.
 - others: Unknown error occurred.

Notes

- The configuration changes will be saved in the NVS partition if [AT+SYSSSTORE=1](#).
- This command requires Station mode to be enabled.
- After ESP32 station is connected to an AP, it is recommended to use this command to query Wi-Fi information; when ESP32 station is not connected to an AP, it is recommended to use [AT+CWSTATE](#) to query Wi-Fi information.
- The parameter `<reconn_interval>` of this command is the same as `<interval_second>` of the command [AT+CWRECONNCFG](#). Therefore, if you omit `<reconn_interval>` when running this command, the interval between Wi-Fi reconnections will use the default value 1.
- If the `<"ssid">` and `<"password">` parameter are omitted, AT will use the last configuration.
- Execute command has the same maximum timeout to setup command. The default value is 15 seconds, but you can change it by setting the parameter `<jap_timeout>`.
- The authentication method via [WAPI](#) is not supported for connecting to the router.
- To get an IPv6 address, you need to set [AT+CIPV6=1](#).
- Response OK means that the IPv4 network is ready, but not the IPv6 network. At present, ESP-AT is mainly based on IPv4 network, supplemented by IPv6 network.
- WIFI GOT IPV6 LL represents that the linklocal IPv6 address has been obtained. This address is calculated locally through EUI-64 and does not require the participation of the AP. Because of the parallel timing, this print may be before or after OK.
- WIFI GOT IPV6 GL represents that the global IPv6 address has been obtained. This address is combined by the prefix issued by AP and the suffix calculated internally, which requires the participation of the AP. Because of the parallel timing, this print may be before or after OK, or it may not be printed because the AP does not support IPv6.

Example

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

// If the target AP's SSID is "ab\,c" and the password is "0123456789"\", the
↪command should be:
```

(continues on next page)

(continued from previous page)

```

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

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

```

3.2.8 AT+CWRECONNCFG: Query or Set the Wi-Fi Reconnecting Configuration

Query Command

Function:

Query the configuration of Wi-Fi reconnect.

Command:

```
AT+CWRECONNCFG?
```

Response:

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

Set Command

Function:

Set the configuration of Wi-Fi reconnect.

Command:

```
AT+CWRECONNCFG=<interval_second>,<repeat_count>[,<reconnect_now>]
```

Response:

```
OK
```

Parameters

- **<interval_second>**: the interval between Wi-Fi reconnections. Unit: second. Default: 0. Maximum: 7200.
 - 0: The ESP32 station will not reconnect to the AP when disconnected.
 - [1,7200]: The ESP32 station will reconnect to the AP at the specified interval when disconnected.
- **<repeat_count>**: the number of attempts the ESP32 makes to reconnect to the AP. This parameter only works when the parameter <interval_second> is not 0. Default: 0. Maximum: 1000.
 - 0: The ESP32 station will always try to reconnect to AP.
 - [1,1000]: The ESP32 station will attempt to reconnect to AP for the specified times.
- **<reconnect_now>**: whether to start reconnecting to the AP immediately after the setting is complete. Default: 0. Note: Reconnection only occurs if the ESP32's Wi-Fi has not yet obtained an IP address; if an IP address has already been obtained, the current connection state will remain unchanged.
 - 0: Do not reconnect to the AP immediately.
 - 1: Reconnect to the AP immediately.

Example

```
// The ESP32 station tries to reconnect to AP at the interval of one second for
↪100 times.
AT+CWRECONNCFG=1,100

// The ESP32 station will not reconnect to AP when disconnected.
AT+CWRECONNCFG=0,0

// Start reconnecting to the AP immediately after the command is set.
AT+CWRECONNCFG=2,50,1
```

Notes

- The configuration changes will be saved in the NVS partition if [AT+SYSSTORE=1](#).
- The parameter `<interval_second>` of this command is the same as the parameter `[<reconn_interval>]` of the command [AT+CWJAP](#).
- This command works for passive disconnection from APs, Wi-Fi mode switch, and Wi-Fi auto connect after power on.
- It is recommended to execute this command after a successful Wi-Fi connection.

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

Set Command

Command:

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

Response:

```
OK
```

or

```
ERROR
```

Parameters

- **<reserved>**: reserved item.
- **<print mask>**: determine whether the following parameters are shown in the result of [AT+CWLAP](#). Default: 0x7FF. If you set them to 1, it means showing the corresponding parameters; if you set them as 0, it means NOT showing the corresponding parameters.
 - bit 0: determine whether `<ecn>` will be shown.
 - bit 1: determine whether `<" ssid" >` will be shown.
 - bit 2: determine whether `<rssi>` will be shown.
 - bit 3: determine whether `<" mac" >` will be shown.
 - bit 4: determine whether `<channel>` will be shown.
 - bit 5: determine whether `<freq_offset>` will be shown.
 - bit 6: determine whether `<freqcal_val>` will be shown.
 - bit 7: determine whether `<pairwise_cipher>` will be shown.
 - bit 8: determine whether `<group_cipher>` will be shown.
 - bit 9: determine whether `<bgn>` will be shown.
 - bit 10: determine whether `<wps>` will be shown.

- **<rssifilter>**: determine whether the result of the command `AT+CWLAP` will be filtered according to `rssifilter`. In other words, the result of the command will **NOT** show the APs whose signal strength is below `rssifilter`. Unit: dBm. Default: -100. Range: [-100,40].
- **<authmode mask>**: determine whether APs with the following authmodes are shown in the result of `AT+CWLAP`. Default: 0xFFFF. If you set bit `x` to 1, the APs with the corresponding authmode will be shown. If you set bit `x` to 0, the APs with the corresponding authmode will NOT be shown;
 - bit 0: determine whether APs with OPEN authmode will be shown.
 - bit 1: determine whether APs with WEP authmode will be shown.
 - bit 2: determine whether APs with WPA_PSK authmode will be shown.
 - bit 3: determine whether APs with WPA2_PSK authmode will be shown.
 - bit 4: determine whether APs with WPA_WPA2_PSK authmode will be shown.
 - bit 5: determine whether APs with WPA2_ENTERPRISE authmode will be shown.
 - bit 6: determine whether APs with WPA3_PSK authmode will be shown.
 - bit 7: determine whether AP with WPA2_WPA3_PSK authmode will be shown.
 - bit 8: determine whether AP with WAPI_PSK authmode will be shown.
 - bit 9: determine whether AP with OWE authmode will be shown.
 - bit 10: determine whether AP with WPA3_ENT_192 authmode will be shown.
 - bit 11: determine whether AP with WPA3_EXT_PSK authmode will be shown.
 - bit 12: determine whether AP with WPA3_EXT_PSK_MIXED_MODE authmode will be shown.
 - bit 13: determine whether AP with DPP authmode will be shown.
 - bit 14: determine whether AP with WPA3_ENTERPRISE authmode will be shown.
 - bit 15: determine whether AP with WPA2_WPA3_ENTERPRISE authmode will be shown.

Example

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

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

3.2.10 AT+CWLAP: List Available APs

Set Command

Function:

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

Command:

```
AT+CWLAP=[<"ssid">][,<"mac">][,<channel>][,<scan_type>][,<scan_time_min>][,<scan_
↪time_max>][,<2ghz_channel_bitmap>]
```

Execute Command

Function:

List all available APs.

Command:

AT+CWLAP

Response:

```
+CWLAP:(<ecn>,<"ssid">,<rssi>,<"mac">,<channel>,<freq_offset>,<freqcal_val>,<br>↪<pairwise_cipher>,<group_cipher>,<wifi_protocol>,<wps>)<br>OK
```

Parameters

- **<ecn>**: encryption method.
 - 0: OPEN
 - 1: WEP
 - 2: WPA_PSK
 - 3: WPA2_PSK
 - 4: WPA_WPA2_PSK
 - 5: WPA2_ENTERPRISE
 - 6: WPA3_PSK
 - 7: WPA2_WPA3_PSK
 - 8: WAPI_PSK
 - 9: OWE
 - 10: WPA3_ENT_192
 - 11: WPA3_EXT_PSK
 - 12: WPA3_EXT_PSK_MIXED_MODE
 - 13: DPP
 - 14: WPA3_ENTERPRISE
 - 15: WPA2_WPA3_ENTERPRISE
- **<"ssid">**: string parameter showing SSID of the AP.
- **<rssi>**: signal strength.
- **<"mac">**: string parameter showing MAC address of the AP.
- **<channel>**: channel number. When specified, only scan this channel.
- **<scan_type>**: Wi-Fi scan type. Default: 0.
 - 0: active scan
 - 1: passive scan
- **<scan_time_min>**: the minimum active scan time per channel. Unit: millisecond. Range [0,1500]. If the scan type is passive, this parameter is invalid.
- **<scan_time_max>**: the maximum active scan time per channel. Unit: millisecond. Range [0,1500]. If this parameter is 0, the firmware will use the default time: 120 ms for active scan; 360 ms for passive scan.
- **<2ghz_channel_bitmap>**: Bitmap for 2.4 GHz band channels. Effective when **<channel>** is omitted or set to 0. Each bit corresponds to a channel; 1 means scan this channel, 0 means do not scan. For supported 2.4 GHz channels, see: [2.4G Channels](#). For example, if bit 1 and bit 2 are set to 1, it means scanning channels 1 and 2.
- **<freq_offset>**: frequency offset (reserved item).
- **<freqcal_val>**: frequency calibration value (reserved item).
- **<pairwise_cipher>**: pairwise cipher type.
 - 0: None
 - 1: WEP40
 - 2: WEP104
 - 3: TKIP
 - 4: CCMP
 - 5: TKIP and CCMP
 - 6: AES-CMAC-128
 - 7: Unknown
- **<group_cipher>**: group cipher type, same enumerated value to **<pairwise_cipher>**.

- **<wifi_protocol>**: Wi-Fi protocol standard. If the corresponding bit is 1, the corresponding mode is enabled; if the corresponding bit is 0, the corresponding mode is disabled.
 - bit 0: bit to identify if 802.11b mode is enabled or not
 - bit 1: bit to identify if 802.11g mode is enabled or not
 - bit 2: bit to identify if 802.11n mode is enabled or not
 - bit 3: [802.11 LR Espressif Proprietary protocol](#)
 - bit 4: 802.11ax protocol standard
 - bit 5: 802.11a protocol standard
 - bit 6: 802.11ac protocol standard
- **<wps>**: wps flag.
 - 0: WPS disabled
 - 1: WPS enabled

Notes

- When the SSID contains , , " or \ characters, it is necessary to escape these characters before configuration.
- When unable to scan APs with Chinese SSID, ensure SSID character encoding matches the router (usually UTF-8).
- When scan results are abnormal, it is recommended to use packet capture tools to analyze the SSID field in Probe Request packets and check whether the "<ssid>" parameter matches the target AP's SSID completely.

Example

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

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

3.2.11 AT+CWQAP: Disconnect from an AP

Execute Command

Command:

```
AT+CWQAP
```

Response:

```
OK
```

3.2.12 AT+CWSAP: Query or set the configuration of an ESP32 SoftAP

Query Command

Function:

Query the configuration parameters of an ESP32 SoftAP.

Command:

```
AT+CWSAP?
```

Response:

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

Set Command

Function:

Set the configuration of an ESP32 SoftAP.

Command:

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

Response:

```
OK
```

Parameters

- <"ssid">: string parameter showing SSID of the AP.
- <"pwd">: string parameter showing the password. Length: 8 ~ 64 bytes ASCII.
- <channel>: channel ID.
- <ecn>: encryption method; WEP is not supported.
 - 0: OPEN
 - 2: WPA_PSK
 - 3: WPA2_PSK
 - 4: WPA_WPA2_PSK
- <max conn>: maximum number of stations that ESP32 SoftAP can connect. Range: refer to [max_connection description](#).
- <ssid hidden>:
 - 0: Broadcasting SSID (default).
 - 1: Not broadcasting SSID.

Notes

- This command works only when [AT+CWMODE=2](#) or [AT+CWMODE=3](#).
- The configuration changes will be saved in the NVS partition if [AT+SYSTORE=1](#).
- The default SSID varies from devices to device as it consists of the MAC address of the device. You can use [AT+CWSAP?](#) to query the default SSID.

Example

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

3.2.13 AT+CWLIF: Obtain IP Address of the Station That Connects to an ESP32 SoftAP

Execute Command

Command:

```
AT+CWLIF
```

Response:

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

```
OK
```

Parameters

- **<ip addr>**: IP address of the station that connects to the ESP32 SoftAP.
- **<mac>**: MAC address of the station that connects to the ESP32 SoftAP.

Note

- This command cannot get a static IP. It works only when DHCP of both the ESP32 SoftAP and the connected station are enabled.

3.2.14 AT+CWQIF: Disconnect Stations from an ESP32 SoftAP

Execute Command

Function:

Disconnect all stations that are connected to the ESP32 SoftAP.

Command:

```
AT+CWQIF
```

Response:

```
OK
```

Set Command

Function:

Disconnect a specific station from the ESP32 SoftAP.

Command:

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

Response:

```
OK
```

Parameter

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

3.2.15 AT+CWDHCP: Enable or disable DHCP

Query Command

Command:

```
AT+CWDHCP?
```

Response:

```
+CWDHCP:<state>  
OK
```


Set Command

Function:

Enable or disable DHCP.

Command:

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

Response:

```
OK
```

Parameters

- **<operate>**:
 - 0: disable
 - 1: enable
- **<mode>**:
 - bit 0: Station DHCP
 - bit 1: SoftAP DHCP
- **<state>**: the status of DHCP
 - bit 0:
 - * 0: Station DHCP is disabled.
 - * 1: Station DHCP is enabled.
 - bit 1:
 - * 0: SoftAP DHCP is disabled.
 - * 1: SoftAP DHCP is enabled.
 - bit 2:
 - * 0: Ethernet DHCP is disabled.
 - * 1: Ethernet DHCP is enabled.

Notes

- The configuration changes will be saved in the NVS partition if [AT+SYSSTORE=1](#).
- This Set Command correlates with the commands that set static IP, such as [AT+CIPSTA](#) and [AT+CIPAP](#):
 - If DHCP is enabled, static IPv4 address will be disabled;
 - If static IPv4 address is enabled, DHCP will be disabled;
 - The last configuration overwrites the previous configuration.

Example

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

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

3.2.16 AT+CWDHCP: Query or Set the IPv4 Addresses Allocated by an ESP32 SoftAP DHCP Server

Query Command

Command:

```
AT+CWDHCPS?
```

Response:

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

Set Command

Function:

Set the IPv4 address range of the ESP32 SoftAP DHCP server.

Command:

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

Response:

```
OK
```

Parameters

- **<enable>**:
 - 1: Enable DHCP server settings. The parameters below have to be set.
 - 0: Disable DHCP server settings and use the default IPv4 address range.
- **<lease time>**: lease time. Unit: minute. Range [1,2880].
- **<" start IP" >**: start IPv4 address of the IPv4 address range that can be obtained from ESP32 SoftAP DHCP server.
- **<" end IP" >**: end IPv4 address of the IPv4 address range that can be obtained from ESP32 SoftAP DHCP server.

Notes

- The configuration changes will be saved in the NVS partition if [AT+SYSSSTORE=1](#).
- This AT command works only when both SoftAP and DHCP server are enabled for ESP32.
- The IPv4 address should be in the same network segment as the IPv4 address of ESP32 SoftAP.

Example

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

```
AT+CWDHCPS=0 // Disable the settings and use the default IPv4 address range.
```

3.2.17 AT+CWAUTOCONN: Query or Set Automatic Connection to an AP When Powered on

Query Command

Command:

```
AT+CWAUTOCONN?
```

Response:

```
+CWAUTOCONN:<enable>  
OK
```

Set Command**Command:**

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

Response:

```
OK
```

Parameters

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

Note

- The configuration changes will be saved in the NVS partition.
- For power-sensitive scenarios, it is recommended to disable automatic connection to AP on power-on, or set a smaller `<repeat_count>` value in [AT+CWRECONNCFG](#) to avoid additional power consumption from continuous reconnection attempts after power-on.

Example

```
AT+CWAUTOCONN=1
```

3.2.18 AT+CWAPPROTO: Query or Set the Wi-Fi Protocol Standard of SoftAP Mode

Query Command**Command:**

```
AT+CWAPPROTO?
```

Response:

```
+CWAPPROTO:<protocol_2g>  
OK
```

Set Command**Command:**

```
AT+CWAPPROTO=<protocol_2g>
```

Response:

```
OK
```

Parameters

- **<protocol_2g>**: Wi-Fi protocol standard for 2.4 GHz band
 - bit 0: 802.11b protocol standard.
 - bit 1: 802.11g protocol standard.
 - bit 2: 802.11n protocol standard.
 - bit 3: [802.11 LR Espressif Proprietary protocol](#).

Notes

- The configuration changes will be saved in the NVS partition if [AT+SYSTORE=1](#).
- See [Wi-Fi Protocol Mode](#) for the Wi-Fi protocol supported by the ESP32 device.
- By default, the SoftAP protocol standard of ESP32 is 802.11bgn.
- Note: Wi-Fi protocol standards are limited by hardware capabilities and Wi-Fi bandwidth settings. For example: in 40 MHz bandwidth mode, only 802.11n (2.4 GHz) can be configured.

3.2.19 AT+CWSTAPROTO: Query or Set the Wi-Fi Protocol Standard of Station Mode

Query Command**Command:**

```
AT+CWSTAPROTO?
```

Response:

```
+CWSTAPROTO:<protocol_2g>  
OK
```

Set Command**Command:**

```
AT+CWSTAPROTO=<protocol_2g>
```

Response:

```
OK
```

Parameters

- **<protocol_2g>**: Wi-Fi protocol standard for 2.4 GHz band
 - bit 0: 802.11b protocol standard.
 - bit 1: 802.11g protocol standard.

- bit 2: 802.11n protocol standard.
- bit 3: [802.11 LR Espressif Proprietary protocol](#).

Notes

- The configuration changes will be saved in the NVS partition if [AT+SYSTORE=1](#).
- See [Wi-Fi Protocol Mode](#) for the Wi-Fi protocol supported by the ESP32 device.
- By default, the Station protocol standard of ESP32 is 802.11bgn.
- Note: Wi-Fi protocol standards are limited by hardware capabilities and Wi-Fi bandwidth settings. For example: in 40 MHz bandwidth mode, only 802.11n (2.4 GHz) can be configured.

3.2.20 AT+CIPSTAMAC: Query or Set the MAC Address of an ESP32 Station

Query Command

Function:

Query the MAC address of the ESP32 Station.

Command:

```
AT+CIPSTAMAC?
```

Response:

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

Set Command

Function:

Set the MAC address of an ESP32 station.

Command:

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

Response:

```
OK
```

Parameters

- **<" mac" >**: string parameter showing MAC address of an ESP32 station.

Notes

- The configuration changes will be saved in the NVS partition if [AT+SYSTORE=1](#).
- The MAC address of ESP32 Station is different from that of the ESP32 SoftAP and ESP32 Ethernet. 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 address and cannot be set.

Example

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

3.2.21 AT+CIPAPMAC: Query or Set the MAC Address of an ESP32 SoftAP**Query Command****Function:**

Query the MAC address of the ESP32 SoftAP.

Command:

```
AT+CIPAPMAC?
```

Response:

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

Set Command**Function:**

Set the MAC address of the ESP32 SoftAP.

Command:

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

Response:

```
OK
```

Parameters

- <" mac" >: string parameter showing MAC address of the ESP32 SoftAP.

Notes

- The configuration changes will be saved in the NVS partition if [AT+SYSSTORE=1](#).
- The MAC address of ESP32 SoftAP is different from that of the ESP32 station and ESP32 Ethernet. Please make sure that you do not set the same MAC address for both of them.
- Bit 0 of the ESP32 MAC address CANNOT be 1. For example, a MAC address can be "18:..." but not "15:..." .
- FF:FF:FF:FF:FF:FF and 00:00:00:00:00:00 are invalid MAC and cannot be set.

Example

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

3.2.22 AT+CIPSTA: Query or Set the IP Address of an ESP32 Station

Query Command

Function:

Query the IP address of the ESP32 Station.

Command:

```
AT+CIPSTA?
```

Response:

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

OK
```

Set Command

Function:

Set the IPv4 address of the ESP32 station.

Command:

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

Response:

```
OK
```

Parameters

- <" ip" >: string parameter showing the IPv4 address of the ESP32 station.
- <" gateway" >: gateway.
- <" netmask" >: netmask.
- <" ipv6 addr" >: string parameter showing the IPv6 address of the ESP32 station.

Notes

- For the query command, only when the ESP32 station is connected to an AP or the static IP address is configured can its IP address be queried.
- The configuration changes will be saved in the NVS partition if [AT+SYSTORE=1](#).
- The Set Command correlates with the commands that set DHCP, such as [AT+CWDHCP](#).
 - If static IPv4 address is enabled, DHCP will be disabled;
 - If DHCP is enabled, static IPv4 address will be disabled;
 - The last configuration overwrites the previous configuration.
- When using static IP, AT does not provide IP conflict detection. You can implement your own detection mechanism if needed.

Example

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

3.2.23 AT+CIPAP: Query or Set the IP Address of an ESP32 SoftAP

Query Command

Function:

Query the IP address of the ESP32 SoftAP.

Command:

```
AT+CIPAP?
```

Response:

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

OK
```

Set Command

Function:

Set the IPv4 address of the ESP32 SoftAP.

Command:

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

Response:

```
OK
```

Parameters

- <"ip">: string parameter showing the IPv4 address of the ESP32 SoftAP.
- <"gateway">: gateway.
- <"netmask">: netmask.
- <"ipv6 addr">: string parameter showing the IPv6 address of the ESP32 SoftAP.

Notes

- The set command is just applied to the IPv4 network, but not the IPv6 network.
- The configuration changes will be saved in the NVS partition if [AT+SYSTORE=1](#).
- The set command correlates with the commands that set DHCP, such as [AT+CWDHCP](#).
 - If static IPv4 address is enabled, DHCP will be disabled;
 - If DHCP is enabled, static IPv4 address will be disabled;
 - The last configuration overwrites the previous configuration.

Example

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

3.2.24 AT+CWSTARTSMART: Start SmartConfig

Execute Command

Function:

Start SmartConfig of the type ESP-TOUCH+AirKiss.

Command:

```
AT+CWSTARTSMART
```

Set Command

Function:

Start SmartConfig of a designated type.

Command:

```
AT+CWSTARTSMART=<type>[,<auth floor>][,<"esptouch v2 key">]
```

Response:

```
OK
```

Parameters

- **<type>:**
 - 1: ESP-TOUCH
 - 2: AirKiss
 - 3: ESP-TOUCH+AirKiss
 - 4: ESP-TOUCH v2
- **<auth floor>:** Wi-Fi authentication mode floor. ESP-AT will not connect to the AP whose authmode is lower than this floor.
 - 0: OPEN (Default)
 - 1: WEP
 - 2: WPA_PSK
 - 3: WPA2_PSK
 - 4: WPA_WPA2_PSK
 - 5: WPA2_ENTERPRISE
 - 6: WPA3_PSK
 - 7: WPA2_WPA3_PSK
 - 8: WAPI_PSK
 - 9: OWE
 - 10: WPA3_ENT_192
 - 11: WPA3_EXT_PSK
 - 12: WPA3_EXT_PSK_MIXED_MODE
 - 13: DPP
 - 14: WPA3_ENTERPRISE
 - 15: WPA2_WPA3_ENTERPRISE
- **<" esptouch v2 key" >:** ESP-TOUCH v2 decrypt key. It is used to decrypt Wi-Fi password and reserved data. Length: 16 bytes.

Notes

- For more 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 `+SCRD:<length>,<rvd data>` means that ESP-Touch v2 has successfully acquired the reserved data information.
- Message `Smartconfig connected Wi-Fi` is printed if the connection is successful.
- When AT returns `Smartconfig connected Wi-Fi`, it is recommended to delay more than 6 seconds before executing [AT+CWSTOPSMART](#) because the ESP32 needs to synchronize the SmartConfig results to the mobile phone.
- Use command [AT+CWSTOPSMART](#) to stop SmartConfig before running other commands. Please make sure that you do not execute other commands during SmartConfig.

Example

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

3.2.25 AT+CWSTOPSMART: Stop SmartConfig

Execute Command

Command:

```
AT+CWSTOPSMART
```

Response:

```
OK
```

Note

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

Example

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

3.2.26 AT+WPS: Enable the WPS Function

Set Command

Command:

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

Response:

OK

Parameters

- **<enable>**:
 - 1: Enable WPS (Wi-Fi Protected Setup) that uses PBC (Push Button Configuration) mode.
 - 0: Disable WPS that uses PBC mode.
- **<auth floor>**: Wi-Fi authentication mode floor. ESP-AT will not connect to the AP whose authmode is lower than this floor.
 - 0: OPEN (Default)
 - 1: WEP
 - 2: WPA_PSK
 - 3: WPA2_PSK
 - 4: WPA_WPA2_PSK
 - 5: WPA2_ENTERPRISE
 - 6: WPA3_PSK
 - 7: WPA2_WPA3_PSK
 - 8: WAPI_PSK
 - 9: OWE
 - 10: WPA3_ENT_192
 - 11: WPA3_EXT_PSK
 - 12: WPA3_EXT_PSK_MIXED_MODE
 - 13: DPP
 - 14: WPA3_ENTERPRISE
 - 15: WPA2_WPA3_ENTERPRISE

Notes

- WPS can only be used when the ESP32 station is enabled.
- WPS does not support WEP (Wired-Equivalent Privacy) encryption.

Example

AT+CWMODE=1
AT+WPS=1

3.2.27 AT+CWJEAP: Connect to a WPA2 Enterprise AP

Query Command

Function:

Query the configuration information of the Enterprise AP to which the ESP32 station is already connected.

Command:

AT+CWJEAP?

Response:

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

Set Command

Function:

Connect to the targeted Enterprise AP.

Command:

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

Response:

```
OK
```

or

```
+CWJEAP:Timeout  
ERROR
```

Parameters

- **<"ssid">**: the SSID of the Enterprise AP.
 - Escape character syntax is needed if SSID or password contains any special characters, such as , , " , or \\.
- **<method>**: WPA2 Enterprise authentication method.
 - 0: EAP-TLS.
 - 1: EAP-PEAP.
 - 2: EAP-TTLS.
- **<"identity">**: identity for phase 1. String limited to 1 ~ 32.
- **<"username">**: username for phase 2. Range: 1 ~ 32 bytes. For the EAP-PEAP and EAP-TTLS method, you must set this parameter. For the EAP-TLS method, you do not need to.
- **<"password">**: password for phase 2. Range: 1 ~ 32 bytes. For the EAP-PEAP and EAP-TTLS method, you must set this parameter. For the EAP-TLS method, you do not need to.
- **<security>**:
 - bit 0: provide the certificate for the server to verify.
 - bit 1: load the CA certificate to verify the server's certificate.
- **<jeap_timeout>**: maximum timeout for **AT+CWJEAP** command. Unit: second. Default: 15. Range: [3,600].

Example

```
// Connect to EAP-TLS mode Enterprise AP, set identity, verify server certificate,  
↪and load client certificate  
AT+CWJEAP="dlink11111",0,"example@espressif.com",,,3  
  
// Connect to EAP-PEAP mode Enterprise AP, set identity, username and password,  
↪not verify server certificate and not load client certificate  
AT+CWJEAP="dlink11111",1,"example@espressif.com","espressif","test11",0
```

Error Code:

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

AT_EAP_MALLOC_FAILED	0x8001
AT_EAP_GET_NVS_CONFIG_FAILED	0x8002
AT_EAP_CONN_FAILED	0x8003

continues on next page

Table 2 – continued from previous page

AT_EAP_MALLOC_FAILED	0x8001
AT_EAP_SET_WIFI_CONFIG_FAILED	0x8004
AT_EAP_SET_IDENTITY_FAILED	0x8005
AT_EAP_SET_USERNAME_FAILED	0x8006
AT_EAP_SET_PASSWORD_FAILED	0x8007
AT_EAP_GET_CA_LEN_FAILED	0x8008
AT_EAP_READ_CA_FAILED	0x8009
AT_EAP_SET_CA_FAILED	0x800A
AT_EAP_GET_CERT_LEN_FAILED	0x800B
AT_EAP_READ_CERT_FAILED	0x800C
AT_EAP_GET_KEY_LEN_FAILED	0x800D
AT_EAP_READ_KEY_FAILED	0x800E
AT_EAP_SET_CERT_KEY_FAILED	0x800F
AT_EAP_ENABLE_FAILED	0x8010
AT_EAP_ALREADY_CONNECTED	0x8011
AT_EAP_GET_SSID_FAILED	0x8012
AT_EAP_SSID_NULL	0x8013
AT_EAP_SSID_LEN_ERROR	0x8014
AT_EAP_GET_METHOD_FAILED	0x8015
AT_EAP_CONN_TIMEOUT	0x8016
AT_EAP_GET_IDENTITY_FAILED	0x8017
AT_EAP_IDENTITY_LEN_ERROR	0x8018
AT_EAP_GET_USERNAME_FAILED	0x8019
AT_EAP_USERNAME_LEN_ERROR	0x801A
AT_EAP_GET_PASSWORD_FAILED	0x801B
AT_EAP_PASSWORD_LEN_ERROR	0x801C
AT_EAP_GET_SECURITY_FAILED	0x801D
AT_EAP_SECURITY_ERROR	0x801E
AT_EAP_METHOD_SECURITY_UNMATCHED	0x801F
AT_EAP_PARAMETER_COUNTS_ERROR	0x8020
AT_EAP_GET_WIFI_MODE_ERROR	0x8021
AT_EAP_WIFI_MODE_NOT_STA	0x8022
AT_EAP_SET_CONFIG_FAILED	0x8023
AT_EAP_METHOD_ERROR	0x8024

Note

- The configuration changes will be saved in the NVS partition if [AT+SYSTORE=1](#).
- This command requires Station mode to be active.
- TLS mode will use client certificate. Please make sure it is enabled.
- If you want to use your own certificate at runtime, use the [AT+SYSMFG](#) command to update the WPA2 Enterprise certificate (for detailed steps, please refer to [AT+SYSMFG command examples](#), the certificate configuration method is the same as SSL certificates). If you want to pre-burn your own certificate, please refer to [How to Update PKI Configuration](#).
- If <security> is configured to 2, in order to check the server certificate validity period, please make sure ESP32 has obtained the current time before sending the [AT+CWJEAP](#) command. (You can send [AT+SYSTIMESTAMP](#) command to configure the current time, and send [AT+SYSTIMESTAMP?](#) command to query the current time.)

3.2.28 AT+CWHOSTNAME: Query or Set the Host Name of an ESP32 Station**Query Command****Function:**

Query the host name of ESP32 Station.

Command:

```
AT+CWHOSTNAME?
```

Response:

```
+CWHOSTNAME:<hostname>
OK
```

Set Command**Function:**

Set the host name of ESP32 Station.

Command:

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

Response:

```
OK
```

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

```
ERROR
```

Parameters

- **<" hostname" >**: the host name of the ESP32 Station. Maximum length: 32 bytes.

Note

- The configuration changes are not saved in the flash.

Example

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

3.2.29 AT+CWCOUNTRY: Query or Set the Wi-Fi Country Code

Query Command**Function:**

Query Wi-Fi country code information.

Command:

```
AT+CWCOUNTRY?
```

Response:

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

```
OK
```

Set Command

Function:

Set the Wi-Fi country code information.

Command:

```
AT+CWCOUNTRY=<country_policy>,<"country_code">,<start_channel>,<total_channel_↵count>
```

Response:

```
OK
```

Parameters

- **<country_policy>**:
 - 0: Will change the county code to be the same as the AP that the ESP32 is connected to.
 - 1: The country code will not change, always be the one set by command.
- **<"country_code">**: country code. Maximum length: 3 characters. Refer to [ISO 3166-1 alpha-2](#) for country codes.
- **<start_channel>**: the channel number to start. Range: [1,14].
- **<total_channel_count>**: total number of channels.

Note

- See [Wi-Fi Country Code](#) for more details.
- The configuration changes are not saved in the flash.

Example

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

3.3 TCP/IP AT Commands

- [Introduction](#)
- [AT+CIPV6](#): Enable or disable the network of Internet Protocol Version 6 (IPv6).
- [AT+CIPSTATE](#): Obtain the TCP/UDP/SSL connection information.
- [AT+CIPDOMAIN](#): Resolve a Domain Name.
- [AT+CIPSTART](#): Establish TCP connection, UDP transmission, or SSL connection.
- [AT+CIPSTARTEX](#): Establish TCP connection, UDP transmission, or SSL connection with an automatically assigned ID.
- [\[Data Mode Only\] +++](#): Exit from the *data mode*.
- [AT+SAVETRANSLINK](#): Set whether to enter Network *Passthrough Mode* on power-up.

- **AT+CIPSEND**: Send data in the *normal transmission mode* or Network *normal transmission mode*.
- **AT+CIPSENDL**: Send long data in parallel in the *normal transmission mode*.
- **AT+CIPSENDLCFG**: Set the configuration for the command **AT+CIPSENDL**.
- **AT+CIPSENDEX**: Send data in the *normal transmission mode* in expanded ways.
- **AT+CIPCLOSE**: Close TCP/UDP/SSL connection.
- **AT+CIPCONNPERSIST**: Query or set TCP/SSL connection persistence.
- **AT+CIFSR**: Obtain the local IP address and MAC address.
- **AT+CIPMUX**: Enable or disable the multiple connections mode.
- **AT+CIPSERVER**: Delete or create a TCP/SSL server.
- **AT+CIPSERVERMAXCONN**: Query or set the maximum connections allowed by a server.
- **AT+CIPMODE**: Query or set the transmission mode.
- **AT+CIPSTO**: Query or set the local TCP Server Timeout.
- **AT+CIPSNTPCFG**: Query or set the time zone and SNTP server.
- **AT+CIPSNTPTIME**: Query the SNTP time.
- **AT+CIPSNTPTINTV**: Query or set the SNTP time synchronization interval.
- **AT+CIPFWVER**: Query the existing AT firmware version on the server.
- **AT+CIUPDATE**: Upgrade the firmware through Network.
- **AT+CIPDINFO**: Set “+IPD” message mode.
- **AT+CIPSSLCCONF**: Query or set SSL clients.
- **AT+CIPSSLCCIPHER**: Query or set the cipher suite of the SSL client.
- **AT+CIPSSLCCN**: Query or set the Common Name of the SSL client.
- **AT+CIPSSLCSNI**: Query or set SSL client Server Name Indication (SNI).
- **AT+CIPSSLCALPN**: Query or set SSL client Application Layer Protocol Negotiation (ALPN).
- **AT+CIPSSLCP SK**: Query or set SSL client Pre-shared Key (PSK) in string format.
- **AT+CIPSSLCP SKHEX**: Query or set SSL client Pre-shared Key (PSK) in hexadecimal format.
- **AT+CIPRECONNINTV**: Query or set the TCP/UDP/SSL reconnection interval for the Network *normal transmission mode*.
- **AT+CIPREC VTYPE**: Query or set socket receiving mode.
- **AT+CIPREC VDATA**: Obtain socket data in passive receiving mode.
- **AT+CIPREC VLEN**: Obtain socket data length in passive receiving mode.
- **AT+PING**: Ping the remote host.
- **AT+CIPDNS**: Query or set DNS server information.
- **AT+MDNS**: Configure the mDNS function.
- **AT+CIPTCPOPT**: Query or set the socket options.

3.3.1 Introduction

Important: The default AT firmware supports all the AT commands mentioned on this page. If you need to modify the commands supported by ESP32 by default, please compile the ESP-AT project by following the steps in [Compile ESP-AT Project Locally](#) documentation. In the project configuration during the fifth step, make the following selections (Each item below is independent. Choose it according to your needs):

- Disable OTA commands (**AT+CIUPDATE** and **AT+CIPFWVER**): Component config>AT>AT OTA command support
 - Disable PING commands (**AT+PING**): Component config>AT>AT ping command support
 - Disable mDNS commands (**AT+MDNS**): Component config>AT>AT MDNS command support
 - Disable TCP/IP commands (Not recommended. Once disabled, all TCP/IP functions will be unavailable and you will need to implement these AT commands yourself): Component config>AT>AT net command support
-

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

Query Command

Function:

Query whether IPv6 is enabled.

Command:

```
AT+CIPV6?
```

Response:

```
+CIPV6:<enable>
OK
```

Set Command

Function:

Enable or disable IPv6 network.

Command:

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

Response:

```
OK
```

Parameters

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

Notes

- The configuration changes will be saved in the NVS partition if AT+SYSSTORE=1.
- You should enable IPv6 network before using IPv6 related upper layer AT commands (TCP/UDP/SSL/PING/DNS based on IPv6 network, also known as TCP6/UDP6/SSL6/PING6/DNS6 or TCPv6/UDPv6/SSLv6/PINGv6/DNSv6).

3.3.3 AT+CIPSTATE: Obtain the TCP/UDP/SSL Connection Information

Query Command

Command:

```
AT+CIPSTATE?
```

Response:

When there is a connection, AT returns:

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

When there is no connection, AT returns:

OK

Parameters

- **<link ID>**: ID of the connection (0~4), used for multiple connections.
- **<" type" >**: string parameter showing the type of transmission: "TCP", "TCPv6", "UDP", "UDPv6", "SSL", or "SSLv6".
- **<" remote IP" >**: string parameter showing the remote IPv4 address or IPv6 address.
- **<remote port>**: the remote port number.
- **<local port>**: the local port number.
- **<tetype>**:
 - 0: ESP32 runs as a client.
 - 1: ESP32 runs as a server.

3.3.4 AT+CIPDOMAIN: Resolve a Domain Name

Set Command

Command:

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

Response:

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

OK

Parameter

- **<" domain name" >**: the domain name.
- **<ip network>**: preferred IP network. Default: 1.
 - 1: Preferred resolution of IPv4 address.
 - 2: Resolve IPv4 address only.
 - 3: Resolve IPv6 address only.
- **<" IP address" >**: the resolved IPv4 address or IPv6 address.
- **<timeout>**: Command timeout. Unit: milliseconds. Default value: 0. Range: [0,60000]. When set to 0, the command timeout depends on the network and lwIP protocol stack; when set to a non-zero value, the command will return within the specified timeout, but it will consume about 5 KB more heap space.
- **<show_all_ip>**: whether to show all resolved IP addresses. Default: 0.
 - 0: Only show the first resolved IP address.
 - 1: Show all resolved IP addresses.

Notes

- To resolve a domain name to multiple IP addresses, please increase the value of `python build.py menu-config>Component config>LWIP>DNS>Maximum number of IP addresses per host`. Then recompile the ESP-AT project. The command returns the following format:

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

Example

```

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

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

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

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

```

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

- *Establish TCP Connection*
- *Establish UDP Transmission*
- *Establish SSL Connection*

Establish TCP Connection**Set Command Command:**

```

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

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

```

Response:

For single connection, it returns:

```

CONNECT

OK

```

For multiple connections, it returns:

```

<link ID>,<CONNECT>

OK

```

Parameters

- **<link ID>**: ID of network connection (0~4), used for multiple connections. The range of this parameter depends on two configuration items in menuconfig. One is AT_SOCKET_MAX_CONN_NUM of the AT component, and its default value is 5. The other is LWIP_MAX_SOCKETS of the LWIP component, and its default value is 10. To modify the range of this parameter, you need to set AT_SOCKET_MAX_CONN_NUM and make sure it is no larger than the value of LWIP_MAX_SOCKETS. (See [Compile ESP-AT Project Locally](#) for details on configuring and build ESP-AT projects.)
- **<"type">**: string parameter showing the type of transmission: "TCP", or "TCPv6". Default: "TCP".

- **<" remote host" >**: IPv4 address, IPv6 address, or domain name of remote host. The maximum length is 64 bytes. If you need to use a domain name and the length of the domain name exceeds 64 bytes, use the [AT+CIPDOMAIN](#) command to obtain the IP address corresponding to the domain name, and then use the IP address to establish a connection.
- **<remote port>**: the remote port number.
- **<keep_alive>**: It configures the [SO_KEEPALIVE](#) option for socket. Unit: second.
 - Range: [0,7200].
 - * 0: Disable keep-alive function (default).
 - * 1 ~ 7200: Enable keep-alive function. [TCP_KEEPIPLE](#) value is **<keep_alive>**, [TCP_KEEPIPLE](#) value is 1, and [TCP_KEEPCNT](#) value is 3.
 - This parameter of this command is the same as the **<keep_alive>** parameter of [AT+CIPTCPOPT](#) command. It always takes the value set later by either of the two commands. If it is omitted or not set, the last configured value is used by default.
- **<" local IP" >**: the local IPv4 address or IPv6 address that the connection binds. This parameter is useful when you are using multiple network interfaces or multiple IP addresses. By default, it is disabled. If you want to use it, you should specify it first. Null is also valid.
- **<timeout>**: Command timeout. Unit: milliseconds. Default value: 0. Range: [0,60000]. When set to 0, the command timeout depends on the network and lwIP protocol stack; when set to a non-zero value, the command will return within the specified timeout, but it will consume about 5 KB more heap space.

Notes

- To establish a TCP connection based on an IPv6 network, do as follows:
 - Make sure that the AP supports IPv6
 - Set [AT+CIPV6=1](#)
 - Obtain an IPv6 address through the [AT+CWJAP](#) command
 - (Optional) Check whether ESP32 has obtained an IPv6 address using the [AT+CIPSTA?](#) command
- To view the reason for TCP connection establishment failure, please first run [AT+SYSLOG=1](#) to enable logging, then retry this command. Upon failure, AT will return a more detailed error code +ERRNO:<error_code> to help locate the problem.

Example

```
AT+CIPSTART="TCP", "iot.espressif.cn", 8000
AT+CIPSTART="TCP", "192.168.101.110", 1000
AT+CIPSTART="TCP", "192.168.101.110", 2500, 60
AT+CIPSTART="TCP", "192.168.101.110", 1000, , "192.168.101.100"

// Connect to GitHub's TCP server with a 5-second timeout
AT+CIPSTART="TCP", "www.github.com", 80, , , 5000

AT+CIPSTART="TCPv6", "test-ipv6.com", 80
AT+CIPSTART="TCPv6", "fe80::860d:8eff:fe9d:cd90", 1000, , "fe80::411c:1fdb:22a6:4d24"

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

Establish UDP Transmission

Set Command Command:

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

// Multiple connections (AT+CIPMUX=1):
```

(continues on next page)

(continued from previous page)

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

Response:

For single connection, it returns:

```
CONNECT
OK
```

For multiple connections, it returns:

```
<link ID>,CONNECT
OK
```

Parameters

- **<link ID>**: ID of network connection (0~4), used for multiple connections.
- **<" type">**: string parameter showing the type of transmission: "UDP" , or "UDPv6" . Default: "TCP"
- **<" remote host">**: IPv4 address, IPv6 address, or domain name of remote host. The maximum length is 64 bytes. If you need to use a domain name and the length of the domain name exceeds 64 bytes, use the [AT+CIPDOMAIN](#) command to obtain the IP address corresponding to the domain name, and then use the IP address to establish a connection.
- **<remote port>**: remote port number.
- **<local port>**: UDP port of ESP32.
- **<mode>**: In the UDP passthrough, the value of this parameter has to be 0.
 - 0: After UDP data is received, the parameters <"remote host"> and <remote port> will stay unchanged (default).
 - 1: Only the first time that UDP data is received from an IP address and port that are different from the initially set value of parameters <remote host> and <remote port>, will they be changed to the IP address and port of the device that sends the data.
 - 2: Each time UDP data is received, the <"remote host"> and <remote port> will be changed to the IP address and port of the device that sends the data.
- **<" local IP">**: the local IPv4 address or IPv6 address that the connection binds. This parameter is useful when you are using multiple network interfaces or multiple IP addresses. By default, it is disabled. If you want to use it, you should specify it first. Null is also valid.
- **<timeout>**: Command timeout. Unit: milliseconds. Default value: 0. Range: [0,60000]. When set to 0, the command timeout depends on the network and lwIP protocol stack; when set to a non-zero value, the command will return within the specified timeout, but it will consume about 5 KB more heap space.

Notes

- If the remote host over the UDP is an IPv4 multicast address (224.0.0.0 ~ 239.255.255.255), the ESP32 will send and receive the UDPv4 multicast.
- If the remote host over the UDP is an IPv4 broadcast address (255.255.255.255), the ESP32 will send and receive the UDPv4 broadcast. Note that AT does not support automatic repeated broadcasting. If periodic broadcasting is required, the MCU needs to send data packets at regular intervals.
- If the remote host over the UDP is an IPv6 multicast address (FF00:0:0:0:0:0:0 ~ FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF), the ESP32 will send and receive the UDP multicast based on IPv6 network.
- To use the parameter <mode>, parameter <local port> must be set first.
- To establish an UDP transmission based on an IPv6 network, do as follows:
 - Make sure that the AP supports IPv6
 - Set [AT+CIPV6=1](#)
 - Obtain an IPv6 address through the [AT+CWJAP](#) command

- (Optional) Check whether ESP32 has obtained an IPv6 address using the [AT+CIPSTA?](#) command
- If you want to receive a UDP packet longer than 1460 bytes, please compile the firmware on your own by following [Compile ESP-AT Project](#) and choosing the following configurations in the Step 5. Configure: Component config>LWIP>Enable reassembly incoming fragmented IP4 packets.
- To view the reason for UDP transmission establishment failure, please first run [AT+SYSLOG=1](#) to enable logging, then retry this command. Upon failure, AT will return a more detailed error code +ER-RNO:<error_code> to help locate the problem.

Example

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

// Establish UDP transmission with pool.ntp.org, set 5-second timeout
AT+CIPSTART="UDP", "pool.ntp.org", 123, , , 5000

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

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

Establish SSL Connection

Set Command Command:

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

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

Response:

For single connection, it returns:

```
CONNECT
OK
```

For multiple connections, it returns:

```
<link ID>,CONNECT
OK
```

Parameters

- **<link ID>**: ID of network connection (0~4), used for multiple connections.
- **<" type" >**: string parameter showing the type of transmission: "SSL", or "SSLv6". Default: "TCP".
- **<" remote host" >**: IPv4 address, IPv6 address, or domain name of remote host. The maximum length is 64 bytes. If you need to use a domain name and the length of the domain name exceeds 64 bytes, use the [AT+CIPDOMAIN](#) command to obtain the IP address corresponding to the domain name, and then use the IP address to establish a connection.
- **<remote port>**: the remote port number.
- **<keep_alive>**: It configures the [SO_KEEPALIVE](#) option for socket. Unit: second.

- Range: [0,7200].
 - * 0: Disable keep-alive function (default).
 - * 1 ~ 7200: Enable keep-alive function. `TCP_KEEPIDLE` value is `<keep_alive>`, `TCP_KEEPINTVL` value is 1, and `TCP_KEEPCNT` value is 3.
- This parameter of this command is the same as the `<keep_alive>` parameter of `AT+CIPTCP OPT` command. It always takes the value set later by either of the two commands. If it is omitted or not set, the last configured value is used by default.
- **<" local IP">**: the local IPv4 address or IPv6 address that the connection binds. This parameter is useful when you are using multiple network interfaces or multiple IP addresses. By default, it is disabled. If you want to use it, you should specify it first. Null is also valid.
- **<timeout>**: Command timeout. Unit: milliseconds. Default value: 0. Range: [0,60000]. When set to 0, the command timeout depends on the network and lwIP protocol stack; when set to a non-zero value, the command will return within the specified timeout, but it will consume about 5 KB more heap space.

Notes

- The number of SSL connections depends on available memory and the maximum number of connections.
- SSL connection needs a large amount of memory. Insufficient memory may reboot the system.
- If the AT+CIPSTART is based on an SSL connection and the timeout of each packet is 10 s, the total timeout increases with the number of handshake packets.
- Some SSL servers require the SSL client to support the SNI extension field during connection. Please configure SNI using the `AT+CIPSSLCSNI` command before establishing an SSL connection. Typically, the value of SNI is the domain name of the server.
- To modify the TLS protocol version, recompile the ESP-AT project. Navigate to `python build.py menuconfig>Component config>mbedtls` directory. Disable TLS versions you do not need or enable versions not supported by default.
 - For example, to enable the TLS 1.3 protocol:
 - `python build.py menuconfig>Component config>mbedtls>Using dynamic TX/RX buffer>Uncheck`
 - `python build.py menuconfig>Component config>mbedtls>mbedtls v3.x related>Keep peer certificate after handshake completion>Check`
 - `python build.py menuconfig>Component config>mbedtls>mbedtls v3.x related>Support TLS 1.3 protocol>Check`
- To establish an SSL connection based on an IPv6 network, do as follows:
 - Make sure that the AP supports IPv6
 - Set `AT+CIPV6=1`
 - Obtain an IPv6 address through the `AT+CWJAP` command
 - (Optional) Check whether ESP32 has obtained an IPv6 address using the `AT+CIPSTA?` command
- To view the reason for SSL connection establishment failure, please first run `AT+SYSLOG=1` to enable logging, then retry this command. Upon failure, AT will return a more detailed error code `+ERRNO:<error_code>` to help locate the problem.

Example

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

// Connect to Microsoft Bing's SSL server with a 5-second timeout
AT+CIPSTART="SSL", "www.bing.com", 443, , , 5000

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

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

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

3.3.7 [Data Mode Only] +++: Exit from Data Mode

Special Execute Command

Function:

Exit from *Data Mode* and enter the *Command Mode*.

Command:

```
// Only for data mode
+++
```

Notes

- This special execution command consists of three identical + characters (0x2b ASCII), and no CR-LF appends to the command tail.
- Make sure there is more than 20 ms interval before the first + character, more than 20 ms interval after the third + character, less than 20 ms interval among the three + characters. Otherwise, the + characters will be sent out as normal data.
- This command returns no reply.
- Please wait for at least one second before sending the next AT command.

3.3.8 AT+CIPSEND: Send Data in the Normal Transmission Mode or Network Passthrough Mode

Set Command

Function:

Set the data length to be send in the *Normal Transmission Mode*. If the length of data you need to send exceeds 8192 bytes, please use the *AT+CIPSENDL* command.

Command:

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

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

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

Response:

```
OK

>
```

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

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

```
ERROR
```

If the data has been successfully sent to the protocol stack (It does not mean that the data has been sent to the opposite end), the system returns:

```
SEND OK
```

Execute Command

Function:

Enter the Network *Passthrough Mode*.

Command:

```
AT+CIPSEND
```

Response:

```
OK
>
```

or

```
ERROR
```

Enter the Network *Passthrough Mode*. The ESP32 can receive 8192 bytes and send 2920 bytes at most each time. If the data received by ESP32 reaches or exceeds 2920 bytes, the data will be immediately sent in chunks of 2920 bytes. Otherwise, it will wait for 20 milliseconds before being sent (You can configure this interval using *AT+TRANSINTVL* command). When a single packet containing +++ is received, the ESP32 will exit the data sending mode under the *Passthrough Mode*. Please wait for at least one second before sending the next AT command.

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

Parameters

- **<link ID>**: ID of the connection (0~4), for multiple connections.
- **<length>**: data length. Maximum: 8192 bytes.
- **<" remote host" >**: IPv4 address, IPv6 address, or domain name of remote host. It can be set in UDP transmission.
- **<remote port>**: the remote port number.

Notes

- You can use *AT+CIPTCPOPT* command to configure socket options for each TCP connection. For example, setting <so_sndtimeo> to 5000 will enable TCP send operation to return results within 5 seconds, regardless of success or failure. This can save the time that the MCU waits for AT command response.

3.3.9 AT+CIPSENDL: Send Long Data in Parallel in the Normal Transmission Mode.

Set Command

Function:

In the *Normal Transmission Mode*, set the data length to be sent, and then send data to remote host in parallel (the AT command port receives data in parallel with the AT sending data to the remote host). You can use the *AT+CIPSENDLCFG* command to configure this command. If the length of data you need to send is less than 8192 bytes, you also can use the *AT+CIPSEND* command.

Command:

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

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

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

Response:

```
OK

>
```

This response indicates that AT enters the *Data Mode* and AT command port is ready to receive data. You can enter the data now. Once the port receives data, it will be pushed to underlying protocol stack and the transmission starts.

If the transmission starts, the system reports message according to *AT+CIPSENDLCFG* configuration:

```
+CIPSENDL:<had sent len>,<port recv len>
```

If the transmission is cancelled by *+++* command, the system returns:

```
SEND CANCELLED
```

If not all the data has been sent out, the system finally returns:

```
SEND FAIL
```

If the data has been successfully sent to the protocol stack (It does not mean that the data has been sent to the opposite end), the system finally returns:

```
SEND OK
```

When the connection is disconnected, you can send *+++* command to cancel the transmission, then the ESP32 will exit from the *Data Mode*, otherwise, the *Data Mode* will not end until the AT command port receives all the data of the specified *<length>*.

Parameters

- **<link ID>**: ID of the connection (0~4), for multiple connections.
- **<length>**: data length. Maximum: $2^{31} - 1$ bytes.
- **<" remote host" >**: IPv4 address, IPv6 address, or domain name of remote host. It can be set in UDP transmission.
- **<remote port>**: the remote port number.
- **<had sent len>**: the length of data successfully sent to the underlying protocol stack.
- **<port recv len>**: data length received by AT command port.

Notes

- It is recommended to use UART flow control, because if the UART receives data at a faster rate than the network sends, data loss may occur.

- You can use [AT+CIPTCPOPT](#) command to configure socket options for each TCP connection. For example, setting `<so_sndtimeo>` to 5000 will enable TCP send operation to return results within 5 seconds, regardless of success or failure. This can save the time that the MCU waits for AT command response.

3.3.10 AT+CIPSENDLCFG: Set the Configuration for the Command AT+CIPSENDL

Query Command

Function:

Query the configuration of [AT+CIPSENDL](#).

Command:

```
AT+CIPSENDLCFG?
```

Response:

```
+CIPSENDLCFG:<report size>,<transmit size>
OK
```

Set Command

Function:

Set the configuration of [AT+CIPSENDL](#).

Command:

```
AT+CIPSENDLCFG=<report size>[,<transmit size>]
```

Response:

```
OK
```

Parameters

- **<report size>**: report block size for [AT+CIPSENDL](#). Default: 1024. Range: [100,2²⁰]. For example, set `<report size>` to 100, `<had sent len>` report sequence in the response of [AT+CIPSENDL](#) will be (100, 200, 300, 400, ...). The final `<had sent len>` report is always equal to the data length that had been sent out.
- **<transmit size>**: transmit block size of [AT+CIPSENDL](#). It specifies the size of the data block sent to the underlying protocol stack. Default: 2920. Range: [100,2920]. If the data received by ESP32 reaches or exceeds `<transmit size>` bytes, the data will be immediately sent in chunks of `<transmit size>` bytes. Otherwise, it will wait for 20 milliseconds before being sent (You can configure this interval using [AT+TRANSINTVL](#) command).

Note

- For devices with small throughput but high real-time requirements, it is recommended to set a smaller `<transmit size>`. It is also recommended to set `TCP_NODELAY` by [AT+CIPTCPOPT](#) command.
- For devices with large throughput, it is recommended to set a larger `<transmit size>`. It is also recommended to read [How to Improve ESP-AT Throughput Performance](#) first.

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

Set Command

Function:

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

Command:

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

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

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

Response:

```
OK

>
```

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

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

```
ERROR
```

If the data has been successfully sent to the protocol stack (It does not mean that the data has been sent to the opposite end), the system returns:

```
SEND OK
```

Parameters

- **<link ID>**: ID of the connection (0~4), for multiple connections.
- **<length>**: data length. Maximum: 8192 bytes.
- **<" remote host" >**: IPv4 address, IPv6 address, or domain name of remote host. It can be set in UDP transmission.
- **<remote port>**: remote port can be set in UDP transmission.

Notes

- When the requirement of data length is met, or when the string \0 (0x5c, 0x30 in ASCII) appears, the transmission of data starts. Go back to the normal command mode and wait for the next AT command.
- If the data contains the \<any>, it means that drop backslash symbol and only use <any> character.
- When sending \0, please use a backslash to escape it as \\0.
- You can use *AT+CIPTCP* command to configure socket options for each TCP connection. For example, setting <so_sndtimeo> to 5000 will enable TCP send operation to return results within 5 seconds, regardless of success or failure. This can save the time that the MCU waits for AT command response.

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

Set Command

Function:

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

Command:

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

Response:

```
<link ID>,CLOSED
```

```
OK
```

Execute Command

Function:

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

```
AT+CIPCLOSE
```

Response:

```
CLOSED
```

```
OK
```

Parameter

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

3.3.13 AT+CIPCONNPERSIST: Query or Set TCP/SSL Connection Persistence

Query Command

Function:

Query the persistence attributes of TCP/SSL connections.

Command:

```
AT+CIPCONNPERSIST?
```

Response:

```
+CIPCONNPERSIST:<persist_link>[...][,<persist_link>]
```

```
OK
```

Set Command

Function:

Set the persistence attributes of TCP/SSL connections.

Command:

```
// Single connection: (AT+CIPMUX=0)
AT+CIPCONNPERSIST=<persist_link>

// Multiple connections: (AT+CIPMUX=1)
AT+CIPCONNPERSIST=<link ID>,<persist_link>
```

Response:

```
OK
```

Parameters

- **<link ID>**: ID of the network connection to configure.
- **<persist_link>**: Whether to enable connection persistence. When disabled (default), ESP32 will actively close TCP/SSL connections on this netif immediately after the netif stops, disconnects, or loses its IP address; when enabled, AT will not actively close the connection, letting the protocol stack handle the connection state itself.
 - 0: Disable connection persistence (default).
 - 1: Enable connection persistence.

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

Execute Command

Command:

```
AT+CIFSR
```

Response:

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

```
OK
```

Parameters

- **<" APIP" >**: IPv4 address of ESP32 Wi-Fi softAP interface
- **<" APIP6LL" >**: Linklocal IPv6 address of ESP32 Wi-Fi softAP interface
- **<" APIP6GL" >**: Global IPv6 address of ESP32 Wi-Fi softAP interface

- <" **APMAC**" >: MAC address of ESP32 Wi-Fi softAP interface
- <" **STAIP**" >: IPv4 address of ESP32 Wi-Fi station interface
- <" **STAIP6LL**" >: Linklocal IPv6 address of ESP32 Wi-Fi station interface
- <" **STAIP6GL**" >: Global IPv6 address of ESP32 Wi-Fi station interface
- <" **STAMAC**" >: MAC address of ESP32 Wi-Fi station interface
- <" **ETHIP**" >: IPv4 address of ESP32 ethernet interface
- <" **ETHIP6LL**" >: Linklocal IPv6 address of ESP32 ethernet interface
- <" **ETHIP6GL**" >: Global IPv6 address of ESP32 ethernet interface
- <" **ETHMAC**" >: MAC address of ESP32 ethernet interface

Note

- Only when the ESP32 has the valid interface information can you query its IP address and MAC address.

3.3.15 AT+CIPMUX: Enable or disable Multiple Connections

Query Command

Function:

Query the connection type.

Command:

```
AT+CIPMUX?
```

Response:

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

Set Command

Function:

Set the connection type.

Command:

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

Response:

```
OK
```

Parameter

- <**mode**>: connection mode. Default: 0.
 - 0: Single connection.
 - 1: Multiple connections.

Notes

- This mode can only be changed after all connections are disconnected.
- If you want to set the multiple connections mode, ESP32 should be in the *Normal Transmission Mode* (*AT+CIPMODE=0*).

- If you want to set the single connection mode when the TCP/SSL server is running, you should delete the server first. (*AT+CIPSERVER=0*).
- When transmitting large amounts of data in parallel over multiple connections, memory peaks may be insufficient. Please test according to your product's actual situation and implement MCU handling for insufficient memory.

Example

```
AT+CIPMUX=1
```

3.3.16 AT+CIPSERVER: Delete or create a TCP/SSL Server

Query Command

Function:

Query the TCP/SSL server status.

Command:

```
AT+CIPSERVER?
```

Response:

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

Set Command

Command:

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

Response:

```
OK
```

Parameters

- **<mode>**:
 - 0: Delete a server.
 - 1: Create a server.
- **<param2>**: It means differently depending on the parameter <mode>:
 - If <mode> is 1, <param2> represents the port number. Default: 333.
 - If <mode> is 0, <param2> represents whether the server closes all connections. Default: 0.
 - * 0: Shutdown the server and keep existing connections.
 - * 1: Shutdown the server and close all connections.
- **<"type">**: server type: "TCP" , "TCPv6" , "SSL" , or "SSLv6" . Default: "TCP" .
- **<CA enable>**:
 - 0: Disable CA.
 - 1: Enable CA.
- **<netif>**: Specify the network interface for the server to listen on. Default: 0.
 - 0: all network interfaces

- 1: Wi-Fi station interface
- 2: Wi-Fi SoftAP interface
- 3: Ethernet interface

Notes

- A TCP/SSL server can only be created when multiple connections are activated (*AT+CIPMUX=1*).
- A server monitor will be created automatically when the server is created. Only one server can be created at most.
- When a client is connected to the server, it will take up one connection and be assigned an ID.
- To create a TCP/SSL server based on an IPv6 network, set *AT+CIPV6=1* first and obtain an IPv6 address.
- Parameters <"type"> and <CA enable> must be omitted when delete a server.
- To view the reason for server creation failure, please first run *AT+SYSLOG=1* to enable logging, then retry this command. If the command fails, AT will return a more detailed error code +ERRNO:<error_code> to help locate the problem.
- For TLS protocol version configuration, please refer to the [modify tls protocol version note](#).

Example

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

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

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

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

3.3.17 AT+CIPSERVERMAXCONN: Query or Set the Maximum Connections Allowed by a Server

Query Command

Function:

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

Command:

```
AT+CIPSERVERMAXCONN?
```

Response:

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

Set Command

Function:

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

Command:

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

Response:

```
OK
```

Parameter

- **<num>**: the maximum number of clients allowed to connect to the TCP/SSL server. Range: [1,5]. For how to change the upper limit of this range, please refer to the description of the [<link ID>](#) parameter of the *AT+CIPSTART* command.

Note

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

Example

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

3.3.18 AT+CIPMODE: Query or Set the Transmission Mode

Query Command**Function:**

Query the transmission mode.

Command:

```
AT+CIPMODE?
```

Response:

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

Set Command**Function:**

Set the transmission mode.

Command:

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

Response:

```
OK
```

Parameter

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

Notes

- The configuration changes will NOT be saved in flash.
- After the ESP32 enters the Network *Passthrough Receiving Mode*, no Bluetooth function can be used.

Example

```
AT+CIPMODE=1
```

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

Query Command**Function:**

Query the local TCP/SSL server timeout.

Command:

```
AT+CIPSTO?
```

Response:

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

Set Command**Function:**

Set the local TCP/SSL server timeout.

Command:

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

Response:

```
OK
```

Parameter

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

Notes

- When a TCP/SSL client does not communicate with the ESP32 server within the <time> value, the server will terminate this connection.
- If you set <time> to 0, the connection will never timeout. This configuration is not recommended.
- When the client initiates a communication with the server or the server initiate a communication with the client within the set time, the timer will restart. After the timeout expires, the client is closed.

Example

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

3.3.20 AT+CIPSNTPCFG: Query or Set the Time Zone and the SNTP Server

Query Command

Command:

```
AT+CIPSNTPCFG?
```

Response:

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

Set Command

Command:

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

Response:

```
OK
```

Parameters

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

- **<" SNTP server3" >**: the third SNTP server.

Note

- If the three SNTP servers are not configured, one of the following default servers will be used: "cn.ntp.org.cn", "ntp.sjtu.edu.cn", and "us.pool.ntp.org".
- For the query command, **<timezone>** parameter in the response may be different from the **<timezone>** parameter in set command. Because the **<timezone>** parameter supports the second UTC offset format, for example, set **AT+CIPSNTPCFG=1,015**, for query command, ESP-AT ignores the leading zero of the **<timezone>** parameter, and the valid value is 15. It does not belong to the first format, so it is parsed according to the second UTC offset format, that is, UTC+00:15, that is, **timezone** is 0 in the response.
- Since SNTP operates over the UDP protocol for sending requests and receiving replies, if there is packet loss in the network, the time synchronization of ESP32 may be delayed. Once the AT command output shows **+TIME_UPDATED**, it indicates that the time has been synchronized, and you can then send the **AT+CIPSNTPTIME?** command to query the current time.

Example

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

// Enable SNTP server, set timezone to New York of the United States (UTC-05:00)
AT+CIPSNTPCFG=1,-5,"0.pool.ntp.org","time.google.com"
or
AT+CIPSNTPCFG=1,-500,"0.pool.ntp.org","time.google.com"

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

3.3.21 AT+CIPSNTPTIME: Query the SNTP Time

Query Command

Command:

```
AT+CIPSNTPTIME?
```

Response:

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

Note

- The asctime style time is defined at [asctime man page](#).
- When ESP32 enters Light-sleep or Deep-sleep mode and then wakes up, the system time may become inaccurate. It is recommended to resend the **AT+CIPSNTPCFG** command to obtain the new time from the NTP server.
- The time obtained from SNTP is stored in the RTC area, so it will not be lost after a software reset (chip does not lose power).

Example

```
AT+CWMODE=1
AT+CWJAP="1234567890","1234567890"
AT+CIPSNTPCFG=1,8,"cn.ntp.org.cn","ntp.sjtu.edu.cn"
AT+CIPSNTPTIME?
+CIPSNTPTIME:Tue Oct 19 17:47:56 2021
OK

or

AT+CWMODE=1
AT+CWJAP="1234567890","1234567890"
AT+CIPSNTPCFG=1,530
AT+CIPSNTPTIME?
+CIPSNTPTIME:Tue Oct 19 15:17:56 2021
OK
```

3.3.22 AT+CIPSNTPTINTV: Query or Set the SNTP time synchronization interval**Query Command****Command:**

```
AT+CIPSNTPTINTV?
```

Response:

```
+CIPSNTPTINTV:<interval second>
OK
```

Set Command**Command:**

```
AT+CIPSNTPTINTV=<interval second>
```

Response:

```
OK
```

Parameters

- **<interval second>**: the SNTP time synchronization interval. Unit: second. Range: [15,4294967].

Notes

- It configures interval for synchronization, which means that it sets interval how often ESP32 connects to NTP servers to get new time.

Example

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

OK

// synchronize SNTP time every hour
AT+CIPSNTPINTV=3600

OK
```

3.3.23 AT+CIPFWVER: Query the Existing AT Firmware Version on the Server

Query Command

Function:

Query the existing ESP32 AT firmware version on the server.

Command:

```
AT+CIPFWVER?
```

Response:

```
+CIPFWVER:<"version">

OK
```

Parameters

- <"version">: ESP32 AT firmware version.

Notes

- When selecting the OTA version to be upgraded, it is strongly not recommended to upgrade from a high version to a low version.

3.3.24 AT+CIUPDATE: Upgrade Firmware Through Network

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

Query Command

Function:

Query ESP32 upgrade status.

Command:

```
AT+CIUPDATE?
```

Response:

```
+CIUPDATE:<state>

OK
```

Execute Command

Function:

Upgrade OTA the latest version of firmware via TCP from the server in blocking mode.

Command:

```
AT+CIUPDATE
```

Response:

Please refer to the [response](#) in the set command.

Set Command

Function:

Upgrade the specified version of firmware from the server.

Command:

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

Response:

If OTA succeeds in blocking mode, the system returns:

```
+CIPUPDATE:1
+CIPUPDATE:2
+CIPUPDATE:3
+CIPUPDATE:4

OK
```

If OTA succeeds in non-blocking mode, the system returns:

```
OK
+CIPUPDATE:1
+CIPUPDATE:2
+CIPUPDATE:3
+CIPUPDATE:4
```

If OTA fails in blocking mode, the system returns:

```
+CIPUPDATE:<state>

ERROR
```

If OTA fails in non-blocking mode, the system returns:

```
OK
+CIPUPDATE:<state>
+CIPUPDATE:-1
```

Parameters

- **<ota mode>:**
 - 0: OTA via HTTP.
 - 1: OTA via HTTPS. If it does not work, please check whether `./build.py menuconfig>Component config>AT>OTA based upon ssl` is enabled. For more information, please refer to [Compile ESP-AT Project Locally](#).

- **<version>**: AT version, such as, v1.2.0.0, v1.1.3.0, v1.1.2.0.
- **<"firmware name">**: firmware to upgrade, such as, ota, mqtt_ca, client_ca or other custom partition in at_customize.csv.
- **<nonblocking>**:
 - 0: OTA by blocking mode. In this mode, you can not send AT command until OTA completes successfully or fails.
 - 1: OTA by non-blocking mode. You need to manually restart after upgrade done (+CIPUPDATE:4).
- **<state>**:
 - 1: Server found.
 - 2: Connected to the server.
 - 3: Got the upgrade version.
 - 4: Upgrade done.
 - -1: OTA fails in non-blocking mode.

Notes

- The speed of the upgrade depends on the network status.
- If the upgrade fails due to unfavorable network conditions, AT will return ERROR. Please wait for some time before retrying.
- If you use Espressif's AT BIN, AT+CIUPDATE will download a new AT BIN from the Espressif Cloud.
- If you use a user-compiled AT BIN, you need to implement your own AT+CIUPDATE FOTA function or use [AT+USEROTA](#) or [AT+WEBSEVER](#) command. ESP-AT project provides an example of [FOTA](#).
- After you upgrade the AT firmware, you are suggested to call the command [AT+RESTORE](#) to restore the factory default settings.
- The timeout of OTA process is 3 minutes.
- The response OK in non-blocking mode does not necessarily come before the response +CIPUPDATE:<state>. It may be output before +CIPUPDATE:<state> or after it.
- Downgrading to an older version is not recommended due to potential compatibility issues and the risk of operational failure. If you still prefer downgrading to an older version, please test and verify the functionality based on your product.
- Please refer to [How to Implement OTA Upgrade](#) for more OTA commands.

Example

```
AT+CWMODE=1
AT+CWJAP="1234567890", "1234567890"
AT+CIUPDATE
AT+CIUPDATE=1
AT+CIUPDATE=1, "v1.2.0.0"
AT+CIUPDATE=1, "v2.2.0.0", "mqtt_ca"
AT+CIUPDATE=1, "v2.2.0.0", "ota", 1
AT+CIUPDATE=1,,,1
AT+CIUPDATE=1,, "ota", 1
AT+CIUPDATE=1, "v2.2.0.0",, 1
```

3.3.25 AT+CIPDINFO: Set “+IPD” Message Mode

Query Command

Command:

```
AT+CIPDINFO?
```

Response:

```
+CIPDINFO:true  
OK
```

or

```
+CIPDINFO:false  
OK
```

Set Command

Command:

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

Response:

```
OK
```

Parameters

- **<mode>**:
 - 0: Does not show the remote host and port in “+IPD” and “+CIPRECVDATA” messages.
 - 1: Show the remote host and port in “+IPD” and “+CIPRECVDATA” messages.

Example

```
AT+CIPDINFO=1
```

3.3.26 AT+CIPSSLCONF: Query or Set SSL Clients

Query Command

Function:

Query the configuration of each connection where the ESP32 runs as an SSL client.

Command:

```
AT+CIPSSLCONF?
```

Response:

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

Set Command

Command:

```
// Single connection: (AT+CIPMUX=0)  
AT+CIPSSLCONF=<auth_mode>[,<pki_number>][,<ca_number>]  
  
// Multiple connections: (AT+CIPMUX=1)  
AT+CIPSSLCONF=<link ID>,<auth_mode>[,<pki_number>][,<ca_number>]
```

Response:

OK

Parameters

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

Notes

- If you want this configuration to take effect immediately, run this command before establishing an SSL connection.
- The configuration changes will be saved in the NVS partition. If you set the command [AT+SAVETRANSLINK](#) to enter SSL *Passthrough Mode* on power-up, the ESP32 will establish an SSL connection based on this configuration when powered up next time.
- If you want to use your own certificate at runtime, use the [AT+SYSMFG](#) command to update the SSL certificate (for detailed steps, please refer to [AT+SYSMFG command examples](#)). If you want to pre-burn your own certificate, please refer to [How to Update PKI Configuration](#).
- If **<auth_mode>** is configured to 2 or 3, in order to check the server certificate validity period, please make sure ESP32 has obtained the current time before sending the [AT+CIPSTART](#) command. (You can send [AT+CIPSNTPCFG](#) command to configure SNTP and obtain the current time, and send [AT+CIPSNTIME?](#) command to query the current time.)

3.3.27 AT+CIPSSLCCIPHER: Query or Set the Cipher Suite of the SSL Client

Query Command**Function:**

Query the cipher suites supported by ESP32 as an SSL client.

Command:

AT+CIPSSLCCIPHER?

Response:

+CIPSSLCCIPHER:<idx>,<cipher_suite>

OK

Set Command**Command:**

```
// Single connection: (AT+CIPMUX=0)
AT+CIPSSLCCIPHER=<counts>[,<cipher_suite>][...][,<cipher_suite>]

// Multiple connections: (AT+CIPMUX=1)
AT+CIPSSLCCIPHER=<link ID>,<counts>[,<cipher_suite>][...][,<cipher_suite>]
```

Response:

```
OK
```

Parameters

- **<idx>**: the index number of the cipher suite, starting from 0.
- **<link ID>**: network connection ID (0 ~ max). For multiple connections, if the value is max, it means all connections. The default value is 5.
- **<counts>**: number of cipher suites. The maximum value is limited by the maximum command length of 256 bytes.
 - 0: Clear the configured cipher suites.
 - Others: Set the number of cipher suites.
- **<cipher_suite>**: the cipher suite in ClientHello. The corresponding values are defined in [ssl_ciphersuites.h](#).

Notes

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

Example

```
// Single connection: (AT+CIPMUX=0), cipher suites are TLS_ECDHE_ECDSA_WITH_AES_
↪128_CBC_SHA256 and TLS_ECDHE_ECDSA_WITH_AES_256_GCM
AT+CIPSSLCCIPHER=2,0xC023,0xC0AD

// Multiple connections: (AT+CIPMUX=1), cipher suites are TLS_ECDHE_ECDSA_WITH_AES_
↪128_CBC_SHA256 and TLS_ECDHE_ECDSA_WITH_AES_256_GCM
AT+CIPSSLCCIPHER=0,2,0xC023,0xC0AD
```

3.3.28 AT+CIPSSLCCN: Query or Set the Common Name of the SSL Client**Query Command****Function:**

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

Command:

```
AT+CIPSSLCCN?
```

Response:

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

Set Command

Command:

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

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

Response:

```
OK
```

Parameters

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

Note

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

3.3.29 AT+CIPSSLCSNI:Query or Set SSL Client Server Name Indication (SNI)

Query Command

Function:

Query the SNI configuration of each connection.

Command:

```
AT+CIPSSLCSNI?
```

Response:

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

Set Command

Command:

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

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

Response:

```
OK
```

Parameters

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

Notes

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

3.3.30 AT+CIPSSLCALPN: Query or Set SSL Client Application Layer Protocol Negotiation (ALPN)

Query Command

Function:

Query the ALPN configuration of each connection where the ESP32 runs as an SSL client.

Command:

```
AT+CIPSSLCALPN?
```

Response:

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

```
OK
```

Set Command

Command:

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

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

Response:

```
OK
```

Parameters

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

Note

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

3.3.31 AT+CIPSSLCPSK: Query or Set SSL Client Pre-shared Key (PSK) in String Format

Query Command**Function:**

Query the PSK configuration of each connection where the ESP32 runs as an SSL client.

Command:

```
AT+CIPSSLCPSK?
```

Response:

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

Set Command**Command:**

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

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

Response:

```
OK
```

Parameters

- **<link ID>**: ID of the connection (0 ~ max). For single connection, <link ID> is 0. For multiple connections, if the value is max, it means all connections, max is 5 by default.
- **<"psk">**: PSK identity in string format. Maximum length: 32. Please use [AT+CIPSSLCPSKHEX](#) command if your <"psk"> parameter contains \0 characters.
- **<"hint">**: PSK hint. Maximum length: 32.

Notes

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

3.3.32 AT+CIPSSLCPSKHEX: Query or Set SSL Client Pre-shared Key (PSK) in Hexadecimal Format

Note

- Similar to the [AT+CIPSSLCP SK](#) command, this command also sets or queries the SSL Client PSK, but its <"psk"> is in hexadecimal format rather than in string format. So, \0 in the <"psk"> parameter means 00.

Example

```
// Single connection: (AT+CIPMUX=0), PSK identity is "psk", PSK hint is "myhint".
AT+CIPSSLCP SKHEX="70736b", "myhint"

// Multiple connections: (AT+CIPMUX=1), PSK identity is "psk", PSK hint is "myhint
↔ ".
AT+CIPSSLCP SKHEX=0, "70736b", "myhint"
```

3.3.33 AT+CIPRECONNINTV: Query or Set the TCP/UDP/SSL reconnection Interval for the Network Passthrough Mode

Query Command**Function:**

Query the automatic connect interval for the Network *Passthrough Mode*.

Command:

```
AT+CIPRECONNINTV?
```

Response:

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

Set Command**Function:**

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

Command:

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

Response:

```
OK
```

Parameter

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

Note

- The configuration changes will be saved in the NVS partition if [AT+SYSSTORE=1](#).

Example

```
AT+CIPRECONNINTV=10
```

3.3.34 AT+CIPRECVMODE: Query or Set Socket Receiving Mode

Query Command

Function:

Query the socket receiving mode.

Command:

```
AT+CIPRECVMODE?
```

Response:

```
+CIPRECVMODE:<link ID>,<mode>
OK
```

Set Command

Command:

```
// Single connection: (AT+CIPMUX=0)
AT+CIPRECVMODE=<mode>

// Multiple connections: (AT+CIPMUX=1)
AT+CIPRECVMODE=<link ID>,<mode>
```

Response:

```
OK
```

Parameter

- **<link ID>**: ID of the connection (0 ~ max). For a single connection, <link ID> is 0. For multiple connections, if the value is max, it means all connections. Max is 5 by default.
- **<mode>**: the receive mode of socket data. Default: 0.
 - 0: Active mode. ESP-AT will send all the received socket data instantly to the host MCU with the header “+IPD”. (The socket receive window is 5760 bytes by default. The maximum valid bytes sent to MCU is 2920 bytes each time.)
 - 1: Passive mode. ESP-AT will keep the received socket data in an internal buffer (socket receive window, 5760 bytes by default), and wait for the host MCU to read. If the buffer is full, the socket transmission will be blocked for TCP/SSL connections, or data will be lost for UDP connections.

Notes

- The configuration can not be used in the Network *Passthrough Mode*. If it is a UDP transmission in passive mode, data will be lost when the buffer is full.
- When ESP-AT receives socket data in passive mode, it will prompt the following messages in different scenarios:
 - For multiple connections mode (AT+CIPMUX=1), the message is +IPD, <link ID>, <len>.
 - For single connection mode (AT+CIPMUX=0), the message is +IPD, <len>.

- `<len>` is the total length of socket data in the buffer.
- You should read data by running `AT+CIPRECVDATA` once there is a `+IPD` reported. Otherwise, the next `+IPD` will not be reported to the host MCU until the previous `+IPD` has been read.
- In case of disconnection, the buffered socket data will still be there and can be read by the MCU until you send `AT+CIPCLOSE` (AT as client) or `AT+CIPSERVER=0,1` (AT as server). In other words, if `+IPD` has been reported, the message `CLOSED` of this connection will never come until you send `AT+CIPCLOSE` or `AT+CIPSERVER=0,1` or read all data by command `AT+CIPRECVDATA`.
- When a large amount of network data is expected to be received and the MCU cannot process it timely, you can refer to [example](#) and use the passive receive data mode.

Example

```
// Set passive mode in single connection mode
AT+CIPRECVMODE=1

// Set all connections to passive mode in multiple connections mode
AT+CIPRECVMODE=5,1
```

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

Set Command

Command:

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

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

Response:

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

or

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

Parameters

- `<link_id>`: connection ID in multiple connections mode.
- `<len>`: the max value is `0x7ffffff`. If the actual length of the received data is less than `len`, the actual length will be returned.
- `<actual_len>`: length of the data you actually obtain.
- `<data>`: the data you want to obtain.
- `<"remote IP">`: string parameter showing the remote IPv4 address or IPv6 address, enabled by the command `AT+CIPDINFO=1`.
- `<remote port>`: the remote port number, enabled by the command `AT+CIPDINFO=1`.

Note

- The command needs to be executed in passive receive mode. Otherwise, `ERROR` is returned. You can verify whether AT is in passive receive mode by using the `AT+CIPRECVMODE` command.

- When this command is executed without any data available to read, it will directly return ERROR. You can verify if there is readable data at that time by using the [AT+CIPRECVDATA?](#) command.
- When executing the command `AT+CIPRECVDATA=<len>`, at least `<len> + 128` bytes of memory are required. You can use the command [AT+SYSRAM?](#) to check the current available memory. When insufficient memory leads to a memory allocation failure, this command will also return ERROR. You can review the [AT log output](#) for `alloc fail` or similar print messages to confirm whether a memory allocation failure has occurred.

Example

```
AT+CIPRECVDTYPE=1

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

3.3.36 AT+CIPRECVDLEN: Obtain Socket Data Length in Passive Receiving Mode

Query Command

Function:

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

Command:

```
AT+CIPRECVDLEN?
```

Response:

```
+CIPRECVDLEN:<data length of link>[...][,<data length of link>]

OK
```

Parameters

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

Note

- For SSL connections, the data length returned by ESP-AT may be less than the actual data length.

Example

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

3.3.37 AT+PING: Ping the Remote Host

Set Command

Function:

Ping the remote host.

Command:

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

Response:

```
+PING:<time>
```

```
OK
```

or

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

Parameters

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

Notes

- To ping a remote host based on an IPv6 network, do as follows:
 - Make sure that the AP supports IPv6
 - Set [AT+CIPV6=1](#)
 - Obtain an IPv6 address through the [AT+CWJAP](#) command
 - (Optional) Check whether ESP32 has obtained an IPv6 address using the [AT+CIPSTA?](#) command
- If the remote host is a domain name string, ping will first resolve the domain name (IPv4 address preferred) from DNS (domain name server), and then ping the remote IP address.

Example

```
AT+PING="192.168.1.1"  
AT+PING="www.baidu.com"  
  
// China Future Internet Engineering Center  
AT+PING="240c::6666"
```

3.3.38 AT+CIPDNS: Query or Set DNS Server Information

Query Command

Function:

Query the current DNS server information.

Command:

```
AT+CIPDNS?
```

Response:

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

Set Command**Function:**

Set DNS server information.

Command:

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

Response:

```
OK
```

or

```
ERROR
```

Parameters

- **<enable>**: configure DNS server settings
 - 0: Enable automatic DNS server settings from DHCP. The DNS will be restored to 208.67.222.222, 114.114.114.114 and 8.8.8.8. Only when the ESP32 station completes the DHCP process, the DNS server of the ESP32 station could be updated.
 - 1: Enable manual DNS server settings. If you do not set a value for <DNS IPx>, it will use 208.67.222.222, 114.114.114.114 and 8.8.8.8 by default.
- **<" DNS IP1" >**: the first DNS server IP address. For the set command, this parameter only works when you set <enable> to 1, i.e. enable manual DNS settings. If you set <enable> to 1 and a value for this parameter, the ESP-AT will return this parameter as the current DNS setting when you run the query command.
- **<" DNS IP2" >**: the second DNS server IP address. For the set command, this parameter only works when you set <enable> to 1, i.e. enable manual DNS settings. If you set <enable> to 1 and a value for this parameter, the ESP-AT will return this parameter as the current DNS setting when you run the query command.
- **<" DNS IP3" >**: the third DNS server IP address. For the set command, this parameter only works when you set <enable> to 1, i.e. enable manual DNS settings. If you set <enable> to 1 and a value for this parameter, the ESP-AT will return this parameter as the current DNS setting when you run the query command.

Notes

- The configuration changes will be saved in the NVS partition if [AT+SYSTORE=1](#).
- The three parameters cannot be set to the same server.
- When <enable> is set to 0, the DNS server may change according to the configuration of the router which the ESP32 is connected to.
- When <enable> is set to 1, the manually set DNS servers will always take effect, and the DNS configuration obtained from the router via DHCP will not override the manually set values. To restore the use of DHCP-assigned DNS, execute AT+CIPDNS=0 to re-enable automatic acquisition.

Example

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

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

3.3.39 AT+MDNS: Configure the mDNS Function

- *Device Starts an mDNS Service*
- *Query mDNS Services in the Local Network*

Device Starts an mDNS Service**Set Command Command:**

```
AT+MDNS=<mode>[, <"hostname">, <"service_type">, <port>] [, <"instance">] [, <"proto">] [,
↪<txt_number>] [, <"key">, <"value">] [...]
```

Response:

```
OK
```

Parameters

- **<mode>**:
 - 1: Start an mDNS service. The following parameters need to be set.
 - 0: Disable the mDNS function. Please do not set the following parameters.
- **<"hostname">**: mDNS host name.
- **<"service_type">**: mDNS service type.
- **<port>**: mDNS service port.
- **<"instance">**: mDNS instance name. Default: <"hostname">.
- **<"proto">**: mDNS service protocol. Recommended values: `_tcp` or `_udp`. Default: `_tcp`.
- **<txt_number>**: the number of key-value pairs in the TXT record. Range: [1,20].
- **<"key">**: key of the TXT record.
- **<"value">**: value of the TXT record.
- [...]: repeat the key-value pairs of TXT record according to the `<txt_number>`.

Example

```
// Start an mDNS service. Set the hostname to "espressif", service type to "_iot",
↪and port to 8080.
AT+MDNS=1, "espressif", "_iot", 8080

// Disable mDNS function
AT+MDNS=0
```

Detailed examples can be found in: [mDNS Example](#).

Query mDNS Services in the Local Network

Set Command Command:

```
// Query PTR records in the local network
AT+MDNS=<mode>,<type>,<"service">,<"proto">[,<timeout>][,<max_results>]

// Query TXT, SRV, or TXT + SRV records in the local network
AT+MDNS=<mode>,<type>,<"instance">,<"service">,<"proto">[,<timeout>][,<max_results>
↪]

// Query A, AAAA, or A + AAAA records in the local network
AT+MDNS=<mode>,<type>,<"hostname">[,<timeout>][,<max_results>]
```

Response:

```
+MDNS:<data_len>,
IF=<netif>
IP=<ip_type>
PTR=<instance>
SRV=<hostname>:<port>
TXT=<key>=<value>
A=<ip4>
AAAA=<ip6>

OK
```

Command Parameters

- **<mode>**:
 - 2: Query mDNS services in the local network, subsequent parameters are required
 - 0: Disable mDNS function, subsequent parameters are not required
- **<type>**: Query type.
 - 0: PTR record
 - 1: TXT record
 - 2: SRV record
 - 3: TXT + SRV record
 - 4: A record
 - 5: AAAA record
 - 6: A + AAAA record
- **<" instance">**: mDNS instance name. Maximum length: 64 bytes, any length exceeding this will be automatically truncated.
- **<" hostname">**: Host name. Maximum length: 64 bytes, any length exceeding this will be automatically truncated.
- **<"service">**: mDNS service type. Maximum length: 64 bytes, any length exceeding this will be automatically truncated.
- **<" proto">**: mDNS service protocol. Maximum length: 64 bytes, any length exceeding this will be automatically truncated. Recommended values: `_tcp` or `_udp`.
- **<timeout>**: Query timeout in milliseconds. Default: 5000. Range: [1000,180000]. If the number of results does not reach `<max_results>`, the query results will be returned after the timeout.
- **<max_results>**: Maximum number of query results. Default: 1. Range: [1,255]. If the number of results does not reach `<max_results>`, the query results will be returned after the timeout. Otherwise, the results will be returned immediately.

Response Parameters

- **<data_len>**: Length of the query result.
- **<netif>**: Network interface where the mDNS service is located.
 - 1: Wi-Fi Station interface

- 2: Wi-Fi SoftAP interface
 - 3: Ethernet interface
- **<ip_type>**: IP address type of the network interface.
 - 0: IPv4 address
 - 1: IPv6 address
- **<instance>**: mDNS instance name.
- **<hostname>**: Host name.
- **<port>**: mDNS service port.
- **<key>**: Key of the TXT record.
- **<value>**: Value of the TXT record.
- **<ip4>**: IPv4 address where the mDNS service is located.
- **<ip6>**: IPv6 address where the mDNS service is located.

Example

```
// Publish mDNS service on PC using avahi-publish
avahi-publish -s my_instance _my_printer._tcp 35 "version=v4.1.0.0"

// Query PTR records in the local network via AT command
AT+MDNS=2,0,"_my_printer","_tcp"

// Response result
+MDNS:234,
IF=1
IP=0
PTR=my_instance
SRV=espressif-1.local:35
TXT=version=v4.1.0.0
A=192.168.200.249
AAAA=240e:1234:204c:bd02:d80f:8c45:3576:2574
AAAA=240e:1234:204c:bd02:f043:47f0:7c39:0002
AAAA=240e:1234:204c:bd02:2971:f412:14c6:fa00

OK

// Disable mDNS function
AT+MDNS=0
```

3.3.40 AT+CIPTCPOPT: Query or Set the Socket Options

Query Command

Function:

Query current socket options.

Command:

```
AT+CIPTCPOPT?
```

Response:

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

Set Command

Command:


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

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

Response:

OK

or

ERROR

Parameters

- **<link_id>**: ID of the connection (0 ~ max). For multiple connections, if the value is max, it means all connections. By default, max is 5.
- **<so_linger>**: configure the `SO_LINGER` options for the socket (refer to [SO_LINGER description](#)). Unit: second. Default: -1.
 - = -1: off
 - = 0: on, linger time = 0
 - > 0: on, linger time = <so_linger>
- **<tcp_nodelay>**: configure the `TCP_NODELAY` option for the socket (refer to [TCP_NODELAY description](#)). Default: 0.
 - 0: disable `TCP_NODELAY`
 - 1: enable `TCP_NODELAY`
- **<so_sndtimeo>**: configure the `SO_SNDTIMEO` option for socket (refer to [SO_SNDTIMEO description](#)). Unit: millisecond. Default: 0.
- **<keep_alive>**: configure the [SO_KEEPAKIVE](#) option for socket. Unit: second.
 - Range: [0,7200].
 - * 0: Disable keep-alive function (default).
 - * 1 ~ 7200: Enable keep-alive function. [TCP_KEEPIPLE](#) value is <keep_alive>, [TCP_KEEPIPLVL](#) value is 1, and [TCP_KEEPCNT](#) value is 3.
 - This parameter of this command is the same as the <keep_alive> parameter of [AT+CIPSTART](#) command. It always takes the value set later by either of the two commands. If it is omitted or not set, the last configured value is used by default.

Notes

- The configuration changes will be saved in the NVS partition if `AT+SYSSTORE=1`.
- Before configuring these socket options, **please make sure you fully understand the function of them and the possible impact after configuration**.
- It is not recommended to set a large value for the `SO_LINGER` option. For example, if `SO_LINGER` is set to 60, the [AT+CIPCLOSE](#) command may block for up to 60 seconds if ESP32 does not receive a TCP FIN packet from the remote TCP peer due to network issues. During this period, ESP32 will be unable to respond to any other AT commands. Therefore, it is recommended to keep the `SO_LINGER` option at its default value.
- The `TCP_NODELAY` option is used for situations with low throughput but high real-time requirements. When enabled, [LwIP](#) will speed up TCP transmission. However, in poor network conditions, this may lead to increased retransmissions and reduced throughput. Therefore, it is recommended to keep the `TCP_NODELAY` option at its default value.
- The `SO_SNDTIMEO` option is used for situations where the keepalive parameter is not configured in [AT+CIPSTART](#) command. After this option is configured, [AT+CIPSEND](#), [AT+CIPSENDL](#), and [AT+CIPSENDEX](#) commands will exit within this timeout, regardless of whether data are sent successfully or not. Here, `SO_SNDTIMEO` is recommended to be set to 5 ~ 10 seconds.
- The `SO_KEEPAKIVE` option is used for actively and regularly detecting whether the connection is disconnected. It is generally recommended to configure this option when AT is used as a TCP server. After this

option is configured, additional network bandwidth will be cost. Recommended value of SO_KEEPALIVE should be not less than 60 seconds.

3.4 Bluetooth® Low Energy AT Commands

- *Introduction*
- *AT+BLEINIT*: Bluetooth LE initialization.
- *AT+BLEADDR*: Query or set Bluetooth LE device address.
- *AT+BLENAME*: Query or set Bluetooth LE device name.
- *AT+BLESCANPARAM*: Query or set parameters of Bluetooth LE scanning.
- *AT+BLESCAN*: Enable Bluetooth LE scanning.
- *AT+BLESCANRSPDATA*: Set Bluetooth LE scan response.
- *AT+BLEADVPARAM*: Query or set parameters of Bluetooth LE advertising.
- *AT+BLEADVDATA*: Set Bluetooth LE advertising data.
- *AT+BLEADVDATAEX*: Automatically set Bluetooth LE advertising data.
- *AT+BLEADVSTART*: Start Bluetooth LE advertising.
- *AT+BLEADVSTOP*: Stop Bluetooth LE advertising.
- *AT+BLECONN*: Establish Bluetooth LE connection.
- *AT+BLECONNPARAM*: Query or update parameters of Bluetooth LE connection.
- *AT+BLEDISCONN*: End Bluetooth LE connection.
- *AT+BLEDATALEN*: Set Bluetooth LE data packet length.
- *AT+BLECFGMTU*: Set Bluetooth LE MTU length.
- *AT+BLEGATTSSRVCRE*: Generic Attributes Server (GATTS) creates services.
- *AT+BLEGATTSSRVSTART*: GATTS starts services.
- *AT+BLEGATTSSRVSTOP*: GATTS Stops Services.
- *AT+BLEGATTSSRV*: GATTS discovers services.
- *AT+BLEGATTSSCHAR*: GATTS discovers characteristics.
- *AT+BLEGATTSENTFY*: Notify a client of the value of a characteristic value from the server.
- *AT+BLEGATTSSIND*: Indicate the characteristic value from the server to a client.
- *AT+BLEGATTSSSETATTR*: GATTS sets characteristics.
- *AT+BLEGATTCPRIMSRV*: Generic Attributes Client (GATTC) discovers primary services.
- *AT+BLEGATTCCINCLSRV*: GATTC discovers included services.
- *AT+BLEGATTCCCHAR*: GATTC discovers characteristics.
- *AT+BLEGATTCCRD*: GATTC reads characteristics.
- *AT+BLEGATTCCWR*: GATTC writes characteristics.
- *AT+BLESPPCFG*: Query or set Bluetooth LE SPP parameters.
- *AT+BLESP*: Enter Bluetooth LE SPP mode.
- *AT+SAVETRANSLINK*: Set whether to enter Bluetooth LE *Passthrough Mode* on power-up.
- *AT+BLESECPARAM*: Query or set Bluetooth LE encryption parameters.
- *AT+BLEENC*: Initiate Bluetooth LE encryption request.
- *AT+BLEENCRSP*: Respond to the pairing request from the peer device.
- *AT+BLEKEYREPLY*: Reply the key value to the peer device.
- *AT+BLECONFREPLY*: Reply the confirm value to the peer device in the legacy connection stage.
- *AT+BLEENCDEV*: Query bonded Bluetooth LE encryption device list.
- *AT+BLEENCCLEAR*: Clear Bluetooth LE encryption device list.
- *AT+BLESETKEY*: Set Bluetooth LE static pair key.
- *AT+BLEHIDINIT*: Bluetooth LE Human Interface Device (HID) profile initialization.
- *AT+BLEHIDKB*: Send Bluetooth LE HID keyboard information.
- *AT+BLEHIDMUS*: Send Bluetooth LE HID mouse information.
- *AT+BLEHIDCONSUMER*: Send Bluetooth LE HID consumer information.
- *AT+BLUFI*: Start or Stop BluFi.
- *AT+BLUFINAME*: Query or set BluFi device name.
- *AT+BLUFISEND*: Send user-customized data over BluFi.

- [AT+BLERDRSSI](#): Query the current connection RSSI.
- [AT+BLEWL](#): Set the white list.

3.4.1 Introduction

Currently, AT firmware for ESP32 series supports [Bluetooth® Core Specification Version 4.2](#).

Important: The default AT firmware supports all the AT commands mentioned on this page. If you need to modify the commands supported by ESP32 by default, please compile the ESP-AT project by following the steps in [Compile ESP-AT Project Locally](#) documentation. In the project configuration during the fifth step, make the following selections (Each item below is independent. Choose it according to your needs):

- Disable BluFi commands: Component config>AT>AT blufi command support
 - Disable Bluetooth LE commands: Component config>AT>AT ble command support
 - Disable Bluetooth LE HID commands: Component config>AT>AT ble hid command support
-

3.4.2 AT+BLEINIT: Bluetooth LE Initialization

Query Command

Function:

Check the initialization status of Bluetooth LE.

Command:

```
AT+BLEINIT?
```

Response:

If Bluetooth LE is initialized, AT will return:

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

If Bluetooth LE is not initialized, AT will return:

```
+BLEINIT:0  
OK
```

Set Command

Function:

Initialize the role of Bluetooth LE.

Command:

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

Response:

```
OK
```

Parameter

- **<init>**:
- 0: deinit Bluetooth LE
- 1: client role
- 2: server role
- 3: dual role (client and server)

Notes

- To achieve better performance, it is recommended to disable SoftAP by sending the [AT+CWMODE=0/1](#) command before using Bluetooth LE function. For more details, please refer to the [RF Coexistence](#) documentation.
- Before using other Bluetooth LE AT commands, you should run this command first to trigger the initialization process.
- The Bluetooth LE role cannot be changed directly after initialization. To switch roles, first execute [AT+BLEINIT=0](#) to deinitialize Bluetooth LE, or execute [AT+RST](#) to restart the system, then re-initialize the Bluetooth LE role.
- Before you deinitialize the Bluetooth stack, it is recommended to stop broadcasting, stop scanning, and disconnect all existing connections.
- If Bluetooth LE is initialized, [AT+CIPMODE](#) cannot be set to 1.
- In dual mode, both server-specific and client-specific commands are available.
- In dual-mode, BLE passthrough-related commands and BLE auto-passthrough at startup are not supported.
- In dual mode, it is possible to set advertising parameters, query advertising parameters, and start or stop advertising.
- In dual mode, it is possible to set scanning parameters, query scanning parameters, and start or stop scanning.
- In dual mode, it is possible to initiate all connection-related parameter updates and query connection parameters.
- In dual mode, the total number of connections and advertisements combined is limited to 3.
- Service Creation: After initializing BLE as a server role or dual role (set **<init>** to 2 or 3), please execute [AT+BLEGATTSSRVCRE](#) and [AT+BLEGATTSSRVSTART](#) in sequence to create and start GATT services.

Example

```
AT+BLEINIT=1
```

3.4.3 AT+BLEADDR: Query or Set Bluetooth LE Device Address

Query Command

Function:

```
Query the Bluetooth LE Public Address.
```

Command:

```
AT+BLEADDR?
```

Response:

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

Set Command

Function:

Set the Bluetooth LE address type.

Command:

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

Response:

```
OK
```

Parameter

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

Note

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

Example

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

3.4.4 AT+BLENAME: Query or Set Bluetooth LE Device Name

Query Command

Function:

Query the Bluetooth LE device name.

Command:

```
AT+BLENAME?
```

Response:

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

Set Command

Function:

Set the Bluetooth LE device name.

Command:

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

Response:

```
OK
```

Parameter

- **<device_name>**: the Bluetooth LE device name. The maximum length is 32. Default: “ESP-AT” .

Note

- The configuration changes will be saved in the NVS partition if [AT+SYSTORE=1](#).
- After setting the device name with this command, it is recommended that you execute the [AT+BLEADVDATA](#) command to add the device name into the advertising data.

Example

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

3.4.5 AT+BLESCANPARAM: Query or Set Parameters of Bluetooth LE Scanning

Query Command

Function:

Query the parameters of Bluetooth LE scanning.

Command:

```
AT+BLESCANPARAM?
```

Response:

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

Set Command

Function:

Set the parameters of Bluetooth LE scanning.

Command:

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

Response:

OK

Parameters

- **<scan_type>**:
 - 0: passive scan
 - 1: active scan
- **<own_addr_type>**:
 - 0: Public Address
 - 1: Random Address
 - 2: RPA Public Address
 - 3: RPA Random Address
- **<filter_policy>**:
 - 0: BLE_SCAN_FILTER_ALLOW_ALL
 - 1: BLE_SCAN_FILTER_ALLOW_ONLY_WLST
 - 2: BLE_SCAN_FILTER_ALLOW_UND_RPA_DIR
 - 3: BLE_SCAN_FILTER_ALLOW_WLIST_PRA_DIR
- **<scan_interval>**: scan interval. It should be more than or equal to the value of <scan_window>. The range of this parameter is [0x0004,0x4000]. The scan interval equals this parameter multiplied by 0.625 ms, so the range for the actual scan interval is [2.5,10240] ms.
- **<scan_window>**: scan window. It should be less than or equal to the value of <scan_interval>. The range of this parameter is [0x0004,0x4000]. The scan window equals this parameter multiplied by 0.625 ms, so the range for the actual scan window is [2.5,10240] ms.

Example

```
AT+BLEINIT=1 // Role: client
AT+BLES SCANPARAM=0,0,0,100,50
```

3.4.6 AT+BLES SCAN: Enable Bluetooth LE Scanning**Set Command****Function:**

Enable or disable scanning.

Command:

```
AT+BLES SCAN=<enable>[,<duration>][,<filter_type>,<filter_param>]
```

Response:

```
+BLES SCAN:<addr>,<rssi>,<adv_data>,<scan_rsp_data>,<addr_type>
OK
+BLES CANDOONE
```

Parameters

- **<enable>**:
 - 1: enable continuous scanning.
 - 0: disable continuous scanning.
- **[<duration>]**: optional parameter. Unit: second.

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

Notes

- The response OK does not necessarily come before the response `+BLES SCAN:<addr>,<rssi>,<adv_data>,<scan_rsp_data>,<addr_type>`. It may be output before `+BLES SCAN:<addr>,<rssi>,<adv_data>,<scan_rsp_data>,<addr_type>` or after it.
- To obtain the scan response data, use the `AT+BLES SCANPARAM` command to set the scan type to active scan (`AT+BLES SCANPARAM=1, 0, 0, 100, 50`), and the peer device needs to set the `scan_rsp_data`.
- If the filter type is MAC, the default filter is supported, such as `AT+BLES SCAN=1, 3, 1, "24:0A:C4"`.
- If the filter type is NAME, the default filter is supported, such as `AT+BLES SCAN=1, 3, 2, "ESP"`.
- UUID filtering supports 16-bit, 32-bit, and 128-bit UUIDs. The UUID filter parameter must be specified as a hexadecimal string. For example, to set the UUID filter parameter to "0xFF01", use the command: `AT+BLES SCAN=1, 3, 3, "FF01"`.
- The value of the RSSI filter parameter is an integer. For example, if you want to set the RSSI filter parameter to -50, the command should be `AT+BLES SCAN=1, 3, 4, -50`. Only devices with RSSI greater than or equal to -50 dBm will be returned.
- To enable the duplicate check, use the `AT+BLES SCAN` command to set the duplicate check parameter to 1 (If this parameter is not set, the duplicate-checking function is disabled.). The duplicate check function needs to be used together with the previous filter parameters.

Example

```
AT+BLEINIT=1      // Role: client
AT+BLES SCAN=1    // start scanning
AT+BLES SCAN=0    // stop scanning
AT+BLES SCAN=1,3,1,"24:0A:C4:96:E6:88" // start scanning, filter type is MAC address
AT+BLES SCAN=1,3,2,"ESP-AT" // start scanning, filter type is device name
AT+BLES SCAN=1,3,3,"FF01" // start scanning, filter type is UUID
AT+BLES SCAN=1,3,4,-50 // start scanning, filter type is RSSI
AT+BLES SCAN=1,3,1,"24:0A:C4:96:E6:88",1 // start scanning, filter type is MAC_
↪address, perform duplicate check
```

3.4.7 AT+BLES SCANRSPDATA: Set Bluetooth LE Scan Response

Set Command

Function:

Set scan response.

Command:

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

Response:

```
OK
```

Parameter

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

Example

```
AT+BLEINIT=2    // Role: server
AT+BLEADVPARAM=32,32,0,0,7,0,,    // set advertising parameters
AT+BLESCANRSPDATA="1122334455"    // set scan response data
```

3.4.8 AT+BLEADVPARAM: Query or Set Parameters of Bluetooth LE Advertising

Query Command**Function:**

Query the parameters of advertising.

Command:

```
AT+BLEADVPARAM?
```

Response:

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

Set Command**Function:**

Set the parameters of advertising.

Command:

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

Response:

```
OK
```

Parameters

- **<adv_int_min>**: minimum advertising interval. The range of this parameter is [0x0020,0x4000]. The actual advertising interval equals this parameter multiplied by 0.625 ms, so the range for the actual minimum interval is [20, 10240] ms. It should be less than or equal to the value of <adv_int_max>.
- **<adv_int_max>**: maximum advertising interval. The range of this parameter is [0x0020,0x4000]. The actual advertising interval equals this parameter multiplied by 0.625 ms, so the range for the actual maximum interval is [20, 10240] ms. It should be more than or equal to the value of <adv_int_min>.
- **<adv_type>**:
 - 0: ADV_TYPE_IND
 - 1: ADV_TYPE_DIRECT_IND_HIGH
 - 2: ADV_TYPE_SCAN_IND
 - 3: ADV_TYPE_NONCONN_IND
 - 4: ADV_TYPE_DIRECT_IND_LOW
- **<own_addr_type>**: own Bluetooth LE address type.
 - 0: BLE_ADDR_TYPE_PUBLIC
 - 1: BLE_ADDR_TYPE_RANDOM
- **<channel_map>**: channel of advertising.
 - 1: ADV_CHNL_37
 - 2: ADV_CHNL_38
 - 4: ADV_CHNL_39
 - 7: ADV_CHNL_ALL
- **[<adv_filter_policy>]**: filter policy of advertising.
 - 0: ADV_FILTER_ALLOW_SCAN_ANY_CON_ANY
 - 1: ADV_FILTER_ALLOW_SCAN_WLST_CON_ANY
 - 2: ADV_FILTER_ALLOW_SCAN_ANY_CON_WLST
 - 3: ADV_FILTER_ALLOW_SCAN_WLST_CON_WLST
- **[<peer_addr_type>]**: remote Bluetooth LE address type.
 - 0: PUBLIC
 - 1: RANDOM
- **[<" peer_addr" >]**: remote Bluetooth LE address.

Note

- If peer_addr is not set, the query result will be all zero.

Example

```
AT+BLEINIT=2 // Role: server
AT+BLEADDR=1, "c2:34:45:78:66:89"
AT+BLEADVPARAM=50,50,0,1,4,0,1, "12:34:45:78:66:88"
// At this time, the MAC of the ESP device scanned by the BLE client is
↪ "c2:34:45:78:66:89".
```

3.4.9 AT+BLEADVDATA: Set Bluetooth LE Advertising Data

Set Command

Function:

Set advertising data.

Command:

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

Response:

```
OK
```

Parameter

- **<adv_data>**: advertising data in HEX string. For example, to set the advertising data to “0x11 0x22 0x33 0x44 0x55”, the command should be `AT+BLEADVDATA="1122334455"`. The maximum length is 31 bytes.

Note

- If advertising data is preset by command `AT+BLEADVDATAEX=<dev_name>,<uuid>,<manufacturer_data>,<include_power>`, it will be overwritten by this command.
- If you run this command to modify the device name, it is recommended to also execute the `AT+BLENAME` command to set the same device name afterwards.
- If the advertising data exceeds the maximum limit of this command, use command `AT+BLESCANRSPDATA`.

Example

```
AT+BLEINIT=2    // Role: server
AT+BLEADVPARAM=32,32,0,0,7,0,, // set advertising parameters
AT+BLEADVDATA="1122334455"    // set advertising data
```

3.4.10 AT+BLEADVDATAEX: Automatically Set Bluetooth LE Advertising Data

Query Command**Function:**

Query the parameters of advertising data.

Command:

```
AT+BLEADVDATAEX?
```

Response:

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

Set Command**Function:**

Set the advertising data and start advertising.

Command:

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

Response:

OK

Parameters

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

Note

- If advertising data is preset by command `AT+BLEADVDATA=<adv_data>`, it will be overwritten by this command.
- This command automatically changes the adv type previously set using `AT+BLEADVPARAM` to 0.
- The AT module automatically assembles packets in sequence based on the parameters provided by the user. However, the total data length is limited to 31 bytes. Any data exceeding this limit will be truncated.

Example

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

3.4.11 AT+BLEADVSTART: Start Bluetooth LE Advertising

Execute Command

Function:

Start advertising.

Command:

AT+BLEADVSTART

Response:

OK

Notes

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

- After the start of Bluetooth LE advertisement, if no connection is established, the advertisement will keep going on; if a connection is established, the advertisement will automatically end.

Example

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

3.4.12 AT+BLEADVSTOP: Stop Bluetooth LE Advertising

Execute Command**Function:**

Stop advertising.

Command:

```
AT+BLEADVSTOP
```

Response:

```
OK
```

Note

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

Example

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

3.4.13 AT+BLECONN: Establish Bluetooth LE Connection

Query Command**Function:**

Query the Bluetooth LE connection.

Command:

```
AT+BLECONN?
```

Response:

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

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

Set Command

Function:

Establish the Bluetooth LE connection.

Command:

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

Response:

If the connection is established successfully, it will prompt:

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

If the connection fails, it will prompt:

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

If the connection fails due to parameters error or other reasons, it will prompt:

```
ERROR
```

Parameters

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

Notes

- It is recommended to scan devices by running [AT+BLESCAN](#) before initiating a new connection to ensure that the target device is in the broadcast state.
- The maximum timeout for connection is 30 seconds.
- If the Bluetooth LE server is initialized and the connection is established successfully, you can use this command to discover the services in the peer device (GATTC). The following GATTC commands can also be used:
 - [AT+BLEGATTCPRIMSRV](#)
 - [AT+BLEGATTCINCLSRV](#)
 - [AT+BLEGATTCCHAR](#)
 - [AT+BLEGATTCRD](#)
 - [AT+BLEGATTCWR](#)
 - [AT+BLEGATTSIND](#)
- If the [AT+BLECONN?](#) is executed when the Bluetooth LE is not initialized ([AT+BLEINIT=0](#)), the system will not output `+BLECONN:<conn_index>,<remote_address>`.
- Supports address resolution.

Example

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

3.4.14 AT+BLECONNPARAM: Query/Update Parameters of Bluetooth LE Connection

Query Command

Function:

Query the parameters of Bluetooth LE connection.

Command:

```
AT+BLECONNPARAM?
```

Response:

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

Set Command

Function:

Update the parameters of Bluetooth LE connection.

Command:

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

Response:

```
OK
```

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

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

Parameters

- **<conn_index>**: index of Bluetooth LE connection. Range: [0,2].
- **<min_interval>**: minimum connecting interval. It should be less than or equal to the value of **<max_interval>**. The range of this parameter is [0x0006,0x0C80]. The actual connecting interval equals this parameter multiplied by 1.25 ms, so the range for the actual minimum interval is [7.5,4000] ms.
- **<max_interval>**: maximum connecting interval. It should be more than or equal to the value of **<min_interval>**. The range of this parameter is [0x0006,0x0C80]. The actual connecting interval equals this parameter multiplied by 1.25 ms, so the range for the actual maximum interval is [7.5,4000] ms.
- **<cur_interval>**: current connecting interval.
- **<latency>**: latency. Range: [0x0000,0x01F3].
- **<timeout>**: timeout. The range of this parameter is [0x000A,0x0C80]. The actual timeout equals this parameter multiplied by 10 ms, so the range for the actual timeout is [100,32000] ms.

Note

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

Example

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

3.4.15 AT+BLEDISCONN: End Bluetooth LE Connection**Execute Command****Function:**

End the Bluetooth LE connection.

Command:

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

Response:

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

Parameters

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

Example

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

3.4.16 AT+BLEDATALEN: Set Bluetooth LE Data Packet Length**Set Command****Function:**

Set the length of Bluetooth LE data packet.

Command:

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

Response:

```
OK
```

Parameters

- **<conn_index>**: index of Bluetooth LE connection. Range: [0,2].
- **<pkt_data_len>**: data packet's length. Range: [0x001B,0x00FB].

Note

- The Bluetooth LE connection has to be established first.

Example

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

3.4.17 AT+BLECFGMTU: Set Bluetooth LE MTU Length

Query Command**Function:**

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

Command:

```
AT+BLECFGMTU?
```

Response:

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

Set Command**Function:**

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

Command:

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

Response:

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

Parameters

- **<conn_index>**: index of Bluetooth LE connection. Range: [0,2].
- **<mtu_size>**: MTU length. Unit: byte. Range: [23,517].

Notes

- The Bluetooth LE connection must be established first. If this command is not executed, MTU negotiation will not be initiated automatically after the connection is established, and the default length is 23 bytes.
- Both client and server can use this command to initiate MTU negotiation. For servers, execute [AT+BLECONN](#) first with the same **<conn_index>** to establish a reverse connection to the client, then execute this command to update the MTU.

- You can specify the desired MTU value in the command. The MTU length needs to be negotiated between both parties. The OK response only indicates that a negotiation request has been initiated. The actual MTU may be less than the set value. It is recommended to query the actual negotiated result using [AT+BLECFGMTU?](#).

Example

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

3.4.18 AT+BLEGATTSSRVCRE: GATTS Creates Services

Execute Command

Function:

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

Command:

```
AT+BLEGATTSSRVCRE
```

Response:

```
OK
```

Notes

- If you are using an ESP32 as a Bluetooth LE server, the `mfg_nvs.bin` with GATTS configuration should be downloaded into flash in order to provide services.
- This command should be called immediately to create services, right after the Bluetooth LE server is initialized; If a Bluetooth LE connection is established first, the service creation will fail.
- If the Bluetooth LE client is initialized, you can use this command to create local services. Some GATTS commands can also be used, such as those to start and stop services, set attribute values, and send notifications/indications. See the list below for the specific commands.
 - [AT+BLEGATTSSRVCRE](#) (It is recommended to execute this command before the connection is established)
 - [AT+BLEGATTSSRVSTART](#) (It is recommended to execute this command before the connection is established)
 - [AT+BLEGATTSSRV](#)
 - [AT+BLEGATTSSCHAR](#)
 - [AT+BLEGATTSENTFY](#)
 - [AT+BLEGATTSSIND](#)
 - [AT+BLEGATTSSSETATTR](#)

Example

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

3.4.19 AT+BLEGATTSSRVSTART: GATTS Starts Services

Execute Command

Function:

GATTS starts all services.

Command:

```
AT+BLEGATTSSRVSTART
```

Set Command

Function:

GATTS starts a specific service.

Command:

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

Response:

```
OK
```

Parameter

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

Example

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

3.4.20 AT+BLEGATTSSRVSTOP: GATTS Stops Services

Execute Command

Function:

GATTS stops all services.

Command:

```
AT+BLEGATTSSRVSTOP
```

Set Command

Function:

GATTS stops a specific service.

Command:

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

Response:

OK

Parameter

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

Example

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

3.4.21 AT+BLEGATTSSRV: GATTS Discovers Services

Query Command

Function:

GATTS discovers services.

Command:

AT+BLEGATTSSRV?

Response:

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

Parameters

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

Example

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

3.4.22 AT+BLEGATTCHAR: GATTS Discovers Characteristics

Query Command

Function:

GATTS discovers characteristics.

Command:

```
AT+BLEGATTCHAR?
```

Response:

The response for a characteristic:

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

The response for a descriptor:

```
+BLEGATTCHAR: "desc", <srv_index>, <char_index>, <desc_index>, <desc_uuid>
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+BLEGATTCHAR?
```

3.4.23 AT+BLEGATTSENTFY: Notify a Client of the Value of a Characteristic Value from the Server

Set Command**Function:**

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

Command:

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

Response:

```
>
```

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

If the data transmission is successful, AT returns:

```
OK
```

Parameters

- **<conn_index>**: index of Bluetooth LE connection. Range: [0,2].
- **<srv_index>**: service's index. It can be queried with command [AT+BLEGATTSSRVCRE](#).
- **<char_index>**: characteristic's index. It can be queried with command [AT+BLEGATTSSRVSTART](#).
- **<length>**: data length. The maximum length is (*MTU* - 3).

Note

- The maximum data length per transmission is (*MTU* - 3) bytes. If you send more than this, the underlying layer will automatically split the data into packets for transmission.

Example

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

3.4.24 AT+BLEGATTSSIND: Indicate the Characteristic Value from the Server to a Client

Set Command

Function:

Indicate the characteristic value from the server to a client.

Command:

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

Response:

```
>
```

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

If the data transmission is successful, AT returns:

```
OK
```

Parameters

- **<conn_index>**: index of Bluetooth LE connection. Range: [0,2].
- **<srv_index>**: service's index. It can be queried with command [AT+BLEGATTSSRVCRE](#).
- **<char_index>**: characteristic's index; it can be fetched with command [AT+BLEGATTSSRVSTART](#).
- **<length>**: data length. The maximum length is (*MTU* - 3).

Example

```

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

```

3.4.25 AT+BLEGATTSSSETATTR: GATTS Sets Characteristics**Set Command****Function:**

GATTS sets its characteristic (descriptor).

Command:

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

Response:

```
>
```

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

If the setting is successful, AT returns:

```
OK
```

Parameters

- <srv_index>: service's index. It can be queried with command [AT+BLEGATTSSCHAR?](#).
- <char_index>: characteristic's index; it can be fetched with command [AT+BLEGATTSSCHAR?](#).
- [<desc_index>]: descriptor's index.
- If it is set, this command is used to set the value of the descriptor.
- Otherwise, this command is used to set the value of the characteristic.
- <length>: data length.

Note

- If the value of <length> is larger than the maximum length allowed, the setting will fail. The service table is defined in [gatts_data.csv](#).

Example

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

3.4.26 AT+BLEGATTCPRIMSRV: GATTC Discovers Primary Services

Query Command

Function:

Generic Attributes Client (GATTC) discovers primary services.

Command:

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

Response:

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

Parameters

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

Note

- The Bluetooth LE connection has to be established first.

Example

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

3.4.27 AT+BLEGATTCINCLSRV: GATTC Discovers Included Services

Set Command

Function:

GATTC discovers included services.

Command:


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

Response:

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

Parameters

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

Note

- The Bluetooth LE connection has to be established first.

Example

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

3.4.28 AT+BLEGATTCCHAR: GATTC Discovers Characteristics

Set Command**Function:**

GATTC discovers characteristics.

Command:

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

Response:

The response for a characteristic:

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

The response for a descriptor:

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

Parameters

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

Note

- The Bluetooth LE connection has to be established first.

Example

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

3.4.29 AT+BLEGATTCRD: GATTC Reads Characteristics

Set Command

Function:

GATTC reads a characteristic or descriptor.

Command:

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

Response:

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

Parameters

- **<conn_index>**: index of Bluetooth LE connection. Range: [0,2].
- **<srv_index>**: service' s index. It can be queried with command [AT+BLEGATTCPRIMSRV=<conn_index>](#).
- **<char_index>**: characteristic' s index; it can be fetched with command [AT+BLEGATTCCHAR=<conn_index>,<srv_index>](#).
- **[<desc_index>]**: descriptor' s index.
- If it is set, the value of the target descriptor will be read.
- if it is not set, the value of the target characteristic will be read.
- **<len>**: data length.
- **<value>**: <char_value> or <desc_value>.
- **<char_value>**: characteristic' s value. String format is read by command [AT+BLEGATTCRD=<conn_index>,<srv_index>,<char_index>](#). For example, if the response is +BLEGATTCRD:0,1,0, it means that the value length is 1, and the content is "0".

- **<desc_value>**: descriptor's value. String format is read by command `AT+BLEGATTCRD=<conn_index>,<srv_index>,<char_index>,<desc_index>`. For example, if the response is `+BLEGATTCRD:0,4,0123`, it means that the value length is 4, and the content is "0123".

Notes

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

Example

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

3.4.30 AT+BLEGATTCWR: GATTC Writes Characteristics

Set Command

Function:

GATTC writes characteristics or descriptors.

Command:

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

Response:

```
>
```

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

If the setting is successful, AT returns:

```
OK
```

Parameters

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

Notes

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

Example

```
AT+BLEINIT=1 // Role: client.
AT+BLECONN=0,"24:12:5f:9d:91:98"
AT+BLEGATTCPRIMSRV=0
AT+BLEGATTCHAR=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+BLEGATTCCR=0,3,4,,6
// After the symbol ">" shows, input 6 bytes of data, such as "123456". Then the
↳writing starts.
```

3.4.31 AT+BLESPPCFG: Query or Set Bluetooth LE SPP Parameters

Query Command

Function:

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

Command:

```
AT+BLESPPCFG?
```

Response:

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

Set Command

Function:

Set or reset the parameters of Bluetooth LE SPP.

Command:

```
AT+BLESPPCFG=<cfg_enable>[,<tx_service_index>,<tx_char_index>,<rx_service_index>,
↳<rx_char_index>][,<auto_conn>]
```

Response:

```
OK
```

Parameters

- **<cfg_enable>:**
- 0: all the SPP parameters will be reset, and the following parameters do not need input.
- 1: you should input the following parameters.

- **<tx_service_index>**: tx service's index. It can be queried with command [AT+BLEGATTCPRIMSRV=<conn_index>](#) and [AT+BLEGATTSSRV?](#).
- **<tx_char_index>**: tx characteristic's index. It can be queried with command [AT+BLEGATTCCCHAR=<conn_index>,<srv_index>](#) and [AT+BLEGATTSCCHAR?](#).
- **<rx_service_index>**: rx service's index. It can be queried with command [AT+BLEGATTCPRIMSRV=<conn_index>](#) and [AT+BLEGATTSSRV?](#).
- **<rx_char_index>**: rx characteristic's index. It can be queried with command [AT+BLEGATTCCCHAR=<conn_index>,<srv_index>](#) and [AT+BLEGATTSCCHAR?](#).
- **<auto_conn>**: Bluetooth LE SPP auto-reconnection flag. By default, automatic reconnection is enabled.
- 0: disable Bluetooth LE SPP automatic reconnection.
- 1: enable Bluetooth LE SPP automatic reconnection

Notes

- In Bluetooth LE client, the property of tx characteristic must be write with response or write without response, and the property of rx characteristic must be indicate or notify.
- In Bluetooth LE server, the property of tx characteristic must be indicate or notify, and the property of rx characteristic must be write with response or write without response.
- If the automatic reconnection function is disabled, when the connection is disconnected, a disconnection message is displayed (depending on AT+SYMSG), you need to send the connection command again; If this function is enabled, the connection will be automatically reconnected after disconnection, and the MCU side will not be aware of the disconnection. If the MAC of the peer end changes, the connection will fail.

Example

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

3.4.32 AT+BLESPP: Enter Bluetooth LE SPP Mode

Execute Command

Function:

Enter Bluetooth LE SPP mode.

Command:

```
AT+BLESPP
```

Response:

```
OK
>
```

This response indicates that AT has entered Bluetooth LE SPP mode and can send and receive data.

If the Bluetooth LE SPP status is wrong (Notifications are not enabled at the opposite end after the Bluetooth LE connection is established), the system returns:

```
ERROR
```

Notes

- During the SPP transmission, AT will not prompt any exit the Bluetooth LE SPP passthrough mode information unless Bit0 of [AT+SYSMMSG](#) is 1.
- During the SPP transmission, AT will not prompt any connection status changes unless Bit2 of [AT+SYSMMSG](#) is 1.
- When the packet which only contains +++ is received, the device returns to normal command mode. Please wait for at least one second before sending the next AT command.

Example

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

3.4.33 AT+BLESECPARAM: Query or Set Bluetooth LE Encryption Parameters

Query Command

Function:

Query the parameters of Bluetooth LE SMP.

Command:

```
AT+BLESECPARAM?
```

Response:

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

Set Command

Function:

Set the parameters of Bluetooth LE SMP.

Command:

```
AT+BLESECPARAM=<auth_req>,<iocap>,<enc_key_size>,<init_key>,<rsp_key>[,<auth_  
↪option>]
```

Response:

```
OK
```

Parameters

- **<auth_req>**: authentication request.
- 0: NO_BOND
- 1: BOND
- 4: MITM
- 8: SC_ONLY
- 9: SC_BOND
- 12: SC_MITM
- 13: SC_MITM_BOND
- **<iocap>**: input and output capability.

- 0: DisplayOnly
- 1: DisplayYesNo
- 2: KeyboardOnly
- 3: NoInputNoOutput
- 4: Keyboard display
- **<enc_key_size>**: encryption key size. Range: [7,16]. Unit: byte.
- **<init_key>**: initial key represented in bit combinations.
- **<rsp_key>**: response key represented in bit combinations.
- **<auth_option>**: authentication option of security.
- 0: Select the security level automatically.
- 1: If it cannot follow the preset security level, the connection will disconnect.

Note

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

Example

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

3.4.34 AT+BLEENC: Initiate Bluetooth LE Encryption Request

Set Command

Function:

Start a pairing request

Command:

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

Response:

```
OK
```

Parameters

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

Note

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

Example

```
AT+RESTORE
AT+BLEINIT=2
AT+BLEGATTSSRVCRE
AT+BLEGATTSSRVSTART
AT+BLEADDR?
AT+BLESECPARAM=1,0,16,3,3
AT+BLESETKEY=123456
AT+BLEADVSTART
// Use your Bluetooth LE debugging app as a client to establish a Bluetooth LE
↔connection with the ESP32
AT+BLEENC=0,3
```

3.4.35 AT+BLEENCRSP: Respond to the Pairing Request from the Peer Device**Set Command****Function:**

Respond to the pairing request from the peer device.

Command:

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

Response:

```
OK
```

Parameters

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

Note

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

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

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

Example

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


3.4.36 AT+BLEKEYREPLY: Reply the Key Value to the Peer Device

Set Command

Function:

Reply a pairing key.

Command:

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

Response:

```
OK
```

Parameters

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

Example

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

3.4.37 AT+BLECONFREPLY: Reply the Confirm Value to the Peer Device in the Legacy Connection Stage

Set Command

Function:

Reply a pairing result.

Command:

```
AT+BLECONFREPLY=<conn_index>,<confirm>
```

Response:

```
OK
```

Parameters

- **<conn_index>**: index of Bluetooth LE connection. Range: [0,2].
- **<confirm>**:
 - 0: No
 - 1: Yes

Example

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

3.4.38 AT+BLENCDEV: Query Bonded Bluetooth LE Encryption Device List

Query Command

Function:

Query bonded Bluetooth LE encryption device list.

Command:

```
AT+BLENCDEV?
```

Response:

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

Parameters

- **<enc_dev_index>**: index of the bonded devices. This parameter is not necessarily equal to the `conn_index` parameter in the Bluetooth LE connection list queried by the command [AT+BLECONN](#). Range: [0,14].
- **<mac_address>**: MAC address.

Note

- ESP-AT allows a maximum of 15 devices to be bonded. If the number of bonded devices exceeds 15, the newly bonded device information will sequentially (from 0 to 14) overwrite the previous device information according to the binding order.

Example

```
AT+BLENCDEV?
```

3.4.39 AT+BLENCCLEAR: Clear Bluetooth LE Encryption Device List

Set Command

Function:

Remove a device from the security database list with a specific index.

Command:

```
AT+BLENCCLEAR=<enc_dev_index>
```

Response:

```
OK
```

Execute Command

Function:

Remove all devices from the security database.

Command:

```
AT+BLEENCCLEAR
```

Response:

```
OK
```

Parameter

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

Example

```
AT+BLEENCCLEAR
```

3.4.40 AT+BLESETKEY: Set Bluetooth LE Static Pair Key

Query Command**Function:**

```
Query the Bluetooth LE static pair key. If it is not set, AT will return -1.
```

Command:

```
AT+BLESETKEY?
```

Response:

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

Set Command**Function:**

Set a Bluetooth LE static pair key for all Bluetooth LE connections.

Command:

```
AT+BLESETKEY=<static_key>
```

Response:

```
OK
```

Parameter

- **<static_key>**: static Bluetooth LE pair key.

Example

```
AT+BLESETKEY=123456
```

3.4.41 AT+BLEHIDINIT: Bluetooth LE HID Profile Initialization

Query Command

Function:

Query the initialization status of Bluetooth LE HID profile.

Command:

```
AT+BLEHIDINIT?
```

Response:

If Bluetooth LE HID device profile is not initialized, AT will return:

```
+BLEHIDINIT:0  
OK
```

If Bluetooth LE HID device profile is initialized, AT will return:

```
+BLEHIDINIT:1  
OK
```

Set Command

Function:

Initialize the Bluetooth LE HID profile.

Command:

```
AT+BLEHIDINIT=<init>
```

Response:

```
OK
```

Parameter

- **<init>**:
- 0: deinit Bluetooth LE HID profile
- 1: init Bluetooth LE HID profile

Note

- The Bluetooth LE HID command cannot be used at the same time with general GATT/GAP commands.
- The Bluetooth LE HID commands cannot be used at the same time with *BluFi commands*.

Example

```
AT+BLEHIDINIT=1
```

3.4.42 AT+BLEHIDKB: Send Bluetooth LE HID Keyboard Information

Set Command

Function:

Send keyboard information.

Command:

```
AT+BLEHIDKB=<Modifier_keys>,<key_1>,<key_2>,<key_3>,<key_4>,<key_5>,<key_6>
```

Response:

```
OK
```

Parameters

- **<Modifier_keys>**: modifier keys mask
- **<key_1>**: key code 1
- **<key_2>**: key code 2
- **<key_3>**: key code 3
- **<key_4>**: key code 4
- **<key_5>**: key code 5
- **<key_6>**: key code 6

Note

- For more information about key codes, please refer to the chapter Keyboard/Keypad Page of [Universal Serial Bus HID Usage Tables](#).
- To use this command to interact with iOS products, your devices need to pass [MFI](#) certification first.

Example

```
AT+BLEHIDKB=0,4,0,0,0,0,0 // input the string "a"
```

3.4.43 AT+BLEHIDMUS: Send Bluetooth LE HID Mouse Information

Set Command

Function:

Send mouse information.

Command:

```
AT+BLEHIDMUS=<buttons>,<X_displacement>,<Y_displacement>,<wheel>
```

Response:

```
OK
```

Parameters

- **<buttons>**: mouse button
- **<X_displacement>**: X displacement
- **<Y_displacement>**: Y displacement
- **<wheel>**: wheel

Note

- For more information about HID mouse, please refer to the section Generic Desktop Page, and Application Usages of [Universal Serial Bus HID Usage Tables](#).
- To use this command to interact with iOS products, your devices need to pass [MFI](#) certification first.

Example

```
AT+BLEHIDMUS=0,10,10,0
```

3.4.44 AT+BLEHIDCONSUMER: Send Bluetooth LE HID Consumer Information**Set Command****Function:**

Send consumer information.

Command:

```
AT+BLEHIDCONSUMER=<consumer_usage_id>
```

Response:

```
OK
```

Parameter

- **<consumer_usage_id>**: consumer ID, such as power, reset, help, volume and so on. See chapter Consumer Page (0x0C) of [HID Usage Tables for Universal Serial Bus \(USB\)](#) for more information.

Note

- To use this command to interact with iOS products, your devices need to pass [MFI](#) certification first.

Example

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

3.4.45 AT+BLUFI: Start or Stop BluFi

Query Command

Function:

Query the status of BluFi.

Command:

```
AT+BLUFI?
```

Response:

If BluFi is not started, it will return:

```
+BLUFI:0
```

```
OK
```

If BluFi is started, it will return:

```
+BLUFI:1
```

```
OK
```

Set Command

Function:

Start or stop BluFi.

Command:

```
AT+BLUFI=<option>[,<auth floor>]
```

Response:

```
OK
```

Parameter

- **<option>:**
 - 0: stop BluFi
 - 1: start BluFi
- **<auth floor>:** Wi-Fi authentication mode floor. ESP-AT will not connect to the AP whose authmode is lower than this floor.
 - 0: OPEN (Default)
 - 1: WEP
 - 2: WPA_PSK
 - 3: WPA2_PSK
 - 4: WPA_WPA2_PSK
 - 5: WPA2_ENTERPRISE
 - 6: WPA3_PSK
 - 7: WPA2_WPA3_PSK
 - 8: WAPI_PSK
 - 9: OWE
 - 10: WPA3_ENT_192
 - 11: WPA3_EXT_PSK
 - 12: WPA3_EXT_PSK_MIXED_MODE

- 13: DPP
- 14: WPA3_ENTERPRISE
- 15: WPA2_WPA3_ENTERPRISE

Note

- You can only start or stop BluFi when Bluetooth LE is not initialized (*AT+BLEINIT=0*).
- During the BluFi network configuration process, when the ESP32 receives the SSID and password sent from the phone, it will output +SSID:<ssid> and +PASSWORD:<password> respectively.
- The BluFi commands cannot be used at the same time with *Bluetooth LE HID commands*.
- To achieve better performance, it is recommended to disable SoftAP by sending the *AT+CWMODE=0/1* command before using BluFi function. For more details, please refer to the [RF Coexistence](#) documentation.
- After BluFi network configuration is completed, please send the *AT+BLUFI=0* command to disable BluFi and release resources.

Example

```
AT+BLUFI=1
```

3.4.46 AT+BLUFINAME: Query or Set BluFi Device Name

Query Command

Function:

Query the BluFi name.

Command:

```
AT+BLUFINAME?
```

Response:

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

Set Command

Function:

Set the BluFi device name.

Command:

```
AT+BLUFINAME=<device_name>
```

Response:

```
OK
```

Parameter

- **<device_name>**: the name of BluFi device.

Notes

- If you need to set BluFi name, please set it before command `AT+BLUFI=1`. Otherwise, it will use the default name `BLUFI_DEVICE`.
- The maximum length of BluFi name is 26 bytes.
- The Blufi APP can be downloaded from the app store.

Example

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

3.4.47 AT+BLUFISEND: Send User-Customized Data over BluFi

Set Command

Function:

Send user-customized data from the ESP side to a phone over BluFi.

Command:

```
AT+BLUFISEND=<length>
```

Response:

```
>
```

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

If the data transmission is successful, AT returns:

```
OK
```

Parameters

- **<length>**: customized data length. Unit: byte.

Notes

- The length of customized data cannot exceed 600 bytes.
- If your ESP chip receives customized data from the phone, the data will be printed in the form of `+BLUFI-DATA:<len>,<data>`.

Example

```
AT+BLUFISEND=4  
// After the symbol ">" shows, input 4 bytes of data, such as "1234". Then the  
↵data will be transmitted automatically.
```

3.4.48 AT+BLERDRSSI: Query the current connection RSSI

Set Command

Function:

Query the current connection RSSI.

Command:

```
AT+BLERDRSSI=<conn_index>
```

Response:

```
+BLERDRSSI:<rssi>  
OK
```

Parameters

- **<conn_index>**: index of Bluetooth LE connection.

Notes

- This command is used to obtain the signal strength of the connection, so a BLE connection must be established first.

Example

```
AT+BLEINIT=1 // Role: client  
AT+BLECONN=0,"24:0a:c4:09:34:23" // establish a BLE connection  
AT+BLERDRSSI=0 // query current connection RSSI
```

3.4.49 AT+BLEWL: Set the white list

Set Command

Function:

Set the white list.

Command:

```
AT+BLEWL=<add_or_rmv>[,<addr_type>,<addr>]
```

Response:

```
OK
```

Parameters

- **<add_or_rmv>**:
 - 0: remove a single device from the whitelist, or clear the entire whitelist.
 - 1: add a device to the white list.
- **[<addr_type>]**: address type
 - 0: Public Address
 - 1: Random Address

- [**<addr>**]: device address

Notes

- When the `<add_or_rmv>` parameter is set to 0, omitting the `<addr_type>` and `<addr>` parameters will clear the entire whitelist; if these parameters are provided, the specified device will be removed from the whitelist.

Example

```
AT+BLEWL=1,0,"24:0a:c4:09:34:23" // add a device to the white list
AT+BLEWL=0 // remove all devices from the white list
```

3.5 ESP32 Classic Bluetooth® AT Commands

- *Introduction*
- *AT+BTINIT*: Classic Bluetooth initialization.
- *AT+BTNAME*: Query or set Classic Bluetooth device name.
- *AT+BTSCANMODE*: Set Classic Bluetooth scan mode.
- *AT+BTSTARTDISC*: Start Classic Bluetooth discovery.
- *AT+BTSPPINIT*: Classic Bluetooth SPP profile initialization.
- *AT+BTSPPCONN*: Query or establish SPP connection.
- *AT+BTSPDISCONN*: End SPP connection.
- *AT+BTSPSTART*: Start Classic Bluetooth SPP profile.
- *AT+BTSPSEND*: Send data to remote Classic Bluetooth SPP device.
- *AT+BTA2DPINIT*: Classic Bluetooth A2DP profile initialization.
- *AT+BTA2DPCONN*: Establish A2DP connection.
- *AT+BTA2DPDISCONN*: End A2DP connection.
- *AT+BTA2DPSRC*: Query or set the audio file URL.
- *AT+BTA2DPCTRL*: Control the audio play.
- *AT+BTSECPARAM*: Query or set the Classic Bluetooth security parameters.
- *AT+BTKEYREPLY*: Input the Simple Pair Key.
- *AT+BTPINREPLY*: Input the Legacy Pair PIN Code.
- *AT+BTSECCFM*: Reply the confirm value to the peer device in the legacy connection stage.
- *AT+BTENCDEV*: Query Classic Bluetooth encryption device list.
- *AT+BTENCCLEAR*: Clear Classic Bluetooth encryption device list.
- *AT+BTCOD*: Set class of devices.
- *AT+BTPOWER*: Query or set power of Classic Bluetooth.

3.5.1 Introduction

ESP32 AT firmware supports [Bluetooth® Core Specification Version 4.2](#).

Important: The default AT firmware does not support the AT commands listed on this page. If you need ESP32 to support Classic Bluetooth commands, you can compile the ESP-AT project by following the steps in [Compile ESP-AT Project Locally](#) documentation. In the project configuration during the fifth step, make the following selections (Each item below is independent. Choose it according to your needs):

- Enable Generic Bluetooth commands: Component config>AT>AT bt command support
- Enable SPP commands: Component config>AT>AT bt command support>AT bt spp command support
- Enable A2DP commands: Component config>AT>AT bt command support>AT bt a2dp command support

3.5.2 AT+BTINIT: Classic Bluetooth Initialization

Query Command

Function:

Query the initialization status of Classic Bluetooth.

Command:

```
AT+BTINIT?
```

Response:

If Classic Bluetooth is initialized, AT will return:

```
+BTINIT:1  
OK
```

If Classic Bluetooth is not initialized, AT will return:

```
+BTINIT:0  
OK
```

Set Command

Function:

Initialize or deinitialize Classic Bluetooth.

Command:

```
AT+BTINIT=<init>
```

Response:

```
OK
```

Parameter

- **<init>:**
 - 0: Deinitialize Classic Bluetooth.
 - 1: Initialize Classic Bluetooth.

Notes

- To achieve better performance, it is recommended to disable SoftAP by sending the [AT+CWMODE=0/1](#) command before using Classic Bluetooth function. For more details, please refer to the [RF Coexistence](#) documentation.
- If Classic Bluetooth is initialized, [AT+CIPMODE](#) cannot be set to 1.

Example

```
AT+BTINIT=1
```

3.5.3 AT+BTNAME: Query or Set Classic Bluetooth Device Name

Query Command

Function:

Query the Classic Bluetooth device name.

Command:

```
AT+BTNAME?
```

Response:

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

Set Command

Function:

Set the Classic Bluetooth device name.

Command:

```
AT+BTNAME=<device_name>
```

Response:

```
OK
```

Parameter

- **<device_name>**: the Classic Bluetooth device name. The maximum length is 32. Default: “ESP32_AT” .

Notes

- The configuration changes will be saved in the NVS partition if [AT+SYSSSTORE=1](#).
- The default Classic Bluetooth device name is “ESP32_AT” .

Example

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

3.5.4 AT+BTSCANMODE: Set Classic Bluetooth Scan Mode

Set Command

Function:

Set the scan mode of Classic Bluetooth.

Command:

```
AT+BTSCANMODE=<scan_mode>
```

Response:

OK

Parameter

- **<scan_mode>**:
 - 0: Neither discoverable nor connectable.
 - 1: Connectable but not discoverable.
 - 2: Both discoverable and connectable.
 - 3: Discoverable but not connectable.

Example

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

3.5.5 AT+BTSTARTDISC: Start Classic Bluetooth Discovery

Set Command

Function:

Start Classic Bluetooth discovery.

Command:

AT+BTSTARTDISC=<inq_mode>,<inq_len>,<inq_num_rsps>

Response:

+BTSTARTDISC:<bt_addr>,<dev_name>,<major_dev_class>,<minor_dev_class>,<major_srv_↵class>,<rssi>

OK

Parameters

- **<inq_mode>**:
 - 0: General inquiry mode.
 - 1: Limited inquiry mode.
- **<inq_len>**: inquiry duration. Range: 0x01 ~ 0x30.
- **<inq_num_rsps>**: number of inquiry responses that can be received. If you set it to 0, AT will receive an unlimited number of responses.
- **<bt_addr>**: Classic Bluetooth address.
- **<dev_name>**: device name.
- **<major_dev_class>**:
 - 0x0: miscellaneous.
 - 0x1: computer.
 - 0x2: phone (cellular, cordless, pay phone, modem).
 - 0x3: LAN, Network Access Point.
 - 0x4: audio/video (headset, speaker, stereo, video display, VCR).
 - 0x5: peripheral (mouse, joystick, keyboard).
 - 0x6: imaging (printer, scanner, camera, display).
 - 0x7: wearable.
 - 0x8: toy.
 - 0x9: health.

- 0x1F: uncategorized device.
- **<minor_dev_class>**: please refer to [Minor Device Class field](#).
- **<major_srv_class>**:
 - 0x0: an invalid value.
 - 0x1: Limited Discoverable Mode.
 - 0x8: positioning (location identification).
 - 0x10: networking, such as LAN, Ad hoc.
 - 0x20: rendering, such as printing, speakers.
 - 0x40: capturing, such as scanner, microphone.
 - 0x80: object transfer, such as v-Inbox, v-Folder.
 - 0x100: audio, such as speaker, microphone, headset service.
 - 0x200: telephony, such as cordless telephony, modem, headset service.
 - 0x400: information, such as WEB-server, WAP-server.
- **<rsssi>**: signal strength.

Example

```
AT+BTINIT=1
AT+BTSCANMODE=2
AT+BTSTARTDISC=0,10,10
```

3.5.6 AT+BTSPPINIT: Classic Bluetooth SPP Profile Initialization

Query Command

Function:

Query the initialization status of Classic Bluetooth SPP profile.

Command:

```
AT+BTSPPINIT?
```

Response:

If Classic Bluetooth SPP profile is initialized, it will return:

```
+BTSPPINIT:1
OK
```

If Classic Bluetooth SPP profile is not initialized, it will return:

```
+BTSPPINIT:0
OK
```

Set Command

Function:

Initialize or deinitialize Classic Bluetooth SPP profile.

Command:

```
AT+BTSPPINIT=<init>
```

Response:

OK

Parameter

- **<init>**:
 - 0: Deinitialize Classic Bluetooth SPP profile.
 - 1: Initialize Classic Bluetooth SPP profile, the role is master.
 - 2: Initialize Classic Bluetooth SPP profile, the role is slave.

Example

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

3.5.7 AT+BTSPPCONN: Query or establish SPP Connection

Query Command

Function:

Query Classic Bluetooth SPP connection.

Command:

AT+BTSPPCONN?

Response:

+BTSPPCONN:<conn_index>,<remote_address>
OK

If the connection has not been established, AT will return:

+BTSPPCONN:-1

Set Command

Function:

Establish the Classic Bluetooth SPP connection.

Command:

AT+BTSPPCONN=<conn_index>,<sec_mode>,<remote_address>

Response:

OK

If the connection is established successfully, AT will return:

+BTSPPCONN:<conn_index>,<remote_address>

Otherwise, AT will return:


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

Parameters

- **<conn_index>**: index of Classic Bluetooth SPP connection. Only 0 is supported for the single connection right now.
- **<sec_mode>**:
 - 0x0000: no security.
 - 0x0001: authorization required (only needed for out going connection).
 - 0x0036: encryption required.
 - 0x3000: Man-In-The-Middle protection.
 - 0x4000: Min 16 digit for pin code.
- **<remote_address>**: remote Classic Bluetooth SPP device address.

Example

```
AT+BTSPPCONN=0,0,"24:0a:c4:09:34:23"
```

3.5.8 AT+BTSPDISCONN: End SPP Connection

Execute Command

Function:

End the Classic Bluetooth SPP connection.

Command:

```
AT+BTSPDISCONN=<conn_index>
```

Response:

```
OK
```

If the command is successful, it will prompt:

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

If the command is fail, it will prompt:

```
+BTSPDISCONN:-1
```

Parameters

- **<conn_index>**: index of Classic Bluetooth SPP connection. Only 0 is supported for the single connection right now.
- **<remote_address>**: remote Classic Bluetooth A2DP device address.

Example

```
AT+BTSPDISCONN=0
```

3.5.9 AT+BTSPSEND: Send Data to Remote Classic Bluetooth SPP Device

Execute Command

Function:

Enter Classic Bluetooth SPP mode.

Command:

```
AT+BTSPSEND
```

Response:

```
>
```

Set Command

Function:

Send data to the remote Classic Bluetooth SPP device.

Command:

```
AT+BTSPSEND=<conn_index>,<data_len>
```

Response:

```
OK
```

Parameters

- **<conn_index>**: index of Classic Bluetooth SPP connection. Only 0 is supported for the single connection right now.
- **<data_len>**: the length of the data which is ready to be sent.

Notes

- The wrap return is > after this command is executed. Then, the ESP32 enters UART Bluetooth passthrough mode. When the packet which only contains +++ is received, the device returns to normal command mode. Please wait for at least one second before sending the next AT command.

Example

```
AT+BTSPSEND=0,100
AT+BTSPSEND
```

3.5.10 AT+BTSPSTART: Start Classic Bluetooth SPP Profile

Execute Command

Function:

Start Classic Bluetooth SPP profile.

Command:

```
AT+BTSPSTART
```

Response:

```
OK
```

Note

- During the SPP transmission, AT will not prompt any connection status changes unless bit 2 of [AT+SYSMMSG](#) is 1.

Example

```
AT+BTSPSTART
```

3.5.11 AT+BTA2DPINIT: Classic Bluetooth A2DP Profile Initialization

Query Command**Function:**

Query the initialization status of Classic Bluetooth A2DP profile.

Command:

```
AT+BTA2DPINIT?
```

Response:

If Classic Bluetooth A2DP profile is initialized, AT will return:

```
+BTA2DPINIT:<role>
```

```
OK
```

Otherwise, AT will return:

```
+BTA2DPINIT:0
```

```
OK
```

Set Command**Function:**

Initialize or deinitialize Classic Bluetooth A2DP profile.

Command:

```
AT+BTA2DPINIT=<role>
```

Response:

```
OK
```

Parameters

- **<role>**: role
 - 0: Deinitialize Classic Bluetooth A2DP profile.
 - 1: Source.
 - 2: Sink.

Example

```
AT+BTA2DPINIT=2
```

3.5.12 AT+BTA2DPCONN: Query or establish A2DP Connection

Query Command

Function:

Query Classic Bluetooth A2DP connection.

Command:

```
AT+BTA2DPCONN?
```

Response:

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

If the connection has not been established, AT will NOT return the parameter <conn_index> and <remote_address>.

Set Command

Function:

Establish the Classic Bluetooth A2DP connection.

Command:

```
AT+BTA2DPCONN=<conn_index>,<remote_address>
```

Response:

```
OK
```

If the connection is established successfully, it will prompt the message below:

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

Otherwise, it will return:

```
+BTA2DPCONN:<conn_index>,-1
```

Parameters

- **<conn_index>**: index of Classic Bluetooth A2DP connection. Only 0 is supported for the single connection right now.
- **<remote_address>**: remote Classic Bluetooth A2DP device address.

Example

```
AT+BTA2DPCONN=0,0,0,"24:0a:c4:09:34:23"
```

3.5.13 AT+BTA2DPDISCONN: End A2DP Connection**Execute Command****Function:**

End the Classic Bluetooth A2DP connection.

Command:

```
AT+BTA2DPDISCONN=<conn_index>
```

Response:

```
+BTA2DPDISCONN:<conn_index>,<remote_address>  
OK
```

Parameters

- **<conn_index>**: index of Classic Bluetooth A2DP connection. Only 0 is supported for the single connection right now.
- **<remote_address>**: remote Classic Bluetooth A2DP device address.

Example

```
AT+BTA2DPDISCONN=0
```

3.5.14 AT+BTA2DPSRC: Query or Set the Audio File URL**Query Command****Function:**

Query the audio file URL.

Command:

```
AT+BTA2DPSRC?
```

Response:

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

Execute Command**Function:**

Set the audio file URL.

Command:

```
AT+BTA2DPSRC=<conn_index>,<url>
```

Response:

```
OK
```

Parameters

- **<conn_index>**: index of Classic Bluetooth A2DP connection. Only 0 is supported for the single connection right now.
- **<url>**: the path of the source file. HTTP, HTTPS and FLASH are currently supported.
- **<type>**: the type of audio file, such as “mp3” .

Note

- Only mp3 format is currently supported.

Example

```
AT+BTA2DPSRC=0,"https://dl.espressif.com/dl/audio/ff-16b-2c-44100hz.mp3"  
AT+BTA2DPSRC=0,"flash://spiffs/zhifubao.mp3"
```

3.5.15 AT+BTA2DPCTRL: Control the Audio Play

Execute Command**Function:**

Control the audio play.

Command:

```
AT+BTA2DPCTRL=<conn_index>,<ctrl>
```

Response:

```
OK
```

Parameters

- **<conn_index>**: index of Classic Bluetooth A2DP connection. Only 0 is supported for the single connection right now.
- **<ctrl>**: types of control.
 - 0: A2DP Sink, stop play.
 - 1: A2DP Sink, start play.
 - 2: A2DP Sink, forward.
 - 3: A2DP Sink, backward.
 - 4: A2DP Sink, fastward start.
 - 5: A2DP Sink, fastward stop.
 - 0: A2DP Source, stop play.
 - 1: A2DP Source, start play.
 - 2: A2DP Source, suspend.

Example

```
AT+BTA2DPCTRL=0,1 // start play audio
```

3.5.16 AT+BTSECPARAM: Query or Set the Classic Bluetooth Security Parameters

Query Command**Function:**

Query Classic Bluetooth security parameters.

Command:

```
AT+BTSECPARAM?
```

Response:

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

Set Command**Function:**

Set the Classic Bluetooth security parameters.

Command:

```
AT+BTSECPARAM=<io_cap>,<pin_type>,<pin_code>
```

Response:

```
OK
```

Parameters

- **<io_cap>**: input and output capability.
 - 0: DisplayOnly.
 - 1: DisplayYesNo.
 - 2: KeyboardOnly.
 - 3: NoInputNoOutput.
- **<pin_type>**: use variable or fixed PIN.
 - 0: Variable.
 - 1: Fixed.
- **<pin_code>**: Legacy Pair PIN Code. Maximum: 16 bytes.

Note

- If you set the parameter <pin_type> to 0, <pin_code> will be ignored.

Example

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

3.5.17 AT+BTKEYREPLY: Input the Simple Pair Key

Execute Command

Function:

Input the Simple Pair Key.

Command:

```
AT+BTKEYREPLY=<conn_index>,<Key>
```

Response:

```
OK
```

Parameters

- **<conn_index>**: index of Classic Bluetooth connection. Currently, only 0 is supported for the single connection.
- **<Key>**: the Simple Pair Key.

Example

```
AT+BTKEYREPLY=0,123456
```

3.5.18 AT+BTPINREPLY: Input the Legacy Pair PIN Code

Execute Command

Function:

Input the Legacy Pair PIN Code.

Command:

```
AT+BTPINREPLY=<conn_index>,<Pin>
```

Response:

```
OK
```

Parameters

- **<conn_index>**: index of Classic Bluetooth connection. Currently, only 0 is supported for the single connection.
- **<Pin>**: the Legacy Pair PIN Code.

Example

```
AT+BTPINREPLY=0,"6688"
```


3.5.19 AT+BTSECCFM: Reply the Confirm Value to the Peer Device in the Legacy Connection Stage

Execute Command

Function:

Reply the confirm value to the peer device in the legacy connection stage.

Command:

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

Response:

```
OK
```

Parameters

- **<conn_index>**: index of Classic Bluetooth connection. Currently, only 0 is supported for the single connection.
- **<accept>**: reject or accept.
 - 0: Reject.
 - 1: Accept.

Example

```
AT+BTSECCFM=0,1
```

3.5.20 AT+BTENCDEV: Query Classic Bluetooth Encryption Device List

Query Command

Function:

Query the bound devices.

Command:

```
AT+BTENCDEV?
```

Response:

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

Parameters

- **<enc_dev_index>**: index of the bound devices.
- **<mac_address>**: MAC address.

Example

```
AT+BTENCDEV?
```

3.5.21 AT+BTENCCLEAR: Clear Classic Bluetooth Encryption Device List

Set Command

Function:

Remove a device from the security database list with a specific index.

Command:

```
AT+BTENCCLEAR=<enc_dev_index>
```

Response:

```
OK
```

Execute Command

Function:

Remove all devices from the security database.

Command:

```
AT+BTENCCLEAR
```

Response:

```
OK
```

Parameter

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

Example

```
AT+BTENCCLEAR
```

3.5.22 AT+BTCOD: Set Class of Devices

Set Command

Function:

Set the Classic Bluetooth class of devices.

Command:

```
AT+BTCOD=<major>,<minor>,<service>
```

Response:

```
OK
```

Parameters

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

Example

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

3.5.23 AT+BTPOWER: Query or Set TX power of Classic Bluetooth

Query Command

Function:

Query Classic Bluetooth tx power level.

Command:

```
AT+BTPOWER?
```

Response:

```
+BTPOWER:<min_tx_power>,<max_tx_power>  
OK
```

Set Command

Function:

Set the Classic Bluetooth tx power.

Command:

```
AT+BTPOWER=<min_tx_power>,<max_tx_power>
```

Response:

```
OK
```

Parameters

- **<min_tx_power>**: The minimum power level. Range: [0,7].
- **<max_tx_power>**: The maximum power level. Range: [0,7].

Example

```
AT+BTPOWER=5,6 // set Classic Bluetooth tx power.
```

3.6 MQTT AT Commands

- [Introduction](#)
- [AT+MQTTUSERCFG](#): Set MQTT user configuration.
- [AT+MQTTLONGCLIENTID](#): Set MQTT client ID.
- [AT+MQTTLONGUSERNAME](#): Set MQTT username.
- [AT+MQTTLONGPASSWORD](#): Set MQTT password.
- [AT+MQTTCONNCFG](#): Set configuration of MQTT connection.
- [AT+MQTTALPN](#): Set MQTT Application Layer Protocol Negotiation (ALPN).
- [AT+MQTTSDNI](#): Set MQTT Server Name Indication (SNI).
- [AT+MQTTCONN](#): Connect to MQTT Brokers.
- [AT+MQTTPUB](#): Publish MQTT Messages in string.
- [AT+MQTTPUBRAW](#): Publish long MQTT messages.
- [AT+MQTTSUB](#): Subscribe to MQTT topics.
- [AT+MQTTUNSUB](#): Unsubscribe from MQTT topics.
- [AT+MQTTCLEAN](#): Close MQTT connections.
- [MQTT AT Error Codes](#)
- [MQTT AT Notes](#)

3.6.1 Introduction

Important:

- Currently, AT firmware for ESP32 series supports [MQTT Version 3.1.1](#).
 - The default AT firmware supports all the AT commands mentioned on this page. If you don't need ESP32 to support MQTT commands, you can compile the ESP-AT project by following the steps in [Compile ESP-AT Project Locally](#) documentation. In the project configuration during the fifth step, make the following selections:
 - Disable Component config>AT>AT MQTT command support
 - For MQTT connection to local Broker examples, see [MQTT AT Examples](#).
 - For MQTT connection to AWS IoT examples, see [MQTT Cloud Connection AT Examples](#).
-

3.6.2 AT+MQTTUSERCFG: Set MQTT User Configuration

Set Command**Function:**

Set MQTT User Configuration.

Command:

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

Response:

```
OK
```

Parameters

- **<LinkID>**: currently only supports link ID 0.
- **<scheme>**:
 - 1: MQTT over TCP.

- 2: MQTT over TLS (no certificate verify).
- 3: MQTT over TLS (verify server certificate).
- 4: MQTT over TLS (provide client certificate).
- 5: MQTT over TLS (verify server certificate and provide client certificate).
- 6: MQTT over WebSocket (based on TCP).
- 7: MQTT over WebSocket Secure (based on TLS, no certificate verify).
- 8: MQTT over WebSocket Secure (based on TLS, verify server certificate).
- 9: MQTT over WebSocket Secure (based on TLS, provide client certificate).
- 10: MQTT over WebSocket Secure (based on TLS, verify server certificate and provide client certificate).
- **<" client_id" >**: MQTT client ID. Maximum length: 256 bytes.
- **<" username" >**: the username to login to the MQTT broker. Maximum length: 64 bytes.
- **<" password" >**: the password to login to the MQTT broker. Maximum length: 64 bytes.
- **<cert_key_ID>**: certificate ID. Currently, ESP-AT only supports one certificate for ID 0.
- **<CA_ID>**: CA ID. Currently, ESP-AT only supports one CA for ID 0.
- **<" path" >**: the path of the resource. Maximum length: 32 bytes.

Note

- The length of the entire AT command should be less than 256 bytes.
- If you want to use your own certificate at runtime, use the [AT+SYSMFG](#) command to update the MQTT certificate (for detailed steps, please refer to [AT+SYSMFG command examples](#), the certificate configuration method is the same as SSL certificates). If you want to pre-burn your own certificate, please refer to [How to Update PKI Configuration](#).
- If **<scheme>** is configured to 3, 5, 8, or 10, in order to check the server certificate validity period, please make sure ESP32 has obtained the current time before sending the [AT+MQTTCONN](#) command. (You can send [AT+CIPSNTPCFG](#) command to configure SNTP and obtain the current time, and send [AT+CIPSNPTIME?](#) command to query the current time.)
- For TLS protocol version configuration, please refer to the [modify tls protocol version note](#).

3.6.3 AT+MQTTLONGCLIENTID: Set MQTT Client ID

Set Command

Function:

Set MQTT Client ID.

Command:

```
AT+MQTTLONGCLIENTID=<LinkID>,<length>
```

Response:

```
OK
```

```
>
```

This response indicates that AT is ready for receiving MQTT Client ID. You should enter the Client ID, and when the Client ID length reaches the **<length>** value, the system returns:

```
OK
```

Parameters

- **<LinkID>**: currently only supports link ID 0.
- **<length>**: MQTT client ID length. Range: [1,1024].

Notes

- The command *AT+MQTTUSERCFG* can also set MQTT client ID. The differences between the two commands include:
 - You can use *AT+MQTTLONGCLIENTID* to set a relatively long client ID since there is a limitation on the length of the *AT+MQTTUSERCFG* command.
 - You should set *AT+MQTTLONGCLIENTID* after setting the *AT+MQTTUSERCFG* command.

3.6.4 AT+MQTTLONGUSERNAME: Set MQTT Username

Set Command

Function:

Set MQTT username.

Command:

```
AT+MQTTLONGUSERNAME=<LinkID>,<length>
```

Response:

```
OK
```

```
>
```

This response indicates that AT is ready for receiving MQTT username. You should enter the MQTT username, and when the MQTT username length reaches the `<length>` value, the system returns:

```
OK
```

Parameters

- **<LinkID>**: only supports link ID 0 currently.
- **<length>**: MQTT username length. Range: [1,1024].

Notes

- The command *AT+MQTTUSERCFG* can also set MQTT username. The differences between the two commands include:
 - You can use *AT+MQTTLONGUSERNAME* to set a relatively long username since there is a limitation on the length of the *AT+MQTTUSERCFG* command.
 - You should set *AT+MQTTLONGUSERNAME* after setting the command *AT+MQTTUSERCFG*.

3.6.5 AT+MQTTLONGPASSWORD: Set MQTT Password

Set Command

Function:

Set MQTT password.

Command:

```
AT+MQTTLONGPASSWORD=<LinkID>,<length>
```

Response:

```
OK
```

```
>
```

This response indicates that AT is ready for receiving MQTT password. You should enter the MQTT password, and when the MQTT password length reaches the `<length>` value, the system returns:

```
OK
```

Parameters

- **<LinkID>**: only supports link ID 0 currently.
- **<length>**: MQTT password length. Range: [1,1024].

Notes

- The command [AT+MQTTUSERCFG](#) can also set MQTT password. The differences between the two commands include:
 - You can use [AT+MQTTLONGPASSWORD](#) to set a relatively long password since there is a limitation on the length of the [AT+MQTTUSERCFG](#) command.
 - You should set [AT+MQTTLONGPASSWORD](#) after setting the command [AT+MQTTUSERCFG](#).

3.6.6 AT+MQTTCONNCFG: Set Configuration of MQTT Connection

Set Command

Function:

Set configuration of MQTT Connection.

Command:

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

Response:

```
OK
```

Parameters

- **<LinkID>**: only supports link ID 0 currently.
- **<keepalive>**: timeout of MQTT ping. Unit: second. Range [0,7200]. The default value is 0, which will be force-changed to 120 s.
- **<disable_clean_session>**: set MQTT clean session. For more details about this parameter, please refer to the section [Clean Session](#) in *MQTT Version 3.1.1*.
 - 0: Enable clean session.
 - 1: Disable clean session.
- **<" lwt_topic" >**: LWT (Last Will and Testament) message topic. Maximum length: 128 bytes.
- **<" lwt_msg" >**: LWT message. Maximum length: 128 bytes.
- **<lwt_qos>**: LWT QoS, which can be set to 0, 1, or 2. Default: 0.
- **<lwt_retain>**: LWT retain, which can be set to 0 or 1. Default: 0.

Note

- Before setting this command, you should set [AT+MQTTUSERCFG](#).

3.6.7 AT+MQTTALPN: Set MQTT Application Layer Protocol Negotiation (ALPN)

Set Command**Function:**

Set MQTT Application Layer Protocol Negotiation (ALPN).

Command:

```
AT+MQTTALPN=<LinkID>,<alpn_counts>[,<"alpn">][,<"alpn">][,<"alpn">]
```

Response:

```
OK
```

Parameters

- **<LinkID>**: only supports link ID 0 currently.
- **<alpn_counts>**: the number of **<" alpn" >** parameters. Range: [0,5].
 - 0: Clean the MQTT ALPN configuration.
 - [1,5]: Set the MQTT ALPN configuration.
- **<" alpn" >**: you can send more than one ALPN in ClientHello to the server.

Notes

- The length of the entire AT command should be less than 256 bytes.
- MQTT ALPN will be effective only if the MQTT connection is based on TLS or WSS.
- You should set [AT+MQTTALPN](#) after setting the command [AT+MQTTUSERCFG](#).

Example

```
AT+CWMODE=1
AT+CWJAP="ssid","password"
AT+CIPSENTPCFG=1,8,"ntp1.aliyun.com","ntp2.aliyun.com"
AT+MQTTUSERCFG=0,5,"ESP32","espressif","1234567890",0,0,""
AT+MQTTALPN=0,2,"mqtt-ca.cn","mqtt-ca.us"
AT+MQTTCONN=0,"192.168.200.2",8883,1
```

3.6.8 AT+MQTTSDNI: Set MQTT Server Name Indication (SNI)

Set Command**Function:**

Set MQTT Server Name Indication (SNI).

Command:

```
AT+MQTTSDNI=<LinkID>,<"sni">
```

Response:

OK

Parameters

- **<LinkID>**: only supports link ID 0 currently.
- **<" sni" >**: MQTT Server Name Indication. You can send it in ClientHello to the server.

Notes

- The length of the entire AT command should be less than 256 bytes.
- MQTT SNI will be effective only if the MQTT connection is based on TLS or WSS.
- You should set *AT+MQTTSNI* after setting the command *AT+MQTTUSERCFG*.

Example

```
AT+CWMODE=1
AT+CWLJAP="ssid", "password"
AT+CIPSNTPCFG=1, 8, "ntp1.aliyun.com", "ntp2.aliyun.com"
AT+MQTTUSERCFG=0, 5, "ESP32", "espressif", "1234567890", 0, 0, ""
AT+MQTTSNI=0, "my_specific_prefix.iot.my_aws_region.amazonaws.com"
AT+MQTTCONN=0, "my_specific_prefix.iot.my_aws_region.amazonaws.com", 8883, 1
```

3.6.9 AT+MQTTCONN: Connect to MQTT Brokers

Query Command

Function:

Query the MQTT broker that ESP32 are connected to.

Command:

AT+MQTTCONN?

Response:

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

Set Command

Function:

Connect to an MQTT broker.

Command:

AT+MQTTCONN=<LinkID>,<"host">,<port>,<reconnect>[,<timeout_ms>]

Response:

OK

Parameters

- **<LinkID>**: only supports link ID 0 currently.
- **<" host" >**: MQTT broker domain. Maximum length: 128 bytes.
- **<port>**: MQTT broker port. Maximum: port 65535.
- **<" path" >**: path. Maximum length: 32 bytes.
- **<reconnect>**:
 - 0: MQTT will not reconnect automatically. If MQTT connection established and then disconnected, you cannot use this command to reestablish MQTT connection. Please send [AT+MQTTCLEAN=0](#) command to clean MQTT connection first, reconfigure the connection parameters, and then establish a new MQTT connection.
 - 1: MQTT will reconnect automatically. It takes more resources.
- **<timeout_ms>**: timeout for the command. Unit: milliseconds. Range: [3000,60000]. Default: 15000 milliseconds.
- **<state>**: MQTT state.
 - 0: MQTT uninitialized.
 - 1: Already set [AT+MQTTUSERCFG](#).
 - 2: Already set [AT+MQTTCONNCFG](#).
 - 3: Connection disconnected.
 - 4: Connection established.
 - 5: Connected, but did not subscribe to any topic.
 - 6: Connected, and subscribed to MQTT topics.
- **<scheme>**:
 - 1: MQTT over TCP.
 - 2: MQTT over TLS (no certificate verify).
 - 3: MQTT over TLS (verify server certificate).
 - 4: MQTT over TLS (provide client certificate).
 - 5: MQTT over TLS (verify server certificate and provide client certificate).
 - 6: MQTT over WebSocket (based on TCP).
 - 7: MQTT over WebSocket Secure (based on TLS, verify no certificate).
 - 8: MQTT over WebSocket Secure (based on TLS, verify server certificate).
 - 9: MQTT over WebSocket Secure (based on TLS, provide client certificate).
 - 10: MQTT over WebSocket Secure (based on TLS, verify server certificate and provide client certificate).

3.6.10 AT+MQTTPUB: Publish MQTT Messages in String

Set Command

Function:

Publish MQTT messages in string to a defined topic. If the amount of data you publish is relatively large, and the length of a single AT command has exceeded the threshold of 256, please use the [AT+MQTTPUBRAW](#) command.

Command:

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

Response:

```
OK
```

Parameters

- **<LinkID>**: only supports link ID 0 currently.
- **<" topic" >**: MQTT topic. Maximum length: 128 bytes.
- **<data>**: MQTT message in string.
- **<qos>**: QoS of message, which can be set to 0, 1, or 2. Default: 0.

- **<retain>**: retain flag.
- **<timeout_ms>**: timeout for the command. Unit: milliseconds. Range: [3000,60000]. Default: 15000 milliseconds.

Notes

- The length of the entire AT command should be less than 256 bytes.
- This command cannot send data \0. If you need to send \0, please use the command [AT+MQTTPUBRAW](#) instead.

Example

```
AT+CWMODE=1
AT+CWJAP="ssid", "password"
AT+MQTTUSERCFG=0,1, "ESP32", "espressif", "1234567890", 0, 0, ""
AT+MQTTCONN=0, "192.168.10.234", 1883, 0
AT+MQTTPUB=0, "topic", "\"{"timestamp\\":\\"20201121085253\\"}\\\"", 0, 0 // When sending
↪this command, please pay attention to whether special characters need to be
↪escaped.
```

3.6.11 AT+MQTTPUBRAW: Publish Long MQTT Messages

Set Command

Function:

Publish long MQTT messages to a defined topic. If the amount of data you publish is relatively small, and the length of a single AT command is not greater than the threshold of 256, you also can use the [AT+MQTTPUB](#) command.

Command:

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

Response:

```
OK
>
```

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

If the transmission is successful, AT returns:

```
+MQTTPUB:OK
```

Otherwise, it returns:

```
+MQTTPUB:FAIL
```

Parameters

- **<LinkID>**: only supports link ID 0 currently.
- **<"topic">**: MQTT topic. Maximum length: 128 bytes.
- **<length>**: length of MQTT message. The maximum length is limited by available memory.
- **<qos>**: QoS of the published message, which can be set to 0, 1, or 2. Default is 0.
- **<retain>**: retain flag.

- **<timeout_ms>**: timeout for the command. Unit: milliseconds. Range: [3000,60000]. Default: 15000 milliseconds.

3.6.12 AT+MQTTSUB: Subscribe to MQTT Topics

Query Command

Function:

List all MQTT topics that have been already subscribed.

Command:

```
AT+MQTTSUB?
```

Response:

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

Set Command

Function:

Subscribe to defined MQTT topics with defined QoS. Multiple topics are available for subscription (up to 10 topics can be subscribed).

Command:

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

Response:

```
OK
```

When AT receives MQTT messages of the subscribed topic, it will prompt:

```
+MQTTSUBRCV:<LinkID>,<"topic">,<data_length>,<data>
```

If the topic has been subscribed before, it will prompt:

```
ALREADY SUBSCRIBE
```

Parameters

- **<LinkID>**: only supports link ID 0 currently.
- **<state>**: MQTT state.
 - 0: MQTT uninitialized.
 - 1: Already set *AT+MQTTUSERCFG*.
 - 2: Already set *AT+MQTTCONNCFG*.
 - 3: Connection disconnected.
 - 4: Connection established.
 - 5: Connected, but subscribe to no topic.
 - 6: Connected, and subscribed to MQTT topics.
- **<"topic">**: the topic that is subscribed to.
- **<qos>**: the QoS that is subscribed to.

- **<timeout_ms>**: timeout for the command. Unit: milliseconds. Range: [3000,60000]. Default: 15000 milliseconds.

Note

- Due to MTU limitations or traffic policy management on each WAN routing node, a single message sent by the MQTT broker may eventually be split into multiple messages received by the ESP32 device. Therefore, ESP32 may receive multiple +MQTTSUBRECV messages.
- If the ESP32 device receives an MQTT message whose length exceeds 1024 bytes (including the MQTT header and MQTT payload), it will also be split into multiple +MQTTSUBRECV messages. In this case, you can increase the MQTT buffer size as follows to avoid this situation.
 - ./build.py menuconfig>Component config>ESP-MQTT Configurations>MQTT Using custom configurations
 - ./build.py menuconfig>Component config>ESP-MQTT Configurations>MQTT Using custom configurations>Default MQTT Buffer Size>1460

3.6.13 AT+MQTTUNSUB: Unsubscribe from MQTT Topics

Set Command**Function:**

Unsubscribe the client from defined topics. This command can be called multiple times to unsubscribe from different topics.

Command:

```
AT+MQTTUNSUB=<LinkID>,<"topic">[,<timeout_ms>]
```

Response:

```
OK
```

If the topic has not been subscribed, AT will prompt:

```
NO UNSUBSCRIBE
```

```
OK
```

Parameters

- **<LinkID>**: only supports link ID 0 currently.
- **<"topic">**: MQTT topic. Maximum length: 128 bytes.
- **<timeout_ms>**: timeout for the command. Unit: milliseconds. Range: [3000,60000]. Default: 15000 milliseconds.

3.6.14 AT+MQTTCLEAN: Close MQTT Connections

Set Command**Function:**

Close the MQTT connection and release the resource.

Command:

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

Response:

```
OK
```

Parameter

- **<LinkID>**: only supports link ID 0 currently.

3.6.15 MQTT AT Error Codes

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

Error Type	Error Code
AT_MQTT_NO_CONFIGURED	0x6001
AT_MQTT_NOT_IN_CONFIGURED_STATE	0x6002
AT_MQTT_UNINITIATED_OR_ALREADY_CLEAN	0x6003
AT_MQTT_ALREADY_CONNECTED	0x6004
AT_MQTT_MALLOC_FAILED	0x6005
AT_MQTT_NULL_LINK	0x6006
AT_MQTT_NULL_PARAMETER	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

continues on next page

Table 3 – continued from previous page

Error Type	Error Code
AT_MQTT_PATH_IS_NULL	0x6028
AT_MQTT_PATH_IS_OVERLENGTH	0x6029
AT_MQTT_VERSION_READ_FAILED	0x602A
AT_MQTT_KEEPAKIVE_READ_FAILED	0x602B
AT_MQTT_KEEPAKIVE_IS_NULL	0x602C
AT_MQTT_KEEPAKIVE_VALUE_IS_WRONG	0x602D
AT_MQTT_DISABLE_CLEAN_SESSION_READ_FAILED	0x602E
AT_MQTT_DISABLE_CLEAN_SESSION_VALUE_IS_WRONG	0x602F
AT_MQTT_LWT_TOPIC_READ_FAILED	0x6030
AT_MQTT_LWT_TOPIC_IS_NULL	0x6031
AT_MQTT_LWT_TOPIC_IS_OVERLENGTH	0x6032
AT_MQTT_LWT_MSG_READ_FAILED	0x6033
AT_MQTT_LWT_MSG_IS_NULL	0x6034
AT_MQTT_LWT_MSG_IS_OVERLENGTH	0x6035
AT_MQTT_LWT_QOS_READ_FAILED	0x6036
AT_MQTT_LWT_QOS_VALUE_IS_WRONG	0x6037
AT_MQTT_LWT_RETAIN_READ_FAILED	0x6038
AT_MQTT_LWT_RETAIN_VALUE_IS_WRONG	0x6039
AT_MQTT_HOST_READ_FAILED	0x603A
AT_MQTT_HOST_IS_NULL	0x603B
AT_MQTT_HOST_IS_OVERLENGTH	0x603C
AT_MQTT_PORT_READ_FAILED	0x603D
AT_MQTT_PORT_VALUE_IS_WRONG	0x603E
AT_MQTT_RECONNECT_READ_FAILED	0x603F
AT_MQTT_RECONNECT_VALUE_IS_WRONG	0x6040
AT_MQTT_TOPIC_READ_FAILED	0x6041
AT_MQTT_TOPIC_IS_NULL	0x6042
AT_MQTT_TOPIC_IS_OVERLENGTH	0x6043
AT_MQTT_DATA_READ_FAILED	0x6044
AT_MQTT_DATA_IS_NULL	0x6045
AT_MQTT_DATA_IS_OVERLENGTH	0x6046
AT_MQTT_QOS_READ_FAILED	0x6047
AT_MQTT_QOS_VALUE_IS_WRONG	0x6048
AT_MQTT_RETAIN_READ_FAILED	0x6049
AT_MQTT_RETAIN_VALUE_IS_WRONG	0x604A
AT_MQTT_PUBLISH_LENGTH_READ_FAILED	0x604B
AT_MQTT_PUBLISH_LENGTH_VALUE_IS_WRONG	0x604C
AT_MQTT_RECV_LENGTH_IS_WRONG	0x604D
AT_MQTT_CREATE_SEMA_FAILED	0x604E
AT_MQTT_CREATE_EVENT_GROUP_FAILED	0x604F
AT_MQTT_URI_PARSE_FAILED	0x6050
AT_MQTT_IN_DISCONNECTED_STATE	0x6051
AT_MQTT_HOSTNAME_VERIFY_FAILED	0x6052

3.6.16 MQTT AT Notes

- When the MQTT connection ends, it will prompt the message +MQTTDISCONNECTED:<LinkID>.
- When the MQTT connection established, it will prompt the message +MQTTCONNECTED:<LinkID>,<scheme>,<"host">,<port>,<"path">,<reconnect>.

3.7 HTTP AT Commands

- [Introduction](#)
- [AT+HTTPCLIENT](#): Send HTTP Client Request.
- [AT+HTTPGETSIZE](#): Get HTTP Resource Size.
- [AT+HTTPCGET](#): Get HTTP Resource.
- [AT+HTTPCPOST](#): Post HTTP data of specified length.
- [AT+HTTPCPUT](#): Put HTTP data of specified length.
- [AT+HTTPURLCFG](#): Set or get long HTTP URL.
- [AT+HTTPCHEAD](#): Set or query HTTP request headers.
- [AT+HTTPCFG](#): Set HTTP Client Configuration.
- [HTTP AT Error Codes](#)

3.7.1 Introduction

Important: The default AT firmware supports all the AT commands mentioned on this page. If you don't need ESP32 to support HTTP commands, you can compile the ESP-AT project by following the steps in [Compile ESP-AT Project Locally](#) documentation. In the project configuration during the fifth step, make the following selections:

- Disable Component config > AT > AT http command support

3.7.2 AT+HTTPCLIENT: Send HTTP Client Request

Set Command

Command:

```
AT+HTTPCLIENT=<opt>,<content-type>,<"url">,[<"host">],[<"path">],<transport_type>[,  
↪<"data">],[<"http_req_header">],[<"http_req_header">][...]
```

Response:

```
+HTTPCLIENT:<size>,<data>  
  
OK
```

Parameters

- **<opt>**: method of HTTP client request.
 - 1: HEAD
 - 2: GET
 - 3: POST
 - 4: PUT
 - 5: DELETE
- **<content-type>**: data type of HTTP client request.
 - 0: application/x-www-form-urlencoded
 - 1: application/json
 - 2: multipart/form-data
 - 3: text/xml
- **<" url" >**: HTTP URL. The parameter can override the **<"host">** and **<"path">** parameters if they are null.
- **<" host" >**: domain name or IP address.

- **<" path" >**: HTTP Path.
- **<transport_type>**: HTTP Client transport type. Default: 1.
 - 1: HTTP_TRANSPORT_OVER_TCP
 - 2: HTTP_TRANSPORT_OVER_SSL
- **<" data" >**: If **<opt>** is a POST request, this parameter holds the data you send to the HTTP server. If not, this parameter does not exist, which means there is no need to input a comma to indicate this parameter.
- **<" http_req_header" >**: you can send more than one request header to the server.

Notes

- If the length of the entire command containing the URL exceeds 256 bytes, please use the [AT+HTTURLCFG](#) command to preset the URL first, and then set the **<"url">** parameter of this command to "".
- If the **url** parameter is not null, HTTP client will use it and ignore the **host** parameter and **path** parameter; If the **url** parameter is omitted or null string, HTTP client will use **host** parameter and **path** parameter.
- In some released firmware, HTTP client commands are not supported (see [ESP-AT Firmware Differences](#)), but you can enable it by `./build.py menuconfig>Component config>AT>AT http command support` and build the project (see [Compile ESP-AT Project Locally](#)).
- The command does not support redirection. After getting the status code 301 (permanent redirection) or 302 (temporary redirection) from the server, AT will not automatically redirect to the new URL address. You can use some tools to get the actual URL, and then access it using this command.
- If the length of the entire command containing the **<"data">** exceeds 256 bytes, please use the [AT+HTTPOST](#) command.
- To set more HTTP request headers, use the [AT+HTTPCHEAD](#) command.

Example

```
// HEAD Request
AT+HTTPCLIENT=1,0,"http://httpbin.org/get","httpbin.org","/get",1

// GET Request
AT+HTTPCLIENT=2,0,"http://httpbin.org/get","httpbin.org","/get",1

// POST Request
AT+HTTPCLIENT=3,0,"http://httpbin.org/post","httpbin.org","/post",1,"field1=value1&
↪field2=value2"
```

3.7.3 AT+HTTPGETSIZE: Get HTTP Resource Size

Set Command

Command:

```
AT+HTTPGETSIZE=<"url">[,<tx size>][,<rx size>][,<timeout>]
```

Response:

```
+HTTPGETSIZE:<size>
```

```
OK
```

Parameters

- **<" url" >**: HTTP URL. It is a string parameter and should be enclosed with quotes.
- **<tx size>**: HTTP send buffer size. Unit: byte. Default: 2048. Range: [0,10240].
- **<rx size>**: HTTP receive buffer size. Unit: byte. Default: 2048. Range: [0,10240].

- **<timeout>**: Network timeout. Unit: millisecond. Default: 5000. Range: [0,180000].
- **<size>**: HTTP resource size.

Note

- If the length of the entire command containing the URL exceeds 256 bytes, please use the [AT+HTTPURLCFG](#) command to preset the URL first, and then set the **<"url">** parameter of this command to "".
- To set HTTP request headers, use the [AT+HTTPHEAD](#) command to set them.

Example

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

3.7.4 AT+HTTPGET: Get HTTP Resource

Set Command

Command:

```
AT+HTTPGET=<"url">[,<tx size>][,<rx size>][,<timeout>]
```

Response:

```
+HTTPGET:<size>,<data>  
OK
```

Parameters

- **<" url" >**: HTTP URL. It is a string parameter and should be enclosed with quotes.
- **<tx size>**: HTTP send buffer size. Unit: byte. Default: 2048. Range: [0,10240].
- **<rx size>**: HTTP receive buffer size. Unit: byte. Default: 2048. Range: [0,10240].
- **<timeout>**: Network timeout. Unit: millisecond. Default: 5000. Range: [0,180000].

Note

- If the length of the entire command containing the URL exceeds 256 bytes, please use the [AT+HTTPURLCFG](#) command to preset the URL first, and then set the **<"url">** parameter of this command to "".
- To set HTTP request headers, use the [AT+HTTPHEAD](#) command to set them.

3.7.5 AT+HTTPPOST: Post HTTP data of specified length

Set Command

Command:

```
AT+HTTPPOST=<"url">,<length>[,<http_req_header_cnt>][,<"http_req_header">..<"http_↵  
req_header">]
```

Response:

```
OK  
  
>
```

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

If the transmission is successful, AT returns:

```
SEND OK
```

Otherwise, it returns:

```
SEND FAIL
```

Parameters

- <"url">: HTTP URL. It is a string parameter and should be enclosed with quotes.
- <length>: HTTP data length to POST. The maximum length is equal to the system allocable heap size.
- <http_req_header_cnt>: the number of <"http_req_header"> parameters.
- <"http_req_header">: HTTP request header. You can send more than one request header to the server.

Note

- If the length of the entire command containing the URL exceeds 256 bytes, please use the [AT+HTTPURLCFG](#) command to preset the URL first, and then set the <"url"> parameter of this command to "".
- the default type of content-type is application/x-www-form-urlencoded for this command.
- To set HTTP request headers, use the [AT+HTTPCHEAD](#) command to set them.
- Performance optimization: This command uses short connections. It establishes a new connection for each request (HTTPS requires an SSL handshake). For low latency, it is recommended to use [AT+CIPSTART](#) to establish a persistent connection. Construct HTTP request packets with the MCU and send them via [AT+CIPSEND](#). This avoids repeated handshake overhead.
- Memory management: This command creates and closes a connection each time, which generates TIME_WAIT state and occupies memory. If the minimum heap space keeps decreasing, you can optimize it in the following ways:
 - Reduce TCP Maximum Segment Lifetime (MSL): ./build.py menuconfig > Component config > LWIP > TCP > Maximum segment lifetime (MSL), then recompile the AT firmware. Note: Reducing MSL can shorten the duration of TIME_WAIT state, thereby releasing memory faster, but it cannot completely eliminate the TIME_WAIT state.
 - Use persistent TCP connections to avoid frequently creating and closing connections.

3.7.6 AT+HTTPCPUT: Put HTTP data of specified length

Set Command

Command:

```
AT+HTTPCPUT=<"url">,<length>[,<http_req_header_cnt>][,<"http_req_header">.."http\_req\_header">]
```

Response:

```
OK
```

```
>
```

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

If the transmission is successful, AT returns:

```
SEND OK
```

Otherwise, it returns:

```
SEND FAIL
```

Parameters

- **<" url" >**: HTTP URL. It is a string parameter and should be enclosed with quotes.
- **<length>**: HTTP data length to PUT. The maximum length is equal to the system allocable heap size.
- **<http_req_header_cnt>**: the number of **<" http_req_header" >** parameters.
- **<" http_req_header" >**: HTTP request header. You can send more than one request header to the server.

Note

- If the length of the entire command containing the URL exceeds 256 bytes, please use the [AT+HTTPURLCFG](#) command to preset the URL first, and then set the **<"url">** parameter of this command to **" "**.
- To set HTTP request headers, use the [AT+HTTPHEAD](#) command to set them.

3.7.7 AT+HTTPURLCFG: Set or get long HTTP URL

Query Command

Command:

```
AT+HTTPURLCFG?
```

Response:

```
[+HTTPURLCFG:<url length>,<data>]  
OK
```

Set Command

Command:

```
AT+HTTPURLCFG=<url length>
```

Response:

```
OK  
  
>
```

This response indicates that AT is ready for receiving serial data. You should enter the URL now, and when the URL length reaches the **<url length>** value, the system returns:

```
SET OK
```

Parameters

- **<url length>**: HTTP URL length. Unit: byte.
 - 0: Clean the HTTP URL configuration.
 - [8,8192]: Set the HTTP URL configuration.
- **<data>**: HTTP URL data.

3.7.8 AT+HTTPCHEAD: Set or query HTTP Request Headers

Query Command

Command:

```
AT+HTTPCHEAD?
```

Response:

```
+HTTPCHEAD:<index>,<"req_header">
```

```
OK
```

Set Command

Command:

```
AT+HTTPCHEAD=<req_header_len>
```

Response:

```
OK
```

```
>
```

The > symbol indicates that AT is ready to receive AT command data. At this point, you can enter the HTTP request header (in the format of `key: value`, no need to end with `\r\n`). When the data length reaches the value of parameter `<req_header_len>`, AT returns:

```
OK
```

Parameters

- **<index>**: Index value of HTTP request header.
- **<" req_header" >**: HTTP request header.
- **<req_header_len>**: HTTP request header length. Unit: byte.
 - 0: Clear all set HTTP request headers.
 - Other values: Set a new HTTP request header.

Note

- This command can only set one HTTP request header at a time, but it can be set multiple times to support multiple different HTTP request headers.
- The HTTP request headers configured by this command are global. Once set, all HTTP commands will carry these request headers.
- If the `key` in the HTTP request header set by this command is the same as that of other HTTP commands, the HTTP request header set by this command will be used.

Example

```
// Set the request header
AT+HTTPCHEAD=18
```

```
// After receiving the ">" symbol, enter the Range request header below to
```

(continues on next page)

(continued from previous page)

```
↪download only the first 256 bytes of the resource
Range: bytes=0-255

// Download HTTP resource
AT+HTTPCGET="https://docs.espressif.com/projects/esp-at/en/latest/esp32/index.html"
```

3.7.9 AT+HTTPCFG: Set HTTP Client Configuration

Set Command

Command:

```
AT+HTTPCFG=<auth_mode>[,<pki_number>][,<ca_number>]
```

Response:

```
OK
```

Parameters

- **<auth_mode>**:
 - 0: No authentication, in this case, <pki_number> and <ca_number> parameters are not required;
 - 1: ESP-AT provides HTTP client certificate for HTTP server CA certificate verification;
 - 2: ESP-AT HTTP client loads CA certificate to verify the server's certificate;
 - 3: Mutual authentication.
- **<pki_number>**: Index of certificate and private key. If there is only one certificate and private key, the value should be 0.
- **<ca_number>**: Index of CA. If there is only one CA, the value should be 0.

Notes

- By default, AT firmware does not support HTTP certificate configuration. You can enable it via `./build.py menuconfig > Component config > AT > AT http command support > AT HTTP authentication method`.
- The parameters configured by this command are global. Once set, all HTTP commands will share this configuration.
- If you want to use your own certificate at runtime, use the [AT+SYSMFG](#) command to update the HTTP certificate (for detailed steps, please refer to [AT+SYSMFG command examples](#), the certificate configuration method is the same as SSL certificates). If you want to pre-burn your own certificate, please refer to [How to Update PKI Configuration](#).
- If <auth_mode> is set to 2 or 3, to verify the validity period of the server certificate, please ensure that ESP32 has obtained the current time before sending other HTTP commands. (You can configure SNTP and obtain the current time by sending the [AT+CIPSNTPCFG](#) command, and query the current time by sending the [AT+CIPSNTPTIME?](#) command.)

3.7.10 HTTP AT Error Codes

When [AT+SYSLOG=1](#) is enabled, if an HTTP client request fails, AT will return an error code. The error code format is:

```
ERR CODE:0x010a7xxx
```

Where 01 is the module identifier, 0a is the module's response result to the executed AT command, and 7xxx represents an HTTP error code. If xxx falls within the range of standard HTTP status codes, it indicates a standard HTTP status code; otherwise, it indicates an internal error code specific to AT HTTP. The following table lists some common HTTP status codes. For more details, please refer to [RFC 2616](#).

HTTP Error Code (HTTP Status Code)	Description
0x7000	Connection Failed
0x7190 (400)	Bad Request
0x7191 (401)	Unauthorized
0x7192 (402)	Payment Required
0x7193 (403)	Forbidden
0x7194 (404)	Not Found
0x7195 (405)	Method Not Allowed
0x7196 (406)	Not Acceptable
0x7197 (407)	Proxy Authentication Required
0x7198 (408)	Request Timeout
0x7199 (409)	Conflict
0x719a (410)	Gone
0x719b (411)	Length Required
0x719c (412)	Precondition Failed
0x719d (413)	Request Entity Too Large
0x719e (414)	Request-URI Too Long
0x719f (415)	Unsupported Media Type
0x71a0 (416)	Requested Range Not Satisfiable
0x71a1 (417)	Expectation Failed
0x71f4 (500)	Internal Server Error
0x71f5 (501)	Not Implemented
0x71f6 (502)	Bad Gateway
0x71f7 (503)	Service Unavailable
0x71f8 (504)	Gateway Timeout
0x71f9 (505)	HTTP Version Not Supported

3.8 FileSystem AT Commands

- [Introduction](#)
- [AT+FS](#): Filesystem Operations.
- [AT+FSMOUNT](#): Mount or unmount Filesystem.

3.8.1 Introduction

Important: The default AT firmware does not support the AT commands listed on this page. If you need ESP32 to support FileSystem commands, you can compile the ESP-AT project by following the steps in [Compile ESP-AT Project Locally](#) documentation. In the project configuration during the fifth step, make the following selections:

- Enable Component `config>AT>AT FS command support`
-

3.8.2 AT+FS: Filesystem Operations

Set Command

Command:

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

Response:

```
OK
```

Parameters

- **<type>**: Filesystem type, LittleFS is recommended. You can select it in `Component config>AT>AT FS command support>Filesystem type` for AT FS command.
 - 0: FATFS
 - 1: LittleFS
- **<operation>**:
 - 0: Delete file.
 - 1: Write file.
 - 2: Read file.
 - 3: Query the size of the file.
 - 4: List files in a specific directory. Only root directory is currently supported.
 - 5: Calculate the MD5 hash of a file for verifying file integrity.
 - 6: Query the free space and total space of the filesystem.
- **<offset>**: apply to writing and reading operations only.
- **<length>**: data length, applying to writing and reading operations only.

Notes

- This command will automatically mount the filesystem. After the *AT+FS* filesystem operation is all done, it is strongly recommended to use the *AT+FSMOUNT=0* command to unmount the filesystem to free a large amount of RAM space.
- Please make sure that you have downloaded `at_customize.bin` before using this command. For more details, refer to [ESP-IDF Partition Tables](#) and [How to Customize Partitions](#).
- If the length of the read data is greater than the actual file length, only the actual data length of the file will be returned.
- If the operator is `write`, wrap return `>` after the write command, then you can send the data that you want to write. The length should be parameter `<length>`.
- If you need to ensure that the AT firmware is compatible with the legacy FATFS partition after an OTA upgrade, please select `FatFS (legacy partition, upgrade only, READ HELP FIRST)` at the following configuration path: `Component config>AT>AT FS command support>Filesystem type` for AT FS command.

Example

```
// delete a file.
AT+FS=1,0,"filename"

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

// read 100 bytes from offset 0 of a file.
AT+FS=1,2,"filename",0,100

// list all files in the root directory.
AT+FS=1,4,"."
```

(continues on next page)

(continued from previous page)

```
// calculate the MD5 hash of a file.  
AT+FS=1,5,"filename"  
  
// query the free space and total space of the filesystem.  
AT+FS=1,6
```

3.8.3 AT+FSMOUNT: Mount or unmount Filesystem

Set Command

Command:

```
AT+FSMOUNT=<mount>
```

Response:

```
OK
```

Parameters

- **<mount>**:
 - 0: Unmount filesystem
 - 1: Mount filesystem

Notes

- After the [AT+FS](#) filesystem operation is all done, it is strongly recommended to use the [AT+FSMOUNT=0](#) command to unmount the filesystem to free a large amount of RAM space.

Example

```
// unmount the filesystem manually  
AT+FSMOUNT=0  
  
// mount the filesystem manually  
AT+FSMOUNT=1
```

3.9 WebSocket AT Commands

- [Introduction](#)
- [AT+WSCFG](#): Set the WebSocket configuration.
- [AT+WSHEAD](#): Set or query WebSocket request headers.
- [AT+WSOPEN](#): Query or open a WebSocket connection.
- [AT+WSEND](#): Send data to a WebSocket connection.
- [AT+WSDATAFMT](#): Set WebSocket received data format.
- [AT+WSCLOSE](#): Close a WebSocket connection.
- [WebSocket Example](#)

3.9.1 Introduction

Important: The default AT firmware does not support the AT commands listed on this page. If you need ESP32 to support WebSocket commands, you can compile the ESP-AT project by following the steps in [Compile ESP-AT Project Locally](#) documentation. In the project configuration during the fifth step, make the following selections:

- Enable Component config > AT > AT WebSocket command support

3.9.2 AT+WSCFG: Set the WebSocket Configuration

Set Command

Command:

```
AT+WSCFG=<link_id>,<ping_intv_sec>,<ping_timeout_sec>[,<buffer_size>][,<auth_mode>,<pkc_number>,<ca_number>]
```

Response:

OK

or

ERROR

Parameters

- **<link_id>**: ID of the WebSocket connection. Range: [0,2], which means that AT can support up to three WebSocket connections.
- **<ping_intv_sec>**: WebSocket Ping interval. Unit: second. Range: [1,7200]. Default: 10, which means that WebSocket Ping packets are sent every 10 seconds by default.
- **<ping_timeout_sec>**: WebSocket Ping timeout. Unit: second. Range: [1,7200]. Default: 120, which means that by default, if the WebSocket Pong packet is not received within 120 seconds, the connection will be closed.
- **<buffer_size>**: WebSocket buffer size. Unit: byte. Range: [1,16384]. Default: 1024.
- **<auth_mode>**:
 - 0: No authentication. In this case <pkc_number> and <ca_number> are not required.
 - 1: The client provides the client certificate for the server to verify.
 - 2: The client loads CA certificate to verify the server's certificate.
 - 3: Mutual authentication.
- **<pkc_number>**: the index of certificate and private key. If there is only one certificate and private key, the value should be 0.
- **<ca_number>**: the index of CA. If there is only one CA, the value should be 0.

Notes

- This command should be configured before [AT+WSOPEN](#) command. Otherwise, it will not take effect.
- If you want to use your own certificate at runtime, use the [AT+SYSMFG](#) command to update the WebSocket certificate. If you want to pre-burn your own certificate, please refer to [How to Update PKI Configuration](#).
- If <auth_mode> is configured to 2 or 3, in order to check the server certificate validity period, please make sure ESP32 has obtained the current time before sending the [AT+WSOPEN](#) command. (You can send [AT+CIPSNTPCFG](#) command to configure SNTP and obtain the current time, and send [AT+CIPSNPTIME?](#) command to query the current time.)
- Mutual authentication example: [WebSocket Connection over TLS \(Mutual Authentication\)](#).

Example

```
// Set the ping interval to 30 seconds, ping timeout to 60 seconds, and buffer_
↪size to 4096 bytes for link_id: 0.
AT+WSCFG=0,30,60,4096
```

3.9.3 AT+WSHEAD: Set or query WebSocket Request Headers

Query Command

Command:

```
AT+WSHEAD?
```

Response:

```
+WSHEAD:<index>,<"req_header">
OK
```

Set Command

Command:

```
AT+WSHEAD=<req_header_len>
```

Response:

```
OK
>
```

The > symbol indicates that AT is ready to receive AT command data. At this point, you can enter the WebSocket request header (in the format of `key: value`, no need to end with `\r\n`). When the data length reaches the value of parameter `<req_header_len>`, AT returns:

```
OK
```

Parameters

- **<index>**: Index value of WebSocket request header.
- **<"req_header">**: WebSocket request header.
- **<req_header_len>**: WebSocket request header length. Unit: byte.
 - 0: Clear all set WebSocket request headers.
 - Other values: Set a new WebSocket request header.

Notes

- This command can only set one WebSocket request header at a time, but it can be set multiple times to support multiple different WebSocket request headers.
- The WebSocket request headers configured by this command are global. Once set, all WebSocket commands will carry these request headers.

Example

```
// Set the request header
AT+WSHEAD=49

// After receiving the ">" symbol, enter the authorization request header below.
AUTHORIZATION: Basic QTIzMzIyMDE5OTk6MTIzNDU2Nzg=

// Open a WebSocket connection
AT+WSOPEN=0, "wss://demo.piesocket.com/v3/channel_123?api_
↵key=VCXCEuvhGcBDP7XhiJJUDvR1e1D3eiVjgZ9VRiaV&notify_self"
```

3.9.4 AT+WSOPEN: Query or open a WebSocket Connection

Query Command

Command:

```
AT+WSOPEN?
```

Response:

When there is a connection, AT returns:

```
+WSOPEN:<link_id>,<state>,<"uri">
OK
```

When there is no connection, AT returns:

```
OK
```

Set Command

Command:

```
AT+WSOPEN=<link_id>,<"uri">[,<"subprotocol">][,<timeout_ms>][,<"auth">]
```

Response:

```
+WS_CONNECTED:<link_id>
OK
```

or

```
ERROR
```

Parameters

- **<link_id>**: ID of the WebSocket connection. Range: [0,2], which means that AT can support up to three WebSocket connections.
- **<state>**: The state of WebSocket connections.
 - 0: The WebSocket connection is closed.
 - 1: The WebSocket connection is reconnecting.
 - 2: The WebSocket connection is established.
 - 3: Receiving WebSocket Pong timeout or reading connection data error, waiting for reconnection.

- 4: The WebSocket connection Received close frame from the server side and is sending close frame to the server.
- **<" uri" >**: Uniform resource identifier of WebSocket server.
- **<" subprotocol" >**: The subprotocol of WebSocket (refer to [Section: RFC6455 1.9](#) for more details).
- **<timeout_ms>**: Timeout for establishing a WebSocket connection. Unit: millisecond. Range: [0,180000]. Default: 15000.
- **<" auth" >**: The authorization of WebSocket (refer to [Section: RFC6455 4.1.12](#) for more details).

Example

```
// uri parameter comes from https://www.piesocket.com/websocket-tester
AT+WSOPEN=0,"wss://demo.piesocket.com/v3/channel_123?api_
↪key=VCXCEuvhGcBDP7XhiJJUDvR1e1D3eiVjgZ9VRiaV&notify_self"
```

Detailed examples refer to: [WebSocket Example](#).

3.9.5 AT+WSEND: Send Data to a WebSocket Connection

Set Command

Command:

```
AT+WSEND=<link_id>,<length>[,<opcode>][,<timeout_ms>][,<fin>]
```

Response:

```
OK
>
```

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

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

```
ERROR
```

If data is transmitted successfully, the system returns:

```
SEND OK
```

Parameters

- **<link_id>**: ID of the WebSocket connection. Range: [0,2].
- **<length>**: Length of the data to be sent. Unit: byte. The maximum allowable length is the smaller of 16 KiB and the amount of heap memory currently available for allocation.
- **<opcode>**: The opcode in the WebSocket frame sent. Range: [0,0xF]. Default: 1, which means text frame. For details about opcode, please refer to [Section: RFC6455 5.2](#).
 - 0x0: continuation frame
 - 0x1: text frame
 - 0x2: binary frame
 - 0x3 - 0x7: reserved for further non-control frames
 - 0x8: connection close frame
 - 0x9: ping frame
 - 0xA: pong frame
 - 0xB - 0xF: reserved for further control frames
- **<timeout_ms>**: Send timeout. Unit: millisecond. Range: [0,60000]. Default: 10000.

- **<fin>**: Indicates whether the data frame is the last fragment. Default: 1.
 - 0: Indicates the data frame has subsequent fragments.
 - 1: Indicates the data frame is the last fragment.

3.9.6 AT+WSDATAFMT: Set WebSocket Received Data Format

Set Command

Function:

By default, when the ESP32 receives WebSocket data, it will output the data in the format +WS_DATA:<link_id>,<data_len>,<data>\r\n. This command allows you to configure the metadata output format when receiving data to meet different application requirements. The complete output format is as follows:

```
+WS_DATA:<link_id>[,<opcode>][,<fin>][,<payload_len>][,<payload_offset>][,<data_
↳len>],<data>[<\r\n>]
```

Command:

```
AT+WSDATAFMT=<link_id>[,<meta_data_level>][,<meta_data_mask>]
```

Response:

```
OK
```

or

```
ERROR
```

Parameters

- **<link_id>**: ID of the WebSocket connection. Range: [0,2].
- **<meta_data_level>**: Metadata output level. Default: 0. The meanings of each level are as follows:
 - 0: Chunked level. Each time data is received from the transport layer, metadata prefixed with +WS_DATA is added.
 - 1: Frame level. Metadata prefixed with +WS_DATA is only added before and after a complete WebSocket frame.
 - 2: Message level. Metadata prefixed with +WS_DATA is only added before and after a complete WebSocket message.
- **<meta_data_mask>**: A bitmask used to control the +WS_DATA output format. Setting a corresponding bit to 1 will include the field associated with that bit in the output. Default: 48 (in this case, only <data_len> and <\r\n> are output).
 - bit 0: <opcode>, WebSocket frame type
 - bit 1: <fin>, whether it is the last frame of the message
 - bit 2: <payload_len>, complete payload length
 - bit 3: <payload_offset>, offset of current data in the payload
 - bit 4: <data_len>, length of the current output data
 - bit 5: <\r\n>, newline at the end of data

Example

```
// Set the WebSocket connection with link_id 0 to use message level, only output_
↳payload_len.
AT+WSDATAFMT=0,2,4
```

3.9.7 AT+WSCLOSE: Close a WebSocket Connection

Set Command

Command:

```
AT+WSCLOSE=<link_id>
```

Response:

```
OK
```

Parameters

- **<link_id>**: ID of the WebSocket connection. Range: [0,2].

Notes

- This command triggers the TCP layer close process. Returning OK only indicates that the close instruction has been issued and does not guarantee that the peer has completely disconnected. To ensure the connection is fully closed, it is recommended to first send a WebSocket control frame with opcode 8 (Close Frame) using the [AT+WSEND](#) command before executing this command, and wait for the server to respond before closing.

Example

```
// Close the WebSocket connection whose link_id is 0
AT+WSCLOSE=0
```

3.10 ESP32 Ethernet AT Commands

- [Introduction](#)
- [Prerequisite](#)
- [AT+CIPETHMAC](#): Query or set the MAC address of the ESP32 Ethernet.
- [AT+CIPETH](#): Query or set the IP address of the ESP32 Ethernet.

3.10.1 Introduction

Important: The default AT firmware does not support the AT commands listed on this page. If you need ESP32 to support Ethernet commands, please refer to [How to Enable ESP-AT Ethernet](#) documentation and [Compile ESP-AT Project Locally](#).

3.10.2 Prerequisite

Before you run any Ethernet AT Commands, please make the following preparations:

Note: This prerequisite takes [ESP32-Ethernet-Kit](#) as an example. If you use other modules or development boards, please refer to corresponding datasheets for RX/TX pins.

- Change AT UART pins (because default AT UART pins are in conflict with the Ethernet function pins):
 - Open [factory_param_data.csv](#) file.
 - In the row of module WROVER-32, change `uart_tx_pin` from GPIO22 to GPIO2, `uart_rx_pin` from GPIO19 to GPIO4, `uart_cts_pin` from GPIO15 to GPIO1, and `uart_rts_pin` from GPIO14 to GPIO1 (flow control is optional and is not used here). See [How to Set AT Port Pins](#) for more information.
- Enable AT ethernet support. See [How to Enable ESP-AT Ethernet](#) for more information.
- Compile and flash the project onto ESP32-Ethernet-Kit.
- Connect your hardware:
 - Connect Host MCU (PC with USB to serial converter) to GPIO2 (TX) and GPIO4 (RX) of ESP32-Ethernet-Kit when the flow control function is not enabled.
 - Connect ESP32-Ethernet-Kit with Ethernet network.

3.10.3 AT+CIPETHMAC: Query or set the MAC Address of the ESP32 Ethernet

Query Command

Function:

Query the MAC address of the ESP32 Ethernet.

Command:

```
AT+CIPETHMAC?
```

Response:

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

Set Command

Function:

Set the MAC address of the ESP32 Ethernet.

Command:

```
AT+CIPETHMAC=<"mac">
```

Response:

```
OK
```

Parameter

- `<" mac" >`: string parameter showing the MAC address of the Ethernet interface.

Notes

- The default firmware does not support Ethernet AT commands (see [ESP-AT Firmware Differences](#)), but you can enable it by `./build.py menuconfig>Component config>AT>AT ethernet support` and compile the project (see [Compile ESP-AT Project Locally](#)).
- The configuration changes will be saved in the NVS partition if `AT+SYSSTORE=1`.
- Please make sure the MAC address of Ethernet interface you set is different from those of other interfaces.
- 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 addresses and cannot be set.

Example

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

3.10.4 AT+CIPETH: Query or set the IP Address of the the ESP32 Ethernet

Query Command

Function:

Query the IP address of the ESP32 Ethernet.

Command:

```
AT+CIPETH?
```

Response:

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

Set Command

Function:

Set the IP address of the ESP32 Ethernet.

Command:

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

Response:

```
OK
```

Parameters

- **<" ip" >**: string parameter showing the IP address of the ESP32 Ethernet.
- **<" gateway" >**: gateway.
- **<" netmask" >**: netmask.

Notes

- The default firmware does not support Ethernet AT commands (see [ESP-AT Firmware Differences](#)), but you can enable it by `./build.py menuconfig>Component config>AT>AT ethernet support` and compile the project (see [Compile ESP-AT Project Locally](#)).
- The configuration changes will be saved in the NVS partition if `AT+SYSSTORE=1`.
- This Set Command correlates with DHCP commands, such as `AT+CWDHCP`:
 - If static IP is enabled, DHCP will be disabled.
 - If DHCP is enabled, static IP will be disabled.
 - The last configuration overwrites the previous configuration.

Example

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

3.11 Signaling Test AT Commands

- [Introduction](#)
- [AT+FACTPLCP](#): Send with long or short PLCP (Physical Layer Convergence Procedure).

3.11.1 Introduction

Important: The default AT firmware supports all the AT commands mentioned on this page. If you don't need ESP32 to support signaling test commands, you can compile the ESP-AT project by following the steps in [Compile ESP-AT Project Locally](#) documentation. In the project configuration during the fifth step, make the following selections:

- Disable Component config > AT > AT signaling test command support

3.11.2 AT+FACTPLCP: Send with Long or Short PLCP

Set Command

Command:

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

Response:

```
OK
```

Parameters

- **<enable>**: Enable or disable manual configuration.
 - 0: Disable manual configuration. The default value for the parameter <tx_with_long> will be used.
 - 1: Enable manual configuration. The type of PLCP that AT sends depends on <tx_with_long>.
- **<tx_with_long>**: Send with long PLCP or short PLCP.
 - 0: Send with short PLCP (default).
 - 1: Send with long PLCP.

3.12 Web Server AT Commands

- [Introduction](#)
- [AT+WEBSERVER](#): Enable or disable Wi-Fi connection configuration via Web server.

3.12.1 Introduction

Important: The default AT firmware does not support the AT commands listed on this page. If you need ESP32 to support web server commands, you can compile the ESP-AT project by following the steps in [Compile ESP-AT Project Locally](#) documentation. In the project configuration during the fifth step, make the following selections:

- Enable Component config>AT>AT Web Server command support
-

3.12.2 AT+WEBSERVER: Enable or disable Wi-Fi connection configuration via Web server

Set Command

Command:

```
AT+WEBSERVER=<enable>,<server_port>,<connection_timeout>
```

Response:

```
OK
```

Parameters

- **<enable>**: Enable or disable Web server.
 - 0: Disable the Web server and release related resources.
 - 1: Enable Web server, which means that you can use WeChat or a browser to configure Wi-Fi connection information.
- **<server_port>**: The Web server port number.
- **<connection_timeout>**: The timeout for the every connection. Unit: second. Range:[21,60].

Notes

- There are two ways to provide the HTML files needed by the Web server. One is to use FAT file system, and you need to enable AT FS command at this time. The other is to use the embedded file to store HTML files (default setting).
- The default HTML file is [index.html](#) . If you need to customize the display format or text of the HTML file, you can directly modify this file. If you need to customize other content of the HTML file (e.g., add a field), you need to modify the corresponding source code file [at_web_server_cmd.c](#) .
- Please make sure that the maximum number of open sockets is not less than 12, you may change the number by `./build.py menuconfig>Component config>LWIP>Max number of open sockets` and compile the project (see [Compile ESP-AT Project Locally](#)).
- The default firmware does not support Web server AT commands (see [ESP-AT Firmware Differences](#)), but you can enable it by `./build.py menuconfig>Component config>AT>AT WEB Server command support` and compile the project (see [Compile ESP-AT Project Locally](#)).
- ESP-AT supports captive portals in ESP32 series of devices. See [example](#).
- For more examples, please refer to [Web Server AT Example](#).
- The command implementation is open-source. See the source code in [at/src/at_web_server_cmd.c](#).
- Please refer to [How to Implement OTA Upgrade](#) for more OTA commands.

Example

```
// Enable the Web server with port 80, and the timeout for the every connection is
↔50 seconds
AT+WEBSERVER=1,80,50

// Disable the Web server
AT+WEBSERVER=0
```

3.13 Driver AT Commands

- *Introduction*
- *AT+DRVADC*: Read ADC channel value.
- *AT+DRVPWMINIT*: Initialize PWM driver.
- *AT+DRVPWMDUTY*: Set PWM duty.
- *AT+DRVPWMFADE*: Set PWM fade.
- *AT+DRVI2CINIT*: Initialize I2C master driver.
- *AT+DRVI2CRD*: Read I2C data.
- *AT+DRVI2CWRDATA*: Write I2C data.
- *AT+DRVI2CWRBYTES*: Write no more than 4 bytes I2C data.
- *AT+DRVSPICONFGPIO*: Configure SPI GPIO.
- *AT+DRVSPINIT*: Initialize SPI master driver.
- *AT+DRVSPIRD*: Read SPI data.
- *AT+DRVSPIWR*: Write SPI data.

3.13.1 Introduction

Important: The default AT firmware does not support the AT commands listed on this page. If you need ESP32 to support driver commands, you can compile the ESP-AT project by following the steps in [Compile ESP-AT Project Locally](#) documentation. In the project configuration during the fifth step, make the following selections:

- Enable Component config>AT>AT driver command support
-

3.13.2 AT+DRVADC: Read ADC Channel Value

Set Command

Command:

```
AT+DRVADC=<channel>,<atten>
```

Response:

```
+DRVADC:<raw data>
```

```
OK
```

Parameters

- **<channel>**: ADC1 channel.

- For ESP32 devices, the range is [0,7].

CHANNEL	GPIO
0	GPIO36
1	GPIO37
2	GPIO38
3	GPIO39
4	GPIO32
5	GPIO33
6	GPIO34
7	GPIO35

- **<atten>**: attenuation.
 - 0: 0 dB attenuation, effective measurement range is [100, 950] mV.
 - 1: 2.5 dB attenuation, effective measurement range is [100, 1250] mV.
 - 2: 6 dB attenuation, effective measurement range is [150, 1750] mV.
 - 3: 11 dB attenuation, effective measurement range is [150, 2450] mV.
- **<raw data>**: ADC channel value.

Notes

- ESP-AT only supports ADC1.
- ESP32 support 12-bit width.
- For details on how to convert the channel value into voltage, please refer to [ADC Conversion](#).

Example

```
// For ESP32, 0 dB attenuation, effective measurement range is [100, 950] mV
// The returned 2048 means the voltage is 2048 / 4095 * 950 = 475.12 mV
AT+DRVADC=0,0
+DRVADC:2048

OK
```

3.13.3 AT+DRVPWMINIT: Initialize PWM Driver

Set Command

Command:

```
AT+DRVPWMINIT=<freq>,<duty_res>,<ch0_gpio>[,...,<ch3_gpio>]
```

Response:

```
OK
```

Parameters

- **<freq>**: LEDC timer frequency. Unit: Hz. Range: 1 Hz ~ 8 MHz.
- **<duty_res>**: LEDC channel duty resolution. Range: 0 ~ 20 bits.
- **<chx_gpio>**: LEDC output GPIO number of channel x. For example, if you want to use GPIO16 as channel 0, set <ch0_gpio> to 16.

Notes

- AT can support a maximum of 4 channels.
- The number of channels that you initialize using this command will determine how many channels you can set using other PWM commands, including [AT+DRVPWMDUTY](#) and [AT+DRVPWMFADE](#). For example, if you initialize two channels, you can only change the two channels' PWM duty using command AT+DRVPWMDUTY.
- The frequency and the duty resolution are interdependent. See [Supported Range of Frequency and Duty Resolutions](#) for more details.

Example

```
AT+DRVPWMINIT=5000,13,17,16,18,19 // set 4 channels; frequency: 5 kHz; duty_
↪resolution: 13 bits
AT+DRVPWMINIT=10000,10,17 // only use channel 0, frequency: 10 kHz; duty_
↪resolution: 10 bits; other PWM commands can only set one channel
```

3.13.4 AT+DRVPWMDUTY: Set PWM Duty

Set Command

Command:

```
AT+DRVPWMDUTY=<ch0_duty>[, ..., <ch3_duty>]
```

Response:

```
OK
```

Parameter

- **<duty>**: LEDC channel duty. Range: $[0, 2^{\text{duty_resolution}}]$.

Notes

- AT can support a maximum of 4 channels.
- If you do not want to set <duty> for a specific channel, just omit it.

Example

```
AT+DRVPWMDUTY=255,512 // set channel 0 to duty 255, set channel 1 to duty 512
AT+DRVPWMDUTY=,,0 // set channel 2 to duty 0
```

3.13.5 AT+DRVPWMFADE: Set PWM Fade

Set Command

Command:

```
AT+DRVPWMFADE=<ch0_target_duty>,<ch0_fade_time>[, ..., <ch3_target_duty>,<ch3_fade_
↪time>]
```

Response:

OK

Parameters

- **<target_duty>**: target duty of fading. Range: $[0, 2^{\text{duty_resolution}-1}]$.
- **<fade_time>**: the maximum time of fading. Unit: millisecond.

Notes

- AT can support a maximum of 4 channels.
- If you do not want to set **<target_duty>** and **<fade_time>** for a specific channel, just omit them.

Example

```
AT+DRVPMWFADE=,,0,1000 // use one second to change channel 1 duty to 0
AT+DRVPMWFADE=1024,1000,0,2000, // use one second time to change channel 0 duty_
↪to 1024, two seconds to change channel 1 duty to 0
```

3.13.6 AT+DRVI2CINIT: Initialize I2C Master Driver

Set Command

Command:

AT+DRVI2CINIT=<num>,<scl_io>,<sda_io>,<clock>

Response:

OK

Parameters

- **<num>**: I2C port number. Range: 0 ~ 1. If the following parameters are not set, AT will deinitialize the I2C port.
- **<scl_io>**: GPIO number for I2C SCL signal.
- **<sda_io>**: GPIO number for I2C SDA signal.
- **<clock>**: I2C clock frequency for master mode. Unit: Hz. Maximum: 1 MHz.

Note

- This command only supports I2C masters.

Example

```
AT+DRVI2CINIT=0,25,26,1000 // initialize I2C0; GPIO25 is SCL; GPIO26 is SDA; I2C_
↪clock is 1 kHz
AT+DRVI2CINIT=0 // deinitialize I2C0
```

3.13.7 AT+DRVI2CRD: Read I2C Data

Set Command

Command:

```
AT+DRVI2CRD=<num>,<address>,<length>
```

Response:

```
+DRVI2CRD:<read data>
OK
```

Parameters

- **<num>**: I2C port number. Range: 0 ~ 1.
- **<address>**: I2C slave device address.
 - 7-bit address: 0 ~ 0x7F.
 - 10-bit address: The first seven bits of the first byte are the combination 1111 0XX of which the last two bits (XX) are the two Most Significant Bits (MSBs) of the 10-bit address. For example, if the 10-bit address is 0x2FF (b' 101111111), the input address should be 0x7AFF (b' 11110101111111).
- **<length>**: I2C data length. Range: 1 ~ 2048.
- **<read data>**: I2C data.

Note

- I2C transmission timeout is one second.

Example

```
AT+DRVI2CRD=0,0x34,1      // I2C0 reads one byte data from address 0x34
AT+DRVI2CRD=0,0x7AFF,1    // I2C0 reads one byte data from 10-bit address 0x2FF

// I2C0 reads address 0x34, register address 0x27, read 2 bytes
AT+DRVI2CWRBYTES=0,0x34,1,0x27 // I2C0 first writes device address 0x34,
↪register address 0x27
AT+DRVI2CRD=0,0x34,2      // I2C0 reads 2 bytes
```

3.13.8 AT+DRVI2CWRDATA: Write I2C Data

Set Command

Command:

```
AT+DRVI2CWRDATA=<num>,<address>,<length>
```

Response:

```
OK
>
```

This response indicates that you should enter the data you want to write. When the requirement of data length is met, the data transmission starts.

If the data is transmitted successfully, AT returns:

OK

If the data transmission fails, AT returns:

ERROR

Parameters

- **<num>**: I2C port number. Range: 0 ~ 1.
- **<address>**: I2C slave device address.
 - 7-bit address: 0 ~ 0x7F.
 - 10-bit address: The first seven bits of the first byte are the combination 1111 0XX of which the last two bits (XX) are the two Most Significant Bits (MSBs) of the 10-bit address. For example, if the 10-bit address is 0x2FF (b' 101111111), the input address should be 0x7AFF (b' 11110101111111).
- **<length>**: I2C data length. Range: 1 ~ 2048.

Note

- I2C transmission timeout is one second.

Example

AT+DRVI2CWRDATA=0,0x34,10 // I2C0 writes 10 bytes data to address 0x34

3.13.9 AT+DRVI2CWRBYTES: Write No More Than 4 Bytes I2C Data

Set Command

Command:

AT+DRVI2CWRBYTES=<num>,<address>,<length>,<data>

Response:

OK

Parameters

- **<num>**: I2C port number. Range: 0 ~ 1.
- **<address>**: I2C slave device address.
 - 7-bit address: 0 ~ 0x7F.
 - 10-bit address: The first seven bits of the first byte are the combination 1111 0XX of which the last two bits (XX) are the two Most Significant Bits (MSBs) of the 10-bit address. For example, if the 10-bit address is 0x2FF (b' 101111111), the input address should be 0x7AFF (b' 11110101111111).
- **<length>**: the length of the I2C data you want to write. Range: 1 ~ 4 bytes.
- **<data>**: the data of <length> long. Range: 0 ~ 0xFFFFFFFF.

Note

- I2C transmission timeout is one second.

Example

```

AT+DRVI2CWRBYTES=0,0x34,2,0x1234      // I2C0 writes 2 bytes data 0x1234 to address_
↪0x34
AT+DRVI2CWRBYTES=0,0x7AFF,2,0x1234    // I2C0 writes 2 bytes data 0x1234 to 10-bit_
↪address 0x2FF

// I2C0 writes address 0x34; register address: 0x27; data: c0xFF
AT+DRVI2CWRBYTES=0,0x34,2,0x27FF

```

3.13.10 AT+DRVSPICONFGPIO: Configure SPI GPIO**Set Command****Command:**

```
AT+DRVSPICONFGPIO=<mosi>,<miso>,<sclk>,<cs>
```

Response:

```
OK
```

Parameters

- **<mosi>**: GPIO pin for Master Out Slave In signal.
- **<miso>**: GPIO pin for Master In Slave Out signal, or -1 if not used.
- **<sclk>**: GPIO pin for SPI Clock signal.
- **<cs>**: GPIO pin for slave selection signal, or -1 if not used.

3.13.11 AT+DRVSPINIT: Initialize SPI Master Driver**Set Command****Command:**

```
AT+DRVSPINIT=<clock>,<mode>,<cmd_bit>,<addr_bit>,<dma_chan>[,bits_msb]
```

Response:

```
OK
```

Parameters

- **<clock>**: Clock speed, divisors of 80 MHz. Unit: Hz. Maximum: 40 MHz.
- **<mode>**: SPI mode. Range: 0 ~ 3.
- **<cmd_bit>**: Default amount of bits in command phase. Range: 0 ~ 16.
- **<addr_bit>**: Default amount of bits in address phase. Range: 0 ~ 64.
- **<dma_chan>**: Either channel 1 or 2, or 0 in the case when no DMA is required.
- **<bits_msb>**: SPI data format:
 - bit 0:
 - * 0: Transmit MSB first (default).
 - * 1: Transmit LSB first.
 - bit 1:
 - * 0: Receive data MSB first (default).
 - * 1: Receive data LSB first.

Note

- You should configure SPI GPIO before SPI initialization.

Example

```
AT+DRVSPIINIT=102400,0,0,0,0,3 // SPI clock: 100 kHz; mode: 0; both command and
↳address bits are 0; not use DMA; transmit and receive LSB first
OK
AT+DRVSPIINIT=0 // delete SPI Driver
OK
```

3.13.12 AT+DRVSPIRD: Read SPI Data

Set Command**Command:**

```
AT+DRVSPIRD=<data_len>[,<cmd>,<cmd_len>][,<addr>,<addr_len>]
```

Response:

```
+DRVSPIRD:<read data>
OK
```

Parameters

- **<data_len>**: length of SPI data you want to read. Range: 1 ~ 4092 bytes.
- **<cmd>**: command data. The length of the data is set in **<cmd_len>**.
- **<cmd_len>**: command length in this transaction. Range: 0 ~ 2 bytes.
- **<addr>**: command address. The length of the address is set in **<addr_len>**.
- **<addr_len>**: The address length in this transaction. Range: 0 ~ 4 bytes.

Note

- If you do not use DMA, the maximum **<data_len>** you can set is 64 bytes each time.

Example

```
AT+DRVSPIRD=2 // read 2 bytes data
+DRVI2CREAD:ffff
OK

AT+DRVSPIRD=2,0x03,1,0x001000,3 // read 2 bytes data; <cmd> is 0x03; <cmd_len> is
↳1 byte; <addr> is 0x1000; <addr_len> is 3 bytes
+DRVI2CREAD:ffff
OK
```

3.13.13 AT+DRVSPIWR: Write SPI Data

Set Command**Command:**

```
AT+DRVSPIWR=<data_len>[,<cmd>,<cmd_len>][,<addr>,<addr_len>]
```

Response:

When <data_len> is larger than 0, AT returns:

```
OK
>
```

This response indicates that you should enter the data you want to write. When the requirement of data length is met, the data transmission starts.

If the data is transmitted successfully, AT returns:

```
OK
```

When <data_len> is equal to 0, which means AT transmits commands and addresses only, and no SPI data, AT returns:

```
OK
```

Parameters

- **<data_len>**: SPI data length. Range: 0 ~ 4092.
- **<cmd>**: command data. The length of the data is set in <cmd_len>.
- **<cmd_len>**: command length in this transaction. Range: 0 ~ 2 bytes.
- **<addr>**: command address. The length of the address is set in <addr_len>.
- **<addr_len>**: The address length in this transaction. Range: 0 ~ 4 bytes.

Note

- If you do not use DMA, the maximum <data_len> you can set is 64 bytes each time.

Example

```
AT+DRVSPIWR=2 // write 2 bytes data
OK
> // begin receiving serial data
OK

AT+DRVSPIWR=0,0x03,1,0x001000,3 // write 0 byte data; <cmd> is 0x03; <cmd_len> is 1 byte; <addr> is 0x1000; <addr_len> is 3 bytes
OK
```

3.14 User AT Commands

- *Introduction*
- **AT+USERRAM**: Operate user's free RAM.
- **AT+USEROTA**: Upgrade the firmware according to the specified URL.
- **AT+USERWKMCFG**: Configure how AT wakes up MCU.
- **AT+USERMCUSLEEP**: MCU indicates its sleep state.
- **AT+USERDOCS**: Query the ESP-AT user guide for current firmware.

3.14.1 Introduction

Important: The default AT firmware supports all the AT commands mentioned on this page. If you don't need ESP32 to support user commands, you can compile the ESP-AT project by following the steps in [Compile ESP-AT Project Locally](#) documentation. In the project configuration during the fifth step, make the following selections:

- Disable Component config>AT>AT user command support
-

3.14.2 AT+USERRAM: Operate user's free RAM

Query Command

Function:

Query the current available user's RAM size.

Command:

```
AT+USERRAM?
```

Response:

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

Set Command

Function:

Operate user's free RAM

Command:

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

Response:

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

Parameters

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

Notes

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

Example

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

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

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

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

3.14.3 AT+USEROTA: Upgrade the Firmware According to the Specified URL

ESP-AT upgrades firmware at runtime by downloading the new firmware from a specific URL.

Set Command

Function:

Upgrade to the firmware version specified by the URL.

Command:

```
AT+USEROTA=<url len>
```

Response:

```
OK

>
```

This response indicates that AT is ready for receiving URL. You should enter the URL, and when the URL length reaches the `<url len>` value, the system returns:

```
Recv <url len> bytes
```

After AT outputs the above information, the upgrade process starts. If the upgrade process is complete, the system return:

```
OK
```

If the parameter is wrong or firmware upgrades fails, the system returns:

```
ERROR
```

Parameters

- **<url len>**: URL length. Maximum: 8192 bytes.

Note

- You can [Download AT firmware from GitHub Actions](#), or you can generate AT firmware from [Compile ESP-AT Project](#) as well.
- OTA firmware is `build/esp-at.bin`.
- The speed of the upgrade depends on the network status.
- If the upgrade fails due to unfavorable network conditions, AT will return `ERROR`. Please wait for some time before retrying.
- Downgrading to an older version is not recommended due to potential compatibility issues and the risk of operational failure. If you still prefer downgrading to an older version, please test and verify the functionality based on your product.
- After you upgrade the AT firmware, you are suggested to call the command `AT+RESTORE` to restore the factory default settings.
- `AT+USEROTA` supports HTTP and HTTPS.
- After AT outputs the `>` character, the special characters in the URL does not need to be escaped through the escape character, and it does not need to end with a new line (CR-LF).
- When the URL is HTTPS, SSL verification is not recommended. If SSL verification is required, you need to generate your own PKI files and download them into the corresponding partition, and then load the certificates in the code implemented by the `AT+USEROTA` command. Please refer to [How to Update PKI Configuration](#) for PKI files. For `AT+USEROTA` command, ESP-AT project provides an example of `USEROTA`.
- Please refer to [How to Implement OTA Upgrade](#) for more OTA commands.

Example

```
AT+USEROTA=36
OK
>
Recv 36 bytes
OK
```

3.14.4 AT+USERWKMCCCFG: Configure How AT Wakes Up MCU

Set Command

Function:

This command configures how AT checks whether MCU is in awake state and how to wake it up.

- If MCU is in awake state, AT will directly send the data to it without waking it up.
- If MCU is in sleep state and AT is ready to actively send the data to MCU (the data sent actively is same to [ESP-AT Message Reports](#)), AT will send wake signals to wake it up first and then send the data to it. Wake signals will be cleared after MCU is waked up or timeout.

Command:

```
AT+USERWKMCCCFG=<enable>,<wake mode>,<wake number>,<wake signal>,<delay time>[,
↪<check mcu awake method>]
```

Response:

OK

Parameters

- **<enable>**: Enable or disable wake-up configuration.
 - 0: Disable wake-up configuration
 - 1: Enable wake-up configuration
- **<wake mode>**: wake mode.
 - 1: GPIO wake-up
 - 2: UART wake-up
- **<wake number>**: It means differently depending on the parameter <wake mode>.
 - If <wake mode> is 1, <wake number> represents GPIO number for wake-up. You need to ensure that the configured GPIO wake-up pin is not used for other purposes. Otherwise, you need to perform compatibility processing.
 - If <wake mode> is 2, <wake number> represents UART number. Currently, only 1 is supported for <wake number>, which means only UART1 can wake up MCU.
- **<wake signal>**: It means differently depending on the parameter <wake mode>.
 - If <wake mode> is 1, <wake signal> represents wake-up level.
 - * 0: low level
 - * 1: high level
 - If <wake mode> is 2, <wake signal> represents wake-up byte. Range: [0,255].
- **<delay time>**: Maximum waiting time. Unit: milliseconds. Range: [0,60000]. It means differently depending on the parameter <wake mode>.
 - If <wake mode> is 1, the <wake signal> level will be held on during the <delay time>. After the <delay time> is reached, the <wake signal> level is reversed.
 - If <wake mode> is 2, AT will send <wake signal> byte immediately and wait until timeout.
- **<check mcu awake method>**: AT checks whether MCU is in awake state.
 - bit 0: Whether to enable [AT+USERMCUSLEEP](#) command linkage. Enabled by default. That is, when receiving AT+USERMCUSLEEP=0 command from MCU, AT knows that MCU is in awake state; when receiving AT+USERMCUSLEEP=1 command, AT knows that MCU is in sleep.
 - bit 1: Whether to enable [AT+SLEEP=0/1/2/3](#) command linkage. Disabled by default. That is, when receiving AT+SLEEP=0 command, AT knows that MCU is in awake state; when receiving AT+SLEEP=1/2/3 command, AT knows that MCU is in sleep.
 - bit 2: Whether to enable the function of indicating that the MCU wakes up after <delay time> timeout. Disabled by default. When disabled, the MCU is indicated as sleeping after <delay time>; when enabled, the MCU is indicated as awake after <delay time>. Bit 2 must be used together with bit 0 or bit 1. If used alone, since the MCU is awake by default, it will not wake up the MCU each time.
 - bit 3 (not implemented yet): Whether to enable the function of indicating MCU state via GPIO. Unsupported by default.

Notes

- This command needs to be configured only once.
- Each time before the AT actively sends data to MCU, it will send a wake-up signal first and then enter the waiting time. When <delay time> is reached, it will directly send data. This wait timeout will reduce the transmission efficiency with MCU.
- If AT receives a wake-up event defined by bit 0 or bit 1 in <check mcu awake method> before the <delay time> (in milliseconds) expires, the wake-up state is cleared immediately. Otherwise, AT continues waiting until the <delay time> times out and then automatically clears the wake-up state after the timeout.

Example

```
// Enable wake-up MCU configuration. Before each time the AT sends data to the MCU,
↪ it will first use the GPIO18 pin of the Wi-Fi module to wake up the MCU at a
```

(continues on next page)

(continued from previous page)

```
↪ high level and hold on the high level for 10 seconds.  
AT+USERWKMUCCFG=1,1,18,1,10000,3  
  
// Enable wake-up configuration  
AT+USERWKMUCCFG=0
```

3.14.5 AT+USERMCUSLEEP: MCU Indicates Its Sleep State

Set Command

Function:

This command can only be taken effect when the <check mcu wake method> Bit 0 of the [AT+USERWKMUCCFG](#) command is configured. It used to inform AT that the current MCU sleep state.

Command:

```
AT+USERMCUSLEEP=<state>
```

Response:

```
OK
```

Parameters

- <state>:
 - 0: Indicates that MCU is in awake state.
 - 1: Indicates that MCU is in sleep state.

Example

```
// MCU tells AT that the current MCU is in awake state  
AT+USERMCUSLEEP=0
```

3.14.6 AT+USERDOCS: Query the ESP-AT User Guide for Current Firmware

Query Command

Function:

Query the ESP-AT English and Chinese user guide for current running firmware.

Command:

```
AT+USERDOCS?
```

Response:

```
+USERDOCS:<"en url">  
+USERDOCS:<"cn url">  
  
OK
```

Parameters

- <" en url" >: the URL for English document
- <" cn url" >: the URL for Chinese document

Example

```
AT+USERDOCS?
+USERDOCS:"https://docs.espressif.com/projects/esp-at/en/latest/esp32/index.html"
+USERDOCS:"https://docs.espressif.com/projects/esp-at/zh_CN/latest/esp32/index.html"
↪
OK
```

It is strongly recommended to read the following sections for some common information on AT commands before you dive into the details of each command.

- [AT Command Types](#)
- [AT Commands with Configuration Saved in the Flash](#)
- [AT Messages](#)
- [AT Logs](#)

3.15 AT Command Types

Generic AT command has four types:

Type	Command Format	Description
Test Command	AT+<CommandNam	Query the Set Commands' internal parameters and their range of values.
Query Command	AT+<CommandNam	Return the current value of parameters.
Set Command	AT+<CommandNam ...>	Set the value of user-defined parameters in commands, and run these commands.
Execute Command	AT+<CommandNam	Run commands with no user-defined parameters.

- Not all AT commands support all of the four types mentioned above.
- Currently, only strings and integer numbers are supported as input parameters in AT commands.
- Angle brackets < > designate parameters that can not be omitted.
- **Square brackets [] designate optional parameters that can be omitted. The default value of the parameter will be used instead when you omit it. Below are examples of entering the command [AT+CWJAP](#) with some parameters omitted.**

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

- **If the parameter you want to omit is followed by a parameter(s), you must give a , to indicate it.**

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

- **String values need to be included in double quotation marks.**

```
AT+CWSAP="ESP756290", "21030826", 1, 4
```

- Escape character syntax is needed if a string contains special characters. The characters that need to be escaped are , , " , and \:
 - \\ : escape the backslash itself
 - \ , : escape comma which is not used to separate each parameter

- \": escape double quotation mark which is not used to mark string input
 - \<any>: escape <any> character means that drop backslash symbol and only use <any> character
- **Escape is needed in AT commands only, not elsewhere. For example, when AT command port prints > and wait for your input, the input does not need to be escaped.**

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

- The default baud rate of AT command is 115200.
- The length of each AT command should be no more than 256 bytes.
- AT commands end with a new-line (CR-LF), so the serial tool should be set to “New Line Mode” .
- Definitions of AT command error codes are provided in [AT API Reference](#):
 - [esp_at_error_code](#)
 - [esp_at_para_parse_result_type](#)
 - [esp_at_result_code_string_index](#)

3.16 AT Commands with Configuration Saved in the Flash

Configuration settings entered by the following AT Commands will always be saved in the flash NVS Area, so they can be automatically restored on reset:

- [AT+UART_DEF](#): AT+UART_DEF=115200,8,1,0,3
- [AT+SAVETRANSLINK](#): AT+SAVETRANSLINK=1,"192.168.6.10",1001
- [AT+CWAUTOCONN](#): AT+CWAUTOCONN=1

Saving of configuration settings by several other commands can be switched on or off with [AT+SYSTORE](#) command, which is mentioned in the Note section of these commands.

Note: The parameters of AT commands are saved based on [NVS](#) library. Therefore, if the command is configured with the same parameter value, flash will not be written; If the command is configured with the different parameter value, flash will not be erased frequently.

3.17 AT Messages

There are two types of ESP-AT messages returned from the ESP-AT command port:

- ESP-AT Response Messages (passive)
Each ESP-AT command input returns response messages to tell the sender the result of the ESP-AT command. The last message in the response is either OK or ERROR.

Table 4: ESP-AT Response Messages

AT Response Messages	Description
OK	AT command process done and return OK
ERROR	AT command error or error occurred during the execution
SEND OK	Data has been sent to the protocol stack (specific to AT+CIPSEND and AT+CIPSENDEX command). It doesn't mean that the data has been sent to the opposite end
SEND FAIL	Error occurred during sending the data to the protocol stack (specific to AT+CIPSEND and AT+CIPSENDEX command)
SET OK	The URL has been set successfully (specific to AT+HTTURLCFG command)
+<Command Name>: . . .	Response to the sender that describes AT command process results in details

- ESP-AT Message Reports (active)

ESP-AT will report important state changes or messages in the system.

Table 5: ESP-AT Message Reports

ESP-AT Message Report	Description
ready	The ESP-AT firmware is ready
busy p...	Busy processing. The system is in process of handling the previous command, thus CANNOT accept the new input
ERR CODE:<0x%08x>	Error code for different commands
Will force to restart!!!	Module restart right now
smartconfig type:<xxx>	Smartconfig type
Smart get wifi info	Smartconfig has got the SSID and PASSWORD information
+SCRD:<length>,<reserved data>	ESP-Touch v2 has got the reserved information
smartconfig connected wifi	Smartconfig done. ESP-AT has connected to the Wi-Fi
WIFI CONNECTED	Wi-Fi station interface has connected to an AP
WIFI GOT IP	Wi-Fi station interface has got the IPv4 address
WIFI GOT IPv6 LL	Wi-Fi station interface has got the IPv6 LinkLocal address
WIFI GOT IPv6 GL	Wi-Fi station interface has got the IPv6 Global address
WIFI DISCONNECT	Wi-Fi station interface has disconnected from an AP
+ETH_CONNECTED	Ethernet interface has connected
+ETH_GOT_IP	Ethernet interface has got the IPv4 address
+ETH_DISCONNECTED	Ethernet interface has disconnected
[<conn_id>]CONNECT	A network connection of which ID is <conn_id> is established (ID=0 by default)
[<conn_id>]CLOSED	A network connection of which ID is <conn_id> ends (ID=0 by default)
+ERRNO:<error_code>	Error code when command execution fails. – When TCP connection, UDP transmission, or SSL server fails, the error code is returned in priority order: first, the socket's SO_ERROR; if SO_ERROR is 0, then LwIP's errno; if errno is 0, then the error code defined by ESP-AT. – When SSL connection establishment fails, the error code is returned in priority order: first, certificate-related error code; if certificate-related error code is 0, then mbedTLS error code; if mbedTLS error code is 0, then esp-tls error code; if esp-tls error code is 0, then the error code defined by ESP-AT.
+LINK_CONN	Detailed connection information of TCP/UDP/SSL
+STA_CONNECTED: <sta_mac>	A station has connected to the Wi-Fi softAP interface of ESP-AT
+DIST_STA_IP: <sta_mac>,<sta_ip>	The Wi-Fi softAP interface of ESP-AT distributes an IP address to the station
+STA_DISCONNECTED: <sta_mac>	A station disconnected from the Wi-Fi softAP interface of ESP-AT
>	ESP-AT is waiting for more data to be received
Recv <xxx> bytes	ESP-AT has already received <xxx> bytes from the ESP-AT command port

continues on next page

Table 5 – continued from previous page

ESP-AT Message Report	Description
+IPD	ESP-AT received the data from the network when AT is not in <i>Passthrough Mode</i>. The message formats are as follows: - +IPD,<length> will be outputted if AT+CIPMUX=0 and AT+CIPRECVMODE=1. - +IPD,<link_id>,<length> will be outputted if AT+CIPMUX=1 and AT+CIPRECVMODE=<link_id>,1. - +IPD,<length>:<data> will be outputted if AT+CIPMUX=0, AT+CIPRECVMODE=0, and AT+CIPDINFO=0. - +IPD,<link_id>,<length>:<data> will be outputted if AT+CIPMUX=1, AT+CIPRECVMODE=<link_id>,0, and AT+CIPDINFO=0. - +IPD,<length>,<"remote_ip">,<remote_port>:<data> will be outputted if AT+CIPMUX=0, AT+CIPRECVMODE=0, and AT+CIPDINFO=1. - +IPD,<link_id>,<length>,<"remote_ip">,<remote_port>:<data> will be outputted if AT+CIPMUX=1, AT+CIPRECVMODE=<link_id>,0, and AT+CIPDINFO=1. The link_id represents the connection ID, length indicates the data length, remote_ip refers to the remote IP address, remote_port represents the remote port number, and data denotes the actual data. Note: In the case of a SSL connection, in passive receive mode (AT+CIPRECVMODE=1), the length in the AT command response may not match the actual readable SSL data length. This is because AT prioritizes returning the readable data length at the SSL layer. If the SSL layer has no readable data, AT will return the readable data length at the socket layer.
The Data in <i>Passthrough Mode</i>	ESP-AT received the data from the network or Bluetooth when AT is in <i>Passthrough Mode</i>
SEND Canceled	Cancel to send in <i>normal sending mode</i>
Have <xxx> Connections	Has reached the maximum connection counts for server
+QUIT	ESP-AT quits from the <i>Passthrough Mode</i>
NO CERT FOUND	No valid device certificate found in custom partition
NO PRVT_KEY FOUND	No valid private key found in custom partition
NO CA FOUND	No valid CA certificate found in custom partition
+TIME_UPDATED	The system time has been updated. Only after sending the <i>AT+CIPSNTPCFG</i> command or power on will this message be outputted if the module fetches a new time from the SNTP server.

continues on next page

Table 5 – continued from previous page

ESP-AT Message Report	Description
+MQTTCONNECTED	MQTT connected to the broker
+MQTTDISCONNECTED	MQTT disconnected from the broker
+MQTTSUBRECV	MQTT received the data from the broker
+MQTTPUB:FAIL	MQTT failed to publish data
+MQTTPUB:OK	MQTT publish data done
+BLECONN	A Bluetooth LE connection established
+BLEDISCONN	A Bluetooth LE connection ends
+READ	A read operation from Bluetooth LE connection
+WRITE	A write operation from Bluetooth LE connection
+NOTIFY	A notification from Bluetooth LE connection
+INDICATE	An indication from Bluetooth LE connection
+BLESECNTFYKEY	Bluetooth LE SMP key
+BLESECREQ:<conn_index>	Received an encrypted pairing request from a Bluetooth LE connection.
+BLEAUTHCMPL:<conn_index>,<enc_result>	Bluetooth LE SMP pairing completed
+SSID:<ssid>	In BluFi network configuration mode, the ESP32 received the SSID sent from the phone.
+PASSWORD:<password>	In BluFi network configuration mode, the ESP32 received the password sent from the phone.
+BLUFIDATA:<len>,<data>	The ESP32 received customized data from the phone over BluFi.
+WS_DISCONNECTED:<link_id>	The WebSocket connection of ID <link_id> is disconnected.
+WS_CONNECTED:<link_id>	The WebSocket connection of ID <link_id> is established.
+WS_DATA:<link_id>,<data_len>,<data>	The WebSocket connection of ID <link_id> has received the data. You can use the AT+WSDATAFMT command to control the received data format.
+WS_CLOSED:<link_id>	The WebSocket connection of ID <link_id> is closed.
+BLESCANDONE	Scan finished
+BLESECKEYREQ:<conn_index>	The peer has accepted the pairing request, and the ESP device can enter the key.
+BLE_TT_MODE	Entered BLE passthrough mode.
+WRITE:<conn_index>	In passthrough mode, when enabling notify or indicate, the message prefixed with “+WRITE:” will be printed.

3.18 AT Logs

Through the `Log Output` pin described in [Hardware Connection](#), ESP-AT outputs rich log information to help you understand its runtime status. You can also [enable different log configurations](#) to view more detailed log information. The following table lists some common log messages and their explanations:

Table 6: ESP-AT Logs

AT Log	Description
at-common: write dlen:0	The RX of the AT command port received unexpected data, usually containing multiple \0 characters. If you are communicating with the MCU via UART, please check the following: • Whether the UART RX pin of ESP32 is correctly connected to the TX pin of the MCU • Whether the MCU is sending data not in AT command format • Whether the MCU is correctly configured for UART baud rate, data bits, stop bits, parity, and flow control • Whether the GND of the MCU is correctly connected to the GND pin of ESP32
at-init: at param mode: <mode>	Parameter mode during AT initialization • 0: No parameters • 1: Use parameters in the mfg_nvs partition • 2: Use parameters in the factory_param partition
at-init: v4.1.0.0 (<src>)	AT firmware version number and the source of the corresponding firmware build
at-wifi: negotiated phy mode: <phy_mode>	PHY mode negotiated when Wi-Fi Station connects to AP • 0: Low Rate • 1: 11b • 2: 11g • 3: 11a • 4: HT20 • 5: HT40 • 6: HE20 • 7: VHT20
at-wifi: wifi disconnected, rc:<reason_code>	Wi-Fi station disconnection reason code
at-wifi: wifi reconnect.(<index>/<total_count>)	Wi-Fi station is reconnecting to AP
at-net: unknown host: <hostname>	Unable to resolve hostname <hostname>, please check DNS settings and network connection
at-net: send failed, s:<send_len> r:<ret_len>	Failed to send data: expected to send <send_len> bytes, but only <ret_len> bytes were sent. Please check network connection.
at-net: link is changed	When AT is acting as a server, the client connection was interrupted during data transmission, and a new client has connected to the server.
at-net: link:<link_id> is on netif:<netif_id>	Connection <link_id> was successfully established on network interface <netif_id>. • 1: Wi-Fi station interface • 2: Wi-Fi softAP interface • 3: Ethernet interface

Chapter 4

AT Command Examples

4.1 AT Response Message Format Control Examples

- *Enable the system message filter to download files in HTTP passthrough mode*

4.1.1 Enable the system message filter to download files in HTTP passthrough mode

This example describes how to download an image file in PNG format. The image link is <https://www.espressif.com/sites/default/files/home/hardware.png>.

1. Restore your device to factory settings.

Command:

```
AT+RESTORE
```

Response:

```
OK
```

2. Set the Wi-Fi mode to station.

Command:

```
AT+CWMODE=1
```

Response:

```
OK
```

3. Connect to the router.

Command:

```
AT+CWJAP="espressif","1234567890"
```

Response:

```
WIFI CONNECTED  
WIFI GOT IP
```

```
OK
```

Note:

- The SSID and password you entered may be different from those in the above command. Please replace the SSID and password with those of your router settings.

4. Configure the first system message filter to filter out the header and tail from each HTTP data.

Command:

```
AT+SYMSGFILTERCFG=1,18,3
```

Response:

```
OK
```

```
>
```

Enter `^+HTTPCGET:[0-9]*`, (18 bytes in total) and `\r\n$` (3 bytes in total, the `\r\n` character corresponds to carriage return and line feed character, i.e. 0D 0A). Response:

```
OK
```

Note:

- `^+HTTPCGET:[0-9]*`, is the regular expression for headers. It means the string starts with `+HTTPCGET:`, followed by a string of numbers, and ends with a comma.
- `\r\n$` is the regular expression of tail. It means the string ends with `\r\n`.

5. Configure the second system message filter to filter out the OK system message returned after the picture is downloaded.

Command:

```
AT+SYMSGFILTERCFG=1,0,7
```

Response:

```
OK
```

```
>
```

At this time, enter `\r\nOK\r\n$` (7 bytes in total, where `\r\n` corresponds to line feed and carriage return in ASCII code, i.e. 0D 0A). Response:

```
OK
```

Note:

- `\r\nOK\r\n$` is the regular expression for tails. It means the string ends with `\r\nOK\r\n`.

6. Enable system message filter

Command:

```
AT+SYMSGFILTER=1
```

Response:

```
OK
```

Note:

- The filters set above will take effect only after system message filtering is enabled.

7. Turn off echo

Command:

```
ATE0
```

Response:

```
OK
```

8. Download the picture

Set the sending and receiving buffer size to 2048 bytes, and set the network timeout to 5000 ms. Note that the network timeout is not the command timeout. The 5-second network timeout means closing the network connection if no data is received from the server side for 5 consecutive seconds. In a poor network environment

where the server-side data is received every second, the network connection will not be closed, resulting in a long command timeout.

Command:

```
AT+HTTPGET="https://www.espressif.com/sites/default/files/home/hardware.png",  
→2048,2048,5000
```

Response:

```
// Here, the MCU transparently receives the entire picture resource from  
→https://www.espressif.com/sites/default/files/home/hardware.png.
```

Note:

- If the picture download fails, AT will still send a \r\nERROR\r\n (9 bytes in total) system message to MCU.

9. Clear the filter

Command:

```
AT+SYMSGFILTERCFG=0
```

Response:

```
OK
```

10. Disable the system message filter

Command:

```
AT+SYMSGFILTER=0
```

Response:

```
OK
```

11. Switch echo on

Command:

```
ATE1
```

Response:

```
OK
```

4.2 AT+SYSMFG Command Examples

This document provides detailed examples for using the [AT+SYSMFG](#) command to operate the *manufacturing nvs* user partition.

- [Introduction](#)
 - [Overview](#)
 - [Namespace Structure](#)
 - [Data Types](#)
 - [Other Important Notes](#)
- [Factory Parameter Configuration](#)
 - [Module Name Operations](#)
 - [UART Configuration Operations](#)
 - [Maximum TX Power Operations](#)
 - [Country Code Operations](#)
 - [System Store Mode Operations](#)
 - [Adding Custom Key-Value Pairs](#)

- [PKI Configuration](#)
 - [SSL Client Certificates](#)
 - [SSL Server Certificates](#)
- [GATTS Configuration](#)
- [Custom Data Storage](#)

4.2.1 Introduction

Overview

The manufacturing NVS partition contains multiple namespaces, each containing several key-value pairs to store different types of configuration information. Commonly used namespaces include:

Namespace	Description	Related AT Commands
factory_param	Factory parameters (module name, Wi-Fi configuration, UART configuration, etc.)	See Factory Parameter Configuration
client_cert	SSL client certificates	• AT+CIPSSLCCONF • AT+CIPSTART
client_key	SSL client keys	• AT+CIPSSLCCONF • AT+CIPSTART
client_ca	SSL client CA certificates	• AT+CIPSSLCCONF • AT+CIPSTART
server_cert	SSL server certificates	AT+CIPSERVER
server_key	SSL server keys	AT+CIPSERVER
server_ca	SSL server CA certificates	AT+CIPSERVER
mqtt_cert	MQTT client certificates	AT+MQTTUSERCFG
mqtt_key	MQTT client keys	AT+MQTTUSERCFG
mqtt_ca	MQTT client CA certificates	AT+MQTTUSERCFG
https_cert	HTTPS client certificates	AT+HTTPCFG
https_key	HTTPS client private keys	AT+HTTPCFG
https_ca	HTTPS client CA certificates	AT+HTTPCFG
wpa2_cert	WPA2-Enterprise client certificates	AT+CWJEAP
wpa2_key	WPA2-Enterprise client private keys	AT+CWJEAP
wpa2_ca	WPA2-Enterprise client CA certificates	AT+CWJEAP
wss_cert	WebSocket client certificates	AT+WSCFG
wss_key	WebSocket client private keys	AT+WSCFG
wss_ca	WebSocket client CA certificates	AT+WSCFG
ble_data	Bluetooth LE service configurations	AT+BLEGATTSSRVCRE

Query all namespaces actually supported by the current firmware:

Command:

```
AT+SYSMF?
```

Response:

```
+SYSMF:"factory_param"
+SYSMF:"client_cert"
+SYSMF:"client_key"
+SYSMF:"client_ca"
+SYSMF:"server_cert"
```

(continues on next page)

(continued from previous page)

```
+SYSMFG:"server_key"  
+SYSMFG:"server_ca"  
...  
+SYSMFG:"mqtt_cert"  
+SYSMFG:"mqtt_key"  
+SYSMFG:"mqtt_ca"  
  
OK
```

Note:

- The supported namespaces may vary depending on the chip and firmware version.
 - All command response examples in this document are for reference only. Actual output may differ by chip model, firmware version, and configuration.
-

Namespace Structure

Each namespace is a collection of key-value pairs. Different namespaces use different key naming conventions:

- **Certificate-related namespaces** (`client_cert`, `client_key`, `client_ca`, `server_cert`, `server_key`, `server_ca`): Use `<base_name>.<index>` format (e.g., `client_cert.0`, `client_cert.1`) to store multiple certificate sets.
- **factory_param namespace**: Uses descriptive key names (e.g., `module_name`, `uart_port`, `country_code`).
- **ble_data namespace**: Uses `cfg` with index format (e.g., `cfg0`, `cfg1`, `cfg2`).

Data Types

The `<type>` parameter in `AT+SYSMFG` commands indicates the data type:

- 1: `u8` (8-bit unsigned integer, range: 0-255, e.g., UART port number, channel number)
- 2: `i8` (8-bit signed integer, range: -128 to 127, for signed configuration values)
- 3: `u16` (16-bit unsigned integer, range: 0-65535, e.g., GPIO pin numbers)
- 4: `i16` (16-bit signed integer, range: -32768 to 32767)
- 5: `u32` (32-bit unsigned integer, range: 0-4294967295)
- 6: `i32` (32-bit signed integer, range: -2147483648 to 2147483647, e.g., UART baudrate like 115200)
- 7: `string` (text string, e.g., module name “MINI-1”, country code “CN”)
- 8: `binary` (binary data, e.g., certificates, private keys, BLE service configurations)

Other Important Notes

- **Length Parameter Explanation:** The meaning of the `<length>` parameter in responses varies by data type:
 - For numeric types (types 1-6): `<length>` represents the number of characters in the string representation of the value. For example, the `i32` baudrate value 115200 is displayed as 6 characters, so length is 6.
 - For string and binary types (types 7-8): `<length>` represents the actual number of bytes.
- **Binary Data Transmission:** When writing binary data (type 8), the command returns `>` and waits for you to send exactly the number of bytes specified in the `<value>` parameter.
- **Data Persistence:** Data written to manufacturing NVS persists across device reboots until explicitly erased.
- **Offset Reading:** You can read partial data from a key using the offset and length parameters, useful for large data blocks.
- **Compilation vs Runtime:** While some data (like BLE services) can be customized at compile time by modifying the source CSV files and regenerating `mfg_nvs.bin`, the `AT+SYSMFG` command allows you to update certain values at runtime without recompilation.

- **References:** For more information about Non-Volatile Storage (NVS) concepts and other NVS operations, please refer to the [ESP-IDF NVS documentation](#).

4.2.2 Factory Parameter Configuration

The `factory_param` namespace stores factory parameters, such as Wi-Fi configuration, UART configuration, and module information. ESP-AT uses these parameters during initialization. They affect the device's default behavior. For details, please refer to [factory parameter configuration](#).

Important: After you modify factory parameters, run `AT+RST` or `AT+RESTORE` to restart the device and apply changes.

Query All Parameters

Command:

```
AT+SYSMFG=1,"factory_param"
```

Response:

```
+SYSMFG:"factory_param","version",2
+SYSMFG:"factory_param","module_name",7
+SYSMFG:"factory_param","max_tx_power",2
+SYSMFG:"factory_param","country_code",7
+SYSMFG:"factory_param","start_channel",1
+SYSMFG:"factory_param","channel_num",1
+SYSMFG:"factory_param","uart_port",2
+SYSMFG:"factory_param","uart_baudrate",6
+SYSMFG:"factory_param","uart_tx_pin",6
+SYSMFG:"factory_param","uart_rx_pin",6
+SYSMFG:"factory_param","uart_cts_pin",6
+SYSMFG:"factory_param","uart_rts_pin",6
+SYSMFG:"factory_param","sys_store",1
```

```
OK
```

Module Name Operations

1. Read module name:

Command:

```
AT+SYSMFG=1,"factory_param","module_name"
```

Response:

```
+SYSMFG:"factory_param","module_name",7,6,MINI-1
```

```
OK
```

2. Write a new module name:

Command:

```
AT+SYSMFG=2,"factory_param","module_name",7,<name_length>
```

Response:

```
OK
```

```
>
```

After > is returned, send the module name string. For example, send “MY-MODULE-01” .

Response after data transmission:

```
OK
```

3. Verify the configuration:

Command:

```
AT+SYSMF=1, "factory_param", "module_name"
```

Response:

```
+SYSMF: "factory_param", "module_name", 7, 12, MY-MODULE-01
```

```
OK
```

The module name has been successfully updated to “MY-MODULE-01” .

UART Configuration Operations

1. To read UART port:

Command:

```
AT+SYSMF=1, "factory_param", "uart_port"
```

Response:

```
+SYSMF: "factory_param", "uart_port", 2, 1, 1
```

```
OK
```

2. To read UART baudrate:

Command:

```
AT+SYSMF=1, "factory_param", "uart_baudrate"
```

Response:

```
+SYSMF: "factory_param", "uart_baudrate", 6, 6, 115200
```

```
OK
```

3. Update UART TX pin:

Command:

```
AT+SYSMF=2, "factory_param", "uart_tx_pin", 6, 17
```

Response:

```
OK
```

Note: After you restart, switch your hardware connection to the new pins or send AT commands using the new UART configuration to verify the change.

Maximum TX Power Operations

1. Read maximum TX power:

Command:

```
AT+SYSMF=1,"factory_param","max_tx_power"
```

Response:

```
+SYSMF:"factory_param","max_tx_power",2,2,78  
OK
```

2. Update maximum TX power:

Command:

```
AT+SYSMF=2,"factory_param","max_tx_power",2,60
```

Response:

```
OK
```

For the valid Wi-Fi maximum TX power values, please refer to the `<wifi_power>` parameter range in [AT+RFPOWER](#).

3. Restart the device to apply changes:

Command:

```
AT+RST
```

Response:

```
ready
```

4. Verify the configuration:

Command:

```
AT+SYSMF=1,"factory_param","max_tx_power"
```

Response:

```
+SYSMF:"factory_param","max_tx_power",2,2,60  
OK
```

The maximum TX power has been successfully updated.

5. Verify using AT+RFPOWER command:

Command:

```
AT+RFPOWER?
```

Response:


```
+RFPOWER:60  
OK
```

The maximum TX power is now 60 ($60 \times 0.25 \text{ dBm} = 15 \text{ dBm}$).

Country Code Operations

1. Read country code:

Command:

```
AT+SYSMF=1,"factory_param","country_code"
```

Response:

```
+SYSMF:"factory_param","country_code",7,2,CN  
OK
```

2. Update country code:

Command:

```
AT+SYSMF=2,"factory_param","country_code",7,2
```

Response:

```
OK  
>
```

After > is returned, send the country code (e.g., “US” , “JP” , “EU”).

Response after data transmission:

```
OK
```

3. Restart the device to apply changes:

Command:

```
AT+RST
```

Response:

```
ready
```

4. Verify the configuration:

Command:

```
AT+SYSMF=1,"factory_param","country_code"
```

Response:

```
+SYSMF:"factory_param","country_code",7,2,US  
OK
```

The country code has been successfully updated to “US” .

5. Update channel number to 11 (US standard):

Command:

```
AT+SYSMF=2, "factory_param", "channel_num", 1, 11
```

Response:

```
OK
```

- Restart the device to apply changes:

Command:

```
AT+RST
```

Response:

```
ready
```

- Verify country code and channel configuration using AT+CWCOUNTRY command:

Command:

```
AT+CWCOUNTRY?
```

Response:

```
+CWCOUNTRY:0, "US", 1, 11
```

```
OK
```

Response description:

- 0: Country code policy (0 means using default country code configuration)
- "US": Country code
- 1: Starting channel (start_channel)
- 11: Number of channels (channel_num)

The configuration has been successfully applied. The device now uses the US country code and channel configuration.

System Store Mode Operations

The `sys_store` parameter controls whether the system automatically stores configuration changes to flash. This parameter is related to the [AT+SYSSTORE](#) command functionality.

- Read the current system store mode:

Command:

```
AT+SYSMF=1, "factory_param", "sys_store"
```

Response:

```
+SYSMF:"factory_param", "sys_store", 1, 1, 1
```

```
OK
```

`sys_store` values:

- 0: Do not automatically store configurations to flash (equivalent to AT+SYSSTORE=0)
- 1: Automatically store configurations to flash (equivalent to AT+SYSSTORE=1, default value)

- Change system store mode to disable automatic storage:

Command:

```
AT+SYSMF=2, "factory_param", "sys_store", 1, 0
```

Response:

```
OK
```

- Restart the device to apply changes:

Command:

```
AT+RST
```

Response:

```
ready
```

- Verify the configuration:

Command:

```
AT+SYSMF=1, "factory_param", "sys_store"
```

Response:

```
+SYSMF: "factory_param", "sys_store", 1, 1, 0  
OK
```

- Verify using AT+SYSSTORE command:

Command:

```
AT+SYSSTORE?
```

Response:

```
+SYSSTORE: 0  
OK
```

The system store mode has been successfully changed to not automatically store (0 means auto-storage is disabled).

Impact of the change:

When `sys_store` is set to 0, configuration changes (such as Wi-Fi settings or other commands that support auto-storage) are not saved to flash automatically. After a restart, the device will use the previously saved settings. For details on which commands are affected, see [AT+SYSSTORE](#).

If you need to save specific configurations while in `sys_store=0` mode, you can use the [AT+SYSSTORE](#) command to temporarily enable storage, execute the configuration command, and then disable it.

Adding Custom Key-Value Pairs

In addition to predefined factory parameters, you can add custom key-value pairs to the `factory_param` namespace to store application-specific configuration data.

This example demonstrates how to add a custom key-value pair, such as storing a device serial number:

- Add a custom string-type key-value pair:

Command:

```
AT+SYSMFG=2,"factory_param","device_sn",7,16
```

Response:

```
OK  
>
```

After > is returned, send the device serial number (e.g., “SN202601050001”).

Response after data transmission:

```
OK
```

2. Read the custom key-value pair:

Command:

```
AT+SYSMFG=1,"factory_param","device_sn"
```

Response:

```
+SYSMFG:"factory_param","device_sn",7,16,SN202601050001  
OK
```

3. Add a numeric-type custom key-value pair:

Command:

```
AT+SYSMFG=2,"factory_param","hw_version",6,1001
```

Response:

```
OK
```

This stores the hardware version number 1001.

4. Query all key-value pairs (including custom ones):

Command:

```
AT+SYSMFG=1,"factory_param"
```

Response:

```
+SYSMFG:"factory_param","version",2  
+SYSMFG:"factory_param","module_name",7  
...  
+SYSMFG:"factory_param","device_sn",7  
+SYSMFG:"factory_param","hw_version",6  
OK
```

5. Delete the custom key-value pair:

Command:

```
AT+SYSMFG=0,"factory_param","device_sn"
```

Response:

```
OK
```

Note:

- Custom key names should avoid conflicts with predefined key names (such as `module_name`, `uart_port`, etc.)
- Use meaningful key names for easier maintenance
- Custom key-value pairs persist across reboots
- Too many key-value pairs may affect performance; it's recommended to store only necessary configuration information
- After modifying values in `factory_param`, it's recommended to restart the device to ensure the application correctly reads the new values

4.2.3 PKI Configuration

PKI (Public Key Infrastructure) configuration is used to create, manage, distribute, store, and revoke digital certificates, as well as manage public key cryptography. ESP-AT supports configuring certificates for various functions, including SSL client/server, MQTT client, HTTPS client, and more. The configuration method for all certificates is identical, implemented through the `AT+SYSMFG` command to operate the corresponding namespaces. For more details, please refer to [PKI Configuration Introduction](#).

Namespace mapping for different features:

Feature	Certificate/Key Namespaces
SSL Client	<code>client_cert</code> , <code>client_key</code> , <code>client_ca</code>
SSL Server	<code>server_cert</code> , <code>server_key</code> , <code>server_ca</code>
MQTT Client	<code>mqtt_cert</code> , <code>mqtt_key</code> , <code>mqtt_ca</code>
HTTPS Client	<code>https_cert</code> , <code>https_key</code> , <code>https_ca</code>
WebSocket Client	<code>wss_cert</code> , <code>wss_key</code> , <code>wss_ca</code>
WPA2 Enterprise Client	<code>wpa2_cert</code> , <code>wpa2_key</code> , <code>wpa2_ca</code>

Note: You can query all namespaces supported by the current firmware using the `AT+SYSMFG?` command to confirm if the certificate namespaces you need are available.

This section uses SSL certificates as an example to illustrate the complete process of certificate configuration. Certificate configuration for other functions (such as MQTT, HTTPS) can follow the same method, only requiring modification of the corresponding namespace names according to the table above.

SSL Client Certificates

The `client_cert`, `client_key`, and `client_ca` namespaces store SSL certificates and keys used for secure connections. These three namespaces can work together to enable SSL authentication (one-way or mutual authentication).

Query Certificate Information

1. To read all keys in the client certificate namespace:

Command:

```
AT+SYSMFG=1,"client_cert"
```

Response:

```
+SYSMFG:"client_cert","client_cert.0",8
+SYSMFG:"client_cert","client_cert.1",8

OK
```

Similarly, you can query `client_key` and `client_ca` namespaces:

```
AT+SYSMFG=1,"client_key"
AT+SYSMFG=1,"client_ca"
```

2. Read a specific certificate or key:

Command:

```
AT+SYSMFG=1,"client_cert","client_cert.0"
```

Response:

```
+SYSMFG:"client_cert","client_cert.0",8,1164,<value>

OK
```

Configure Certificates

1. Write client certificate:

Command:

```
AT+SYSMFG=2,"client_cert","client_cert.0",8,1164
```

Response:

```
OK

>
```

After `>` is returned, send the certificate data (1164 bytes). After the data is sent, the system returns:

```
OK
```

2. Write client key:

Command:

```
AT+SYSMFG=2,"client_key","client_key.0",8,<key_length>
```

Response:

```
OK

>
```

After `>` is returned, send the key data. After the data is sent, the system returns:

```
OK
```

3. Write CA certificate:

Command:

```
AT+SYSMFG=2,"client_ca","client_ca.0",8,<ca_length>
```

Response:

```
OK
```

```
>
```

After > is returned, send the CA certificate data. After the data is sent, the system returns:

```
OK
```

Use Certificates After writing client certificates, keys (and client CA certificate for mutual authentication) to the manufacturing NVS partition, you can use them to establish a secure SSL connection. Here's an example:

1. Ensure certificates are written to the manufacturing partition:
 - Client certificate in `client_cert` namespace
 - Client key in `client_key` namespace
 - Client CA certificate in `client_ca` namespace
2. Configure SSL connection to use manufacturing partition certificates:

Command:

```
AT+CIPSSLCONF=3,0,0
```

Response:

```
OK
```

This configures the SSL client to use certificates from the manufacturing partition (Index 0).

3. Connect to Wi-Fi:

Command:

```
AT+CWJAP="your_ssid","your_password"
```

Response:

```
OK
```

4. Establish SSL connection to a server:

Command:

```
AT+CIPSTART="SSL","example.com",8883
```

Response:

```
CONNECT
```

```
OK
```

The SSL connection will use the certificates from the manufacturing partition for authentication.

Note: For more details about SSL configuration, please refer to the [AT+CIPSSLCONF](#) command documentation.

Erase Certificates Erase a specific certificate:

Command:

```
AT+SYSMF0="client_cert","client_cert.0"
```

Response:

```
OK
```

Erase all keys in a namespace:

Command:

```
AT+SYSMFG=0,"client_cert"
```

Response:

```
OK
```

You can use the same commands to erase client_key and client_ca data.

SSL Server Certificates

The server_cert, server_key, and server_ca namespaces store the SSL certificates and keys required for the device to act as an SSL server. These three namespaces can work together to enable secure server-side SSL connections (one-way or mutual authentication).

Query Certificate Information

1. To read all keys in the server certificate namespace:

Command:

```
AT+SYSMFG=1,"server_cert"
```

Response:

```
+SYSMFG:"server_cert","server_cert.0",8  
+SYSMFG:"server_cert","server_cert.1",8
```

```
OK
```

Similarly, you can query server_key and server_ca namespaces:

```
AT+SYSMFG=1,"server_key"  
AT+SYSMFG=1,"server_ca"
```

2. Read a specific server certificate or key:

Command:

```
AT+SYSMFG=1,"server_cert","server_cert.0"
```

Response:

```
+SYSMFG:"server_cert","server_cert.0",8,<length>,<value>
```

```
OK
```

Command:

```
AT+SYSMFG=1,"server_key","server_key.0"
```

Response:

```
+SYSMFG:"server_key","server_key.0",8,<length>,<value>
```

```
OK
```

Configure Certificates

1. Write server certificate:

Command:

```
AT+SYSMFG=2,"server_cert","server_cert.0",8,<cert_length>
```

Response:

```
OK
```

```
>
```

After > is returned, send the certificate data.

Response after data transmission:

```
OK
```

2. Write server key:

Command:

```
AT+SYSMFG=2,"server_key","server_key.0",8,<key_length>
```

Response:

```
OK
```

```
>
```

After > is returned, send the key data.

Response after data transmission:

```
OK
```

3. Write CA certificate:

Command:

```
AT+SYSMFG=2,"server_ca","server_ca.0",8,<ca_length>
```

Response:

```
OK
```

```
>
```

After > is returned, send the CA certificate data.

Response after data transmission:

```
OK
```

Use Certificates After writing the server certificate, key (and server CA certificate for mutual authentication) to the manufacturing NVS partition, you can create an SSL server based on these certificates. Here is a complete example:

1. Ensure server credentials are written to the manufacturing partition:
 - Server certificate in `server_cert` namespace
 - Server key in `server_key` namespace
 - Server CA certificate in `server_ca` namespace (optional, only needed if client authentication is required)
2. Enable multiple connections mode:

Command:

```
AT+CIPMUX=1
```

Response:

```
OK
```

3. Start the SSL server:

- For one-way authentication (server provides certificate to client):

Command:

```
AT+CIPSERVER=1,8883,"SSL"
```

Response:

```
OK
```

- For mutual authentication (server and client verify each other's certificates):

Command:

```
AT+CIPSERVER=1,8883,"SSL",1
```

Response:

```
OK
```

The last parameter 1 enables CA verification, and the server will verify client certificates using the CA certificate from the manufacturing partition.

The device is now listening for SSL connections on port 8883. The SSL server will automatically use the server certificate and key from the manufacturing partition.

- ### 4. When a client connects, the SSL handshake will use the server certificate and key from the manufacturing partition for authentication.

Note:

- The SSL server automatically reads certificates and keys from the `server_cert` and `server_key` namespaces in the manufacturing partition without additional configuration.
 - If CA verification is enabled in the [AT+CIPSERVER](#) command (`<CA enable> = 1`), the server will also read the CA certificate from the `server_ca` namespace to verify client certificates, enabling mutual authentication.
 - For more details about SSL server configuration, please refer to the [AT+CIPSERVER](#) command documentation.
-

Erase Certificates Erase a specific server certificate:

Command:

```
AT+SYSMF=0,"server_cert","server_cert.0"
```

Response:

```
OK
```

Erase all keys in a namespace:

Command:

```
AT+SYSMF=0,"server_cert"
```

Response:

```
OK
```

You can use the same commands to erase `server_key` and `server_ca` data.

4.2.4 GATTS Configuration

The `ble_data` namespace stores Bluetooth Low Energy GATTS (Generic Attribute Profile Server) service configurations. This namespace contains individual GATT service configuration items organized as `cfg0`, `cfg1`, `cfg2`, etc., where each configuration item represents one attribute (service, characteristic, or descriptor) in the GATT table. For more details, please refer to [Bluetooth LE service source file introduction](#).

Important:

- When modifying the `ble_data` namespace with the `AT+SYSMFG` command, it must be executed before BLE is initialized. If BLE is already initialized, run `AT+BLEINIT=0` to disable BLE first.
- Before using `AT+SYSMFG` to read or write the `ble_data` namespace, please read the [Bluetooth LE service source file introduction](#) document to understand the structure and format of Bluetooth LE service configurations.

Query Configuration Information

Read Configuration Items

1. To read all configuration items in the `ble_data` namespace:

Command:

```
AT+SYSMFG=1, "ble_data"
```

Response:

```
+SYSMFG: "ble_data", "cfg0", 7
+SYSMFG: "ble_data", "cfg1", 7
+SYSMFG: "ble_data", "cfg2", 7
+SYSMFG: "ble_data", "cfg3", 7
...
+SYSMFG: "ble_data", "cfg30", 7

OK
```

2. Read a specific BLE configuration item (e.g., `cfg2` represents one row in the GATT service table):

Command:

```
AT+SYSMFG=1, "ble_data", "cfg2"
```

Response:

```
+SYSMFG: "ble_data", "cfg2", 7, 23, 2, 16, 0xC300, 0x01, 1, 1, 30

OK
```

Response Format:

The response data represents the following fields from the GATT table:

- 2: index
- 16: uuid_len (UUID length in bits)
- 0xC300: uuid
- 0x01: perm (permission)
- 1: val_max_len (maximum value length)
- 1: val_cur_len (current value length)
- 30: value

Configure Services

Modify BLE Characteristic UUID This example shows how to modify a characteristic's UUID from 0xC300 to 0xC500 at runtime.

1. Query the current configuration:

Command:

```
AT+SYSMFG=1,"ble_data","cfg2"
```

Response:

```
+SYSMFG:"ble_data","cfg2",7,23,2,16,0xC300,0x01,1,1,30
OK
```

The current UUID is 0xC300.

2. Write the new configuration with UUID 0xC500:

Command:

```
AT+SYSMFG=2,"ble_data","cfg2",7,23
```

Response:

```
OK
>
```

Send the data:

```
2,16,0xC500,0x01,1,1,30
```

Response:

```
OK
```

3. Query the configuration item to verify the UUID modification:

Command:

```
AT+SYSMFG=1,"ble_data","cfg2"
```

Response:

```
+SYSMFG:"ble_data","cfg2",7,23,2,16,0xC500,0x01,1,1,30
OK
```

The UUID has been successfully changed to 0xC500.

4. Re-initialize Bluetooth LE and create services to apply changes:

Command:

```
AT+BLEINIT=2
```

Response:

```
OK
```

Command:

```
AT+BLEGATTSSRVCRE
```

Response:

```
OK
```

Command:

```
AT+BLEGATTSSRVSTART
```

Response:

```
OK
```

5. Query services to verify:

Command:

```
AT+BLEGATTSSRV?
```

Response:

```
+BLEGATTSSRV:1,1,0xA002,1
+BLEGATTSSRV:1,2,0xA003,1
OK
```

6. Query characteristics to verify the UUID change:

Command:

```
AT+BLEGATTSSCHAR?
```

Response:

```
+BLEGATTSSCHAR:"char",1,1,0xC500
+BLEGATTSSCHAR:"char",1,2,0xC301
+BLEGATTSSCHAR:"char",1,3,0xC302
+BLEGATTSSCHAR:"char",1,4,0xC303
+BLEGATTSSCHAR:"char",1,5,0xC304
+BLEGATTSSCHAR:"char",1,6,0xC305
+BLEGATTSSCHAR:"char",1,7,0xC306
+BLEGATTSSCHAR:"char",1,8,0xC307
+BLEGATTSSCHAR:"char",2,1,0xC400
+BLEGATTSSCHAR:"char",2,2,0xC401
OK
```

The first characteristic now shows UUID 0xC500 instead of 0xC300.

Add New Custom BLE Service This example demonstrates how to add a series of new BLE services at runtime without modifying existing service configurations. Let's assume we want to add a custom service (UUID: 0xA100) with one readable and writable characteristic (UUID: 0xB100).

1. Query the current last configuration item index:

Command:

```
AT+SYSMFG=1,"ble_data"
```

Response:

```
+SYSMFG:"ble_data","cfg0",7
+SYSMFG:"ble_data","cfg1",7
...
+SYSMFG:"ble_data","cfg30",7
OK
```

Assuming the current last configuration item is `cfg30`, we will start adding new services from `cfg31`.

2. Add Primary Service Declaration:

Command:

```
AT+SYSMFG=2,"ble_data","cfg31",7,26
```

Response:

```
OK
```

```
>
```

Send data:

```
31,16,0x2800,0x01,2,2,A100
```

Data format explanation:

- 31: Configuration item index
- 16: UUID length (16 bits)
- 0x2800: Primary Service Declaration UUID
- 0x01: Permission (readable)
- 2: Maximum value length (2 bytes for service UUID)
- 2: Current value length
- A100: Custom service UUID

Response:

```
OK
```

3. Add Characteristic Declaration:

Command:

```
AT+SYSMFG=2,"ble_data","cfg32",7,24
```

Response:

```
OK
```

```
>
```

Send data:

```
32,16,0x2803,0x01,3,1,02
```

Data format explanation:

- 32: Configuration item index
- 16: UUID length
- 0x2803: Characteristic Declaration
- 0x01: Permission (readable)
- 3: Maximum value length
- 1: Current value length
- 02: Set permission to read

Response:

```
OK
```

4. Add Characteristic Value:

Command:

```
AT+SYSMFG=2,"ble_data","cfg33",7,31
```

Response:

```
OK
```

```
>
```

Send data:

```
33,16,0xB100,0x11,20,8,12345678
```

Data format explanation:

- 33: Configuration item index
- 16: UUID length
- 0xB100: Characteristic value UUID (matches the UUID in characteristic declaration)

- 0x11: Permission (read + write: ESP_GATT_PERM_READ (0x01) + ESP_GATT_PERM_WRITE (0x10) = 0x11)
- 20: Maximum value length (allows up to 20 bytes of data)
- 8: Current value length (initial value is 8 bytes)
- 12345678: Initial value (8 bytes of data)

Response:

```
OK
```

5. Query the newly added configuration items to verify:

Command:

```
AT+SYSMFG=1,"ble_data","cfg31"
```

Response:

```
+SYSMFG:"ble_data","cfg31",7,26,31,16,0x2800,0x01,2,2,A100
```

```
OK
```

Command:

```
AT+SYSMFG=1,"ble_data","cfg32"
```

Response:

```
+SYSMFG:"ble_data","cfg32",7,24,32,16,0x2803,0x01,3,1,02
```

```
OK
```

Command:

```
AT+SYSMFG=1,"ble_data","cfg33"
```

Response:

```
+SYSMFG:"ble_data","cfg33",7,31,33,16,0xB100,0x11,20,8,12345678
```

```
OK
```

6. Re-initialize Bluetooth LE and create services to apply changes:

Command:

```
AT+BLEINIT=2
```

Response:

```
OK
```

Command:

```
AT+BLEGATTSSRVCRE
```

Response:

```
OK
```

Command:

```
AT+BLEGATTSSRVSTART
```

Response:

```
OK
```

7. Query services to verify the new service has been added:

Command:

```
AT+BLEGATTSSRV?
```

Response:

```
+BLEGATTSSRV:1,1,0xA002,1
+BLEGATTSSRV:1,2,0xA003,1
+BLEGATTSSRV:1,3,0xA100,1
```

```
OK
```

The newly added service UUID 0xA100 now appears in the service list.

8. Query characteristics to verify the new characteristic has been added:

Command:

```
AT+BLEGATTSSCHAR?
```

Response:

```
+BLEGATTSSCHAR:"char",1,1,0xC300
+BLEGATTSSCHAR:"char",1,2,0xC301
...
+BLEGATTSSCHAR:"char",3,1,0xB100
```

```
OK
```

The newly added characteristic UUID 0xB100 now appears in the characteristic list (belongs to service 3, characteristic index 1).

Note: BLE Service Configuration Points:

- When adding new services, ensure the `cfg` index used does not conflict with existing configurations
- A complete BLE service requires at least three configuration items: primary service declaration, characteristic declaration, and characteristic value
- If characteristic descriptors are needed (such as Client Characteristic Configuration Descriptor), add additional configuration items
- Characteristic property values can be combined. Common properties include: 0x01 (broadcast), 0x02 (read), 0x04 (write without response), 0x08 (write), 0x10 (notify), 0x20 (indicate), 0x40 (authenticated signed writes), 0x80 (extended properties)

Data Format and Activation:

- Each `cfg` item corresponds to one row in the GATT service table defined in the `gatts_data.csv` file
- Data format: `index,uuid_len,uuid,perm,val_max_len,val_cur_len,value`
- Configurations written via `AT+SYSMFG` take effect after re-initializing BLE (`AT+BLEINIT`) and recreating services (`AT+BLEGATTSSRVCRE`)
- For more details on GATT table structure, see [How to Customize Bluetooth® LE Services](#)

4.2.5 Custom Data Storage

In addition to the predefined namespaces (such as `factory_param`, `client_cert`, `server_cert`, etc.), you can create custom namespaces to store your own application data. This is useful for storing small amounts of persistent data that need to survive reboots, such as custom configuration, user preferences, or device-specific settings.

Create and Use Custom Namespace

This example demonstrates how to create a custom namespace called `my_app_ca` to store application-specific certificates or configuration data.

1. Create a new namespace and write data:

Command:

```
AT+SYSMFG=2, "my_app_ca", "my_app_ca.0", 8, 100
```

Response:

```
OK
```

```
>
```

After > is returned, send your binary data (100 bytes). After the data is sent, the system returns:

```
OK
```

The namespace `my_app_ca` is now created with a key `my_app_ca.0` containing your data.

2. Query the custom namespace:

Command:

```
AT+SYSMFG=1, "my_app_ca"
```

Response:

```
+SYSMFG: "my_app_ca", "my_app_ca.0", 8
```

```
OK
```

3. Read the data from your custom namespace:

Command:

```
AT+SYSMFG=1, "my_app_ca", "my_app_ca.0"
```

Response:

```
+SYSMFG: "my_app_ca", "my_app_ca.0", 8, 100, <data>
```

```
OK
```

The response includes the data type (8 for binary), length (100 bytes), and the actual data.

4. Add more keys to the same namespace:

Command:

```
AT+SYSMFG=2, "my_app_ca", "my_app_ca.1", 7, 20
```

Response:

```
OK
```

```
>
```

Send data:

```
MyCustomConfigValue1
```

Response:

```
OK
```

5. Query all keys in the namespace:

Command:

```
AT+SYSMFG=1, "my_app_ca"
```

Response:

```
+SYSMFG: "my_app_ca", "my_app_ca.0", 8
+SYSMFG: "my_app_ca", "my_app_ca.1", 7

OK
```

6. Create another namespace for application configuration parameters:

If you need to store configuration parameters (rather than certificate data), it's recommended to create a separate namespace to maintain semantic clarity.

Command:

```
AT+SYSMFG=2, "my_app_config", "timeout", 6, 5000
```

Response:

```
OK
```

This stores an i32 configuration value (type 6) in the `my_app_config` namespace with the key name "timeout" and value 5000.

7. Read the configuration parameter:

Command:

```
AT+SYSMFG=1, "my_app_config", "timeout"
```

Response:

```
+SYSMFG: "my_app_config", "timeout", 6, 4, 5000

OK
```

8. Erase a specific key:

Command:

```
AT+SYSMFG=0, "my_app_ca", "my_app_ca.1"
```

Response:

```
OK
```

9. Erase the entire namespace:

Command:

```
AT+SYSMFG=0, "my_app_ca"
```

Response:

```
OK
```

Note: Custom Namespace Best Practices:

- Custom namespaces can be used to store any type of persistent data
- Choose the appropriate data type based on your needs: numeric types (1-6) for numbers, string (7) for text, binary (8) for certificates or binary data

- **Separate namespaces by purpose:** For example, store certificate data (`my_app_ca`) separately from configuration parameters (`my_app_config`) for easier management and maintenance
- Namespace names should be descriptive and clearly indicate their purpose (e.g., `my_app_certs`, `my_app_settings`)
- Key names can be any string, but it's recommended to use meaningful names for easier identification
- All custom data persists across reboots until explicitly erased
- Custom namespaces are independent of system namespaces and do not affect AT firmware functionality
- After creating a custom namespace and storing data, you can query it using `AT+SYSMEG?` to see it in the namespace list

4.3 TCP/IP AT Examples

This document provides detailed command examples to illustrate how to utilize [TCP/IP AT Commands](#) on ESP32.

- *ESP32 as a TCP client in single connection*
- *ESP32 as a TCP server in multiple connections*
- *UDP transmission with fixed remote IP address and port*
- *UDP transmission with changeable remote IP address and port*
- *ESP32 as an SSL client in single connection*
- *ESP32 as an SSL server in multiple connections*
- *ESP32 as an SSL client to create a single connection with mutual authentication*
- *ESP32 as an SSL server to create multiple connections with mutual authentication*
- *Network passthrough transmission when the ESP32 works as a TCP client in single connection*
- *Network passthrough transmission when the ESP32 works as a TCP server*
- *Network passthrough transmission when the ESP32 works as a softAP in UDP transparent transmission*
- *ESP32 obtains socket data in passive receiving mode*
- *ESP32 enables mDNS service, PC connects to the device's TCP server using a domain name*

4.3.1 ESP32 as a TCP client in single connection

1. Set the Wi-Fi mode to station.

Command:

```
AT+CWMODE=1
```

Response:

```
OK
```

2. Connect to the router.

Command:

```
AT+CWJAP="espressif", "1234567890"
```

Response:

```
WIFI CONNECTED
WIFI GOT IP
OK
```

Note:

- The SSID and password you entered may be different from those in the above command. Please replace the SSID and password with those of your router settings.

- Query the device's IP address.

Command:

```
AT+CIPSTA?
```

Response:

```
+CIPSTA:ip:"192.168.3.112"
+CIPSTA:gateway:"192.168.3.1"
+CIPSTA:netmask:"255.255.255.0"

OK
```

Note:

- The query results you obtained may be different from those in the above response.

- Connect the PC to the same router which ESP32 is connected to.

Use a network tool on the PC to create a TCP server. For example, the TCP server on PC is 192.168.3.102, and the port is 8080.

- Connect ESP32 to the TCP server as a client over TCP. The server's IP address is 192.168.3.102, and the port is 8080.

Command:

```
AT+CIPSTART="TCP", "192.168.3.102", 8080
```

Response:

```
CONNECT

OK
```

- Send 4 bytes of data.

Command:

```
AT+CIPSEND=4
```

Response:

```
OK

>
```

Input 4 bytes, for example, test, then AT will respond the following message.

```
Recv 4 bytes

SEND OK
```

Note:

- If the number of bytes inputted are more than the length (n) set by AT+CIPSEND, the system will reply busy p . . . , and send the first n bytes. And after sending the first n bytes, the system will reply SEND OK.

- Receive 4 bytes of data.

Assume that the TCP server sends 4 bytes of data (data is test), the system will prompt:

```
+IPD,4:test
```

4.3.2 ESP32 as a TCP server in multiple connections

When ESP32 works as a TCP server, multiple connections should be enabled by [AT+CIPMUX=1](#) command, because in most cases more than one client needs to be connected to the ESP32 server.

Below is an example showing how a TCP server is established when ESP32 works in the softAP mode. If ESP32 works as a station, you can set up a server in the same way mentioned above after connecting ESP32 to the router.

1. Set the Wi-Fi mode to softAP.

Command:

```
AT+CWMODE=2
```

Response:

```
OK
```

2. Enable multiple connections.

Command:

```
AT+CIPMUX=1
```

Response:

```
OK
```

3. Set softAP.

Command:

```
AT+CWSAP="ESP32_softAP","1234567890",5,3
```

Response:

```
OK
```

4. Query softAP information.

Command:

```
AT+CIPAP?
```

Response:

```
AT+CIPAP?
+CIPAP:ip:"192.168.4.1"
+CIPAP:gateway:"192.168.4.1"
+CIPAP:netmask:"255.255.255.0"

OK
```

Note:

- The address you obtained may be different from that in the above response.

5. Set up a TCP server, the default port is 333.

Command:

```
AT+CIPSERVER=1
```

Response:

```
OK
```

6. Connect the PC to the ESP32 softAP.

7. Use a network tool on PC to create a TCP client and connect it to the TCP server that ESP32 has created.

8. Send 4 bytes of data to connection link 0.

Command:

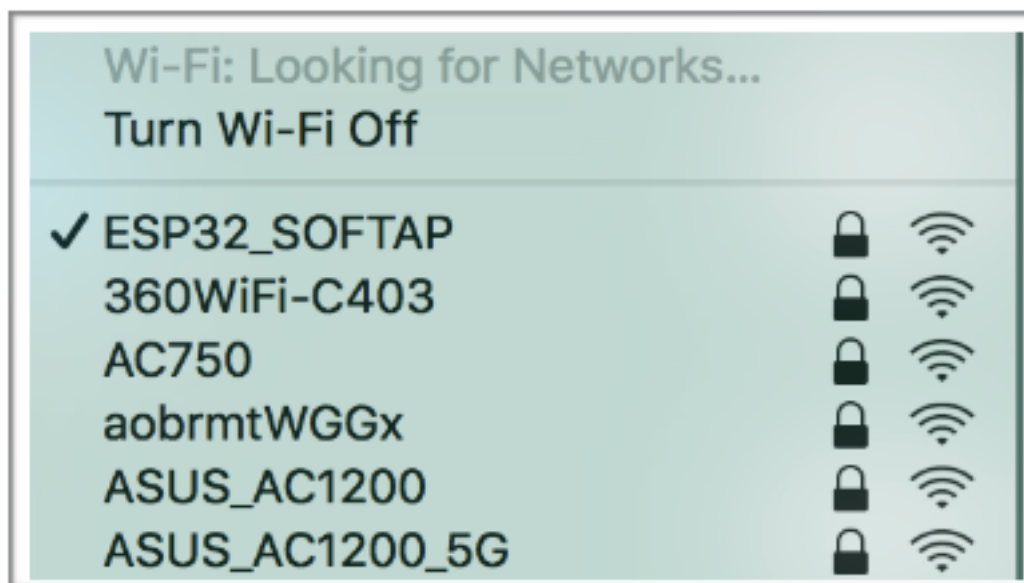
```
AT+CIPSEND=0,4
```

Response:

```
OK
```

```
>
```

Input 4 bytes, for example, `test`, then AT will respond the following messages.



```
Recv 4 bytes
```

```
SEND OK
```

Note:

- If the number of bytes inputted are more than the length (n) set by AT+CIPSEND, the system will reply busy p . . . , and send the first n bytes. And after sending the first n bytes, the system will reply SEND OK.
9. Receive 4 bytes of data from connection link 0.
Assume that the TCP server sends 4 bytes of data (data is test), the system will prompt:

```
+IPD,0,4:test
```

10. Close TCP connection.

Command:

```
AT+CIPCLOSE=0
```

Response:

```
0,CLOSED
```

```
OK
```

4.3.3 UDP transmission with fixed remote IP address and port

1. Set the Wi-Fi mode to station.

Command:

```
AT+CWMODE=1
```

Response:

```
OK
```

2. Connect to the router.

Command:

```
AT+CWJAP="espressif","1234567890"
```

Response:

```
WIFI CONNECTED
WIFI GOT IP

OK
```

Note:

- The SSID and password you entered may be different from those in the above command. Please replace the SSID and password with those of your router settings.

3. Query the device's IP address.

Command:

```
AT+CIPSTA?
```

Response:

```
+CIPSTA:ip:"192.168.3.112"
+CIPSTA:gateway:"192.168.3.1"
+CIPSTA:netmask:"255.255.255.0"

OK
```

Note:

- The query results you obtained may be different from those in the above response.

4. Connect the PC to the same router which ESP32 is connected to.

Use a network tool on the PC to create UDP transmission. For example, the PC's IP address is 192.168.3.102, and the port is 8080.

5. Enable multiple connections.

Command:

```
AT+CIPMUX=1
```

Response:

```
OK
```

6. Create a UDP transmission. The connection link is 4, the remote host's IP address is 192.168.3.102, the remote port is 8080, the local port is 1112, and the mode is 0.

Important: In UDP transmission, whether the remote IP address and port are fixed or not is determined by the `mode` parameter of `AT+CIPSTART`. If the parameter is 0, a specific connection link ID will be given to ensure that the remote IP address and port are fixed and the data sender and receiver will not be replaced by other devices.

Command:

```
AT+CIPSTART=4,"UDP","192.168.3.102",8080,1112,0
```

Response:

```
4,CONNECT

OK
```

Note:

- "192.168.3.102" and 8080 are the remote IP address and port of UDP transmission on the remote side, i.e., the UDP configuration set by PC.
- 1112 is the local port number of ESP32. You can define this port number, or else, a random port will be used.
- 0 means that the remote IP address and port are fixed and cannot be changed. For example, when there is another PC creating a UDP entity and sending data to ESP32 port 1112, ESP32 will still receive the data from UDP port 1112, and if the AT command `AT+CIPSEND=4,X` is used, the data will still be sent to the first PC end. However, if the parameter is not set as 0, the data will be sent to the new PC.

7. Send 7 bytes of data to connection link 4.

Command:

```
AT+CIPSEND=4,7
```

Response:

```
OK  
  
>
```

Input 7 bytes, for example, `abcdefg`, then AT will respond the following messages.

```
Recv 7 bytes  
  
SEND OK
```

Note:

- If the number of bytes inputted are more than the length (n) set by `AT+CIPSEND`, the system will reply `busy p . . .`, and send the first n bytes. And after sending the first n bytes, the system will reply `SEND OK`.
8. Receive 4 bytes of data from connection link 4.
Assume that the PC sends 4 bytes of data (data is `test`), the system will prompt:

```
+IPD,4,4:test
```

9. Close UDP connection link 4.

Command:

```
AT+CIPCLOSE=4
```

Response:

```
4,CLOSED  
  
OK
```

4.3.4 UDP transmission with changeable remote IP address and port

1. Set the Wi-Fi mode to station.

Command:

```
AT+CWMODE=1
```

Response:

```
OK
```

2. Connect to the router.

Command:

```
AT+CWJAP="espressif","1234567890"
```

Response:

```
WIFI CONNECTED  
WIFI GOT IP  
  
OK
```

Note:

- The SSID and password you entered may be different from those in the above command. Please replace the SSID and password with those of your router settings.
3. Query the device's IP address.
Command:


```
AT+CIPSTA?
```

Response:

```
+CIPSTA:ip:"192.168.3.112"
+CIPSTA:gateway:"192.168.3.1"
+CIPSTA:netmask:"255.255.255.0"
```

OK

Note:

- The query results you obtained may be different from those in the above response.

4. Connect the PC to the same router which ESP32 is connected to.

Use a network tool on the PC to create UDP transmission. For example, the PC's IP address is 192.168.3.102, and the port is 8080.

5. Enable single connections.

Command:

```
AT+CIPMUX=0
```

Response:

OK

6. Create a UDP transmission. The remote host's IP address is 192.168.3.102, the remote port is 8080, the local port is 1112, and the mode is 2.

Command:

```
AT+CIPSTART="UDP", "192.168.3.102", 8080, 1112, 2
```

Response:

CONNECT

OK

Note:

- "192.168.3.102" and 8080 are the remote IP address and port of UDP transmission on the remote side, i.e., the UDP configuration set by PC.
- 1112 is the local port number of ESP32. You can define this port number, or else, a random port will be used.
- 2 means the opposite terminal of UDP transmission can be changed. The remote IP address and port will be automatically changed to those of the last UDP connection to ESP32.

7. Send 4 bytes of data.

Command:

```
AT+CIPSEND=4
```

Response:

OK

>

Input 4 bytes, for example, test, then AT will respond the following messages.

Recv 4 bytes

SEND OK

Note:

- If the number of bytes inputted are more than the length (n) set by AT+CIPSEND, the system will reply busy p . . . , and send the first n bytes. And after sending the first n bytes, the system will reply SEND OK.

8. Send data to any other UDP terminal. For example, you can send 4 bytes of data with the remote host's IP address as 192.168.3.103 and the remote port as 1000.

If you want to send data to any other UDP terminal, please designate the IP address and port of the target terminal in the command.

Command:

```
AT+CIPSEND=4, "192.168.3.103", 1000
```

Response:

```
OK
```

```
>
```

Input 4 bytes, for example, `test`, then AT will respond the following messages.

```
Recv 4 bytes
```

```
SEND OK
```

9. Receive 4 bytes of data.

Assume that the PC sends 4 bytes of data (data is `test`), the system will prompt:

```
+IPD,4:test
```

10. Close UDP connection.

Command:

```
AT+CIPCLOSE
```

Response:

```
CLOSED
```

```
OK
```

4.3.5 ESP32 as an SSL client in single connection

1. Set the Wi-Fi mode to station.

Command:

```
AT+CWMODE=1
```

Response:

```
OK
```

2. Connect to the router.

Command:

```
AT+CWJAP="espressif", "1234567890"
```

Response:

```
WIFI CONNECTED
```

```
WIFI GOT IP
```

```
OK
```

Note:

- The SSID and password you entered may be different from those in the above command. Please replace the SSID and password with those of your router settings.

3. Query the device's IP address.

Command:

```
AT+CIPSTA?
```

Response:

```
+CIPSTA:ip:"192.168.3.112"
+CIPSTA:gateway:"192.168.3.1"
+CIPSTA:netmask:"255.255.255.0"

OK
```

Note:

- The query results you obtained may be different from those in the above response.
- Connect the PC to the same router which ESP32 is connected to.
 - Use the OpenSSL command on the PC to create an SSL server. For example, the SSL server on PC is 192.168.3.102, and the port is 8070.

Command:

```
openssl s_server -cert /home/esp-at/components/customized_partitions/raw_data/
→server_cert/server_cert.crt -key /home/esp-at/components/customized_
→partitions/raw_data/server_key/server.key -port 8070
```

Response:

```
ACCEPT
```

- Connect the ESP32 to the SSL server as a client over SSL. The server's IP address is 192.168.3.102, and the port is 8070.

Command:

```
AT+CIPSTART="SSL", "192.168.3.102", 8070
```

Response:

```
CONNECT

OK
```

- Send 4 bytes of data.

Command:

```
AT+CIPSEND=4
```

Response:

```
OK

>
```

Input 4 bytes, for example, test, then AT will respond the following message.

```
Recv 4 bytes

SEND OK
```

Note:

- If the number of bytes inputted are more than the length (n) set by AT+CIPSEND, the system will reply busy p . . . , and send the first n bytes. And after sending the first n bytes, the system will reply SEND OK.
- Receive 4 bytes of data.
- Assume that the SSL server sends 4 bytes of data (data is test), the system will prompt:

```
+IPD,4:test
```

4.3.6 ESP32 as an SSL server in multiple connections

When ESP32 works as an SSL server, multiple connections should be enabled by [AT+CIPMUX=1](#) command, because in most cases more than one client needs to be connected to the ESP32 server.

Below is an example showing how an SSL server is established when ESP32 works in the softAP mode. If ESP32 works as a station, after connecting to the router, follow the steps for establishing a connection to an SSL server in this example.

1. Set the Wi-Fi mode to softAP.

Command:

```
AT+CWMODE=2
```

Response:

```
OK
```

2. Enable multiple connections.

Command:

```
AT+CIPMUX=1
```

Response:

```
OK
```

3. Configure the ESP32 softAP.

Command:

```
AT+CWSAP="ESP32_softAP","1234567890",5,3
```

Response:

```
OK
```

4. Query softAP information.

Command:

```
AT+CIPAP?
```

Response:

```
AT+CIPAP?  
+CIPAP:ip:"192.168.4.1"  
+CIPAP:gateway:"192.168.4.1"  
+CIPAP:netmask:"255.255.255.0"  
OK
```

Note:

- The address you obtained may be different from that in the above response.

5. Set up an SSL server.

Command:

```
AT+CIPSERVER=1,8070,"SSL"
```

Response:

```
OK
```

6. Connect the PC to the ESP32 softAP.

7. Use the OpenSSL command on PC to create an SSL client and connect it to the SSL server that ESP32 has created.

Command:

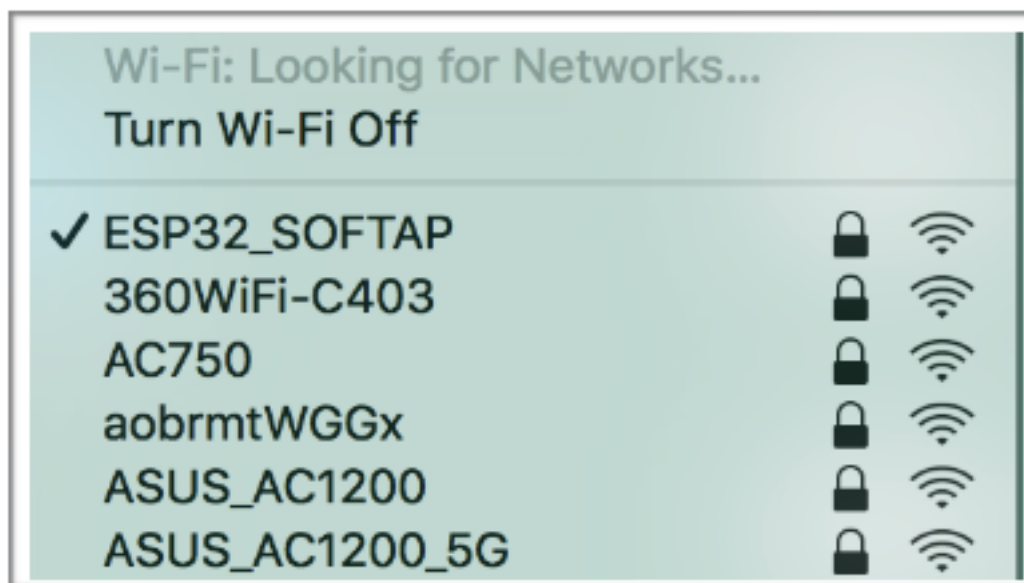
```
openssl s_client -host 192.168.4.1 -port 8070
```

Response on the ESP32:

```
CONNECT
```

8. Send 4 bytes of data to connection link 0.

Command:



```
AT+CIPSEND=0,4
```

Response:

```
OK
```

```
>
```

Input 4 bytes, for example, `test`, then AT will respond the following messages.

```
Recv 4 bytes
```

```
SEND OK
```

Note:

- If the number of bytes inputted are more than the length (n) set by `AT+CIPSEND`, the system will reply `busy p...`, and send the first n bytes. And after sending the first n bytes, the system will reply `SEND OK`.
9. Receive 4 bytes of data from connection link 0.
Assume that the SSL server sends 4 bytes of data (data is `test`), the system will prompt:

```
+IPD,0,4:test
```

10. Close SSL connection.

Command:

```
AT+CIPCLOSE=0
```

Response:

```
0,CLOSED
```

```
OK
```

4.3.7 ESP32 as an SSL client to create a single connection with mutual authentication

In this example, the certificate used is the default certificate in `esp-at`. You can also use your own certificate:

- To use your own SSL client certificate, you can either update it at runtime using the [AT+SYSMFG](#) command (refer to [PKI Configuration in AT+SYSMFG command examples](#)), or replace the default certificate by recompiling the firmware (refer to [How to Update PKI Configuration](#)).

- To use your own SSL server certificate, replace the SSL server certificate path below with your own certificate path.

1. Set the Wi-Fi mode to station.

Command:

```
AT+CWMODE=1
```

Response:

```
OK
```

2. Connect to the router.

Command:

```
AT+CWJAP="espressif","1234567890"
```

Response:

```
WIFI CONNECTED
WIFI GOT IP

OK
```

Note:

- The SSID and password you entered may be different from those in the above command. Please replace the SSID and password with those of your router settings.

3. Set the SNTP server.

Command:

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

Response:

```
OK
```

Note:

- You can set the SNTP server according to your country's time zone.

4. Query the SNTP time.

Command:

```
AT+CIPSNTPTIME?
```

Response:

```
+CIPSNTPTIME:Mon Oct 18 20:12:27 2021
OK
```

Note:

- You can check whether the SNTP time matches the real-time time to determine whether the SNTP server you set takes effect.

5. Query the device's IP address.

Command:

```
AT+CIPSTA?
```

Response:

```
+CIPSTA:ip:"192.168.3.112"
+CIPSTA:gateway:"192.168.3.1"
+CIPSTA:netmask:"255.255.255.0"

OK
```

Note:

- The query results you obtained may be different from those in the above response.

6. Connect the PC to the same router which ESP32 is connected to.

7. Use the OpenSSL command on the PC to create an SSL server. For example, the SSL server on PC is 192.168.3.102, and the port is 8070.

Command:

```
openssl s_server -CAfile /home/esp-at/components/customized_partitions/raw_data/server_ca/server_ca.crt -cert /home/esp-at/components/customized_partitions/raw_data/server_cert/server_cert.crt -key /home/esp-at/components/customized_partitions/raw_data/server_key/server.key -port 8070 -verify_return_error -verify_depth 1 -Verify 1
```

Response on the ESP32:

```
CONNECT
```

Note:

- The certificate path in the command can be adjusted according to the location of your certificate.

8. The ESP32 sets up the SSL client mutual authentication configuration.

Command:

```
AT+CIPSSLCONF=3,0,0
```

Response:

```
OK
```

9. Connect the ESP32 to the SSL server as a client over SSL. The server's IP address is 192.168.3.102, and the port is 8070.

Command:

```
AT+CIPSTART="SSL", "192.168.3.102", 8070
```

Response:

```
CONNECT
```

```
OK
```

10. Send 4 bytes of data.

Command:

```
AT+CIPSEND=4
```

Response:

```
OK
```

```
>
```

Input 4 bytes, for example, test, then AT will respond the following message.

```
Recv 4 bytes
```

```
SEND OK
```

Note:

- If the number of bytes inputted are more than the length (n) set by AT+CIPSEND, the system will reply busy p . . . , and send the first n bytes. And after sending the first n bytes, the system will reply SEND OK.

11. Receive 4 bytes of data.

Assume that the SSL server sends 4 bytes of data (data is test), the system will prompt:

```
+IPD,4:test
```

4.3.8 ESP32 as an SSL server to create multiple connections with mutual authentication

When ESP32 works as an SSL server, multiple connections should be enabled by `AT+CIPMUX=1` command, because in most cases more than one client needs to be connected to the ESP32 server.

Below is an example showing how an SSL server is established when ESP32 works in the station mode. If ESP32 works as a softAP, refer to the example of [ESP32 as an SSL server in multiple connections](#).

1. Set the Wi-Fi mode to station.

Command:

```
AT+CWMODE=1
```

Response:

```
OK
```

2. Connect to the router.

Command:

```
AT+CWJAP="espressif","1234567890"
```

Response:

```
WIFI CONNECTED
WIFI GOT IP
OK
```

Note:

- The SSID and password you entered may be different from those in the above command. Please replace the SSID and password with those of your router settings.

3. Query the device's IP address.

Command:

```
AT+CIPSTA?
```

Response:

```
+CIPSTA:ip:"192.168.3.112"
+CIPSTA:gateway:"192.168.3.1"
+CIPSTA:netmask:"255.255.255.0"
OK
```

Note:

- The query results you obtained may be different from those in the above response.

4. Enable multiple connections.

Command:

```
AT+CIPMUX=1
```

Response:

```
OK
```

5. Set up an SSL server.

Command:

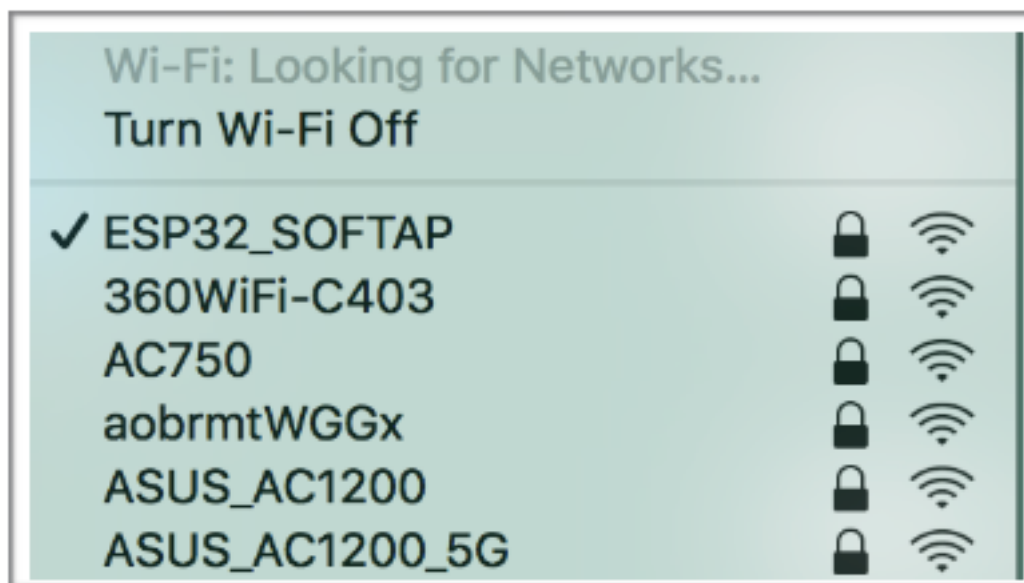
```
AT+CIPSERVER=1,8070,"SSL",1
```

Response:

```
OK
```

6. Connect the PC to the ESP32 softAP.

7. Use the OpenSSL command on PC to create an SSL client and connect it to the SSL server that ESP32 has created.



Command:

```
openssl s_client -CAfile /home/esp-at/components/customized_partitions/raw_
→data/client_ca/client_ca_00.crt -cert /home/esp-at/components/customized_
→partitions/raw_data/client_cert/client_cert_00.crt -key /home/esp-at/
→components/customized_partitions/raw_data/client_key/client_key_00.key -host_
→192.168.3.112 -port 8070
```

Response on the ESP32:

```
0, CONNECT
```

8. Send 4 bytes of data to connection link 0.

Command:

```
AT+CIPSEND=0,4
```

Response:

```
OK
```

```
>
```

Input 4 bytes, for example, `test`, then AT will respond the following messages.

```
Recv 4 bytes
```

```
SEND OK
```

Note:

- If the number of bytes inputted are more than the length (n) set by `AT+CIPSEND`, the system will reply `busy p . . .`, and send the first n bytes. And after sending the first n bytes, the system will reply `SEND OK`.

9. Receive 4 bytes of data from connection link 0.

Assume that the SSL server sends 4 bytes of data (data is `test`), the system will prompt:

```
+IPD,0,4:test
```

10. Close SSL connection.

Command:

```
AT+CIPCLOSE=0
```

Response:

```
0,CLOSED
```

```
OK
```

11. Close SSL server.

Command:

```
AT+CIPSERVER=0
```

Response:

```
OK
```

4.3.9 Network passthrough transmission when the ESP32 works as a TCP client in single connection

1. Set the Wi-Fi mode to station.

Command:

```
AT+CWMODE=1
```

Response:

```
OK
```

2. Connect to the router.

Command:

```
AT+CWJAP="espressif","1234567890"
```

Response:

```
WIFI CONNECTED
```

```
WIFI GOT IP
```

```
OK
```

Note:

- The SSID and password you entered may be different from those in the above command. Please replace the SSID and password with those of your router settings.

3. Query the device's IP address.

Command:

```
AT+CIPSTA?
```

Response:

```
+CIPSTA:ip:"192.168.3.112"
```

```
+CIPSTA:gateway:"192.168.3.1"
```

```
+CIPSTA:netmask:"255.255.255.0"
```

```
OK
```

Note:

- The query results you obtained may be different from those in the above response.

4. Connect the PC to the same router which ESP32 is connected to.

Use a network tool on the PC to create a TCP server. For example, the TCP server on PC is 192.168.3.102, and the port is 8080.

5. Connect the ESP32 to the TCP server as a TCP client over TCP. The server's IP address is 192.168.3.102, and the port is 8080.

Command:

```
AT+CIPSTART="TCP", "192.168.3.102", 8080
```

Response:

```
CONNECT
OK
```

6. Enable the Network *Passthrough Receiving Mode*.

Command:

```
AT+CIPMODE=1
```

Response:

```
OK
```

7. Enter the Network *Passthrough Mode* and send data.

Command:

```
AT+CIPSEND
```

Response:

```
OK
>
```

8. Stop sending data.

When receiving a packet that contains only +++, the Network passthrough transmission process will be stopped. Then please wait at least 1 second before sending the next AT command. Please note that if you input +++ directly by typing, the +++ may not be recognized as three consecutive + because of the prolonged typing duration. For more details, please refer to [\[Passthrough Mode Only\] +++](#).

Important: The aim of ending the packet with +++ is to exit *Passthrough Mode* and to accept normal AT commands, while TCP still remains connected. However, you can also use command AT+CIPSEND to go back into *Passthrough Mode*.

9. Exit the Network *Passthrough Receiving Mode*.

Command:

```
AT+CIPMODE=0
```

Response:

```
OK
```

10. Close TCP connection.

Command:

```
AT+CIPCLOSE
```

Response:

```
CLOSED
OK
```

4.3.10 Network passthrough transmission when the ESP32 works as a TCP server

1. Set the Wi-Fi mode to station.

Command:

```
AT+CWMODE=1
```

Response:

```
OK
```

2. Connect to the router.

Command:

```
AT+CWJAP="espressif","1234567890"
```

Response:

```
WIFI CONNECTED  
WIFI GOT IP
```

```
OK
```

Note:

- The SSID and password you entered may be different from those in the above command. Please replace the SSID and password with those of your router settings.

3. Enable multiple connections.

Command:

```
AT+CIPMUX=1
```

Response:

```
OK
```

Note:

- TCP server can be created only in multiple connections.

4. Set the maximum number of TCP server connections to 1.

Command:

```
AT+CIPSERVERMAXCONN=1
```

Response:

```
OK
```

Note:

- The passthrough mode is point-to-point, so the maximum number of connections to the TCP server can only be 1.

5. Create TCP server.

Command:

```
AT+CIPSERVER=1,8080
```

Response:

```
OK
```

Note:

- Set the TCP server port to 8080. You can also set it to other port.

6. Query the device's IP address.

Command:

```
AT+CIPSTA?
```

Response:

```
+CIPSTA:ip:"192.168.3.112"  
+CIPSTA:gateway:"192.168.3.1"  
+CIPSTA:netmask:"255.255.255.0"
```

```
OK
```

Note:

- The query results you obtained may be different from those in the above response.
7. Connect the PC to the ESP32 TCP server
Connect the PC to the same router which ESP32 is connected to.
Use a network tool on the PC to create a TCP client and connect to the ESP32 TCP server. The remote address is 192.168.3.112, and the port is 8080.

AT Response:

```
0,CONNECT
```

8. Enable the Network *Passthrough Receiving Mode*.

Command:

```
AT+CIPMODE=1
```

Response:

```
OK
```

9. Enter the Network *Passthrough Mode* and send data.

Command:

```
AT+CIPSEND
```

Response:

```
OK
```

```
>
```

10. Stop sending data.

When receiving a packet that contains only +++, the Network passthrough transmission process will be stopped. Then please wait at least 1 second before sending the next AT command. Please note that if you input +++ directly by typing, the +++ may not be recognized as three consecutive + because of the prolonged typing duration. For more details, please refer to [\[Passthrough Mode Only\] +++](#).

Important: The aim of ending the packet with +++ is to exit *Passthrough Mode* and to accept normal AT commands, while TCP still remains connected. However, you can also use command AT+CIPSEND to go back into *Passthrough Mode*.

11. Exit the Network *Passthrough Receiving Mode*.

Command:

```
AT+CIPMODE=0
```

Response:

```
OK
```

12. Close TCP connection.

Command:

```
AT+CIPCLOSE
```

Response:

```
CLOSED
```

```
OK
```

4.3.11 Network passthrough transmission when the ESP32 works as a softAP in UDP transparent transmission

1. Set the Wi-Fi mode to softAP.

Command:

```
AT+CWMODE=2
```

Response:

```
OK
```

2. Set softAP.

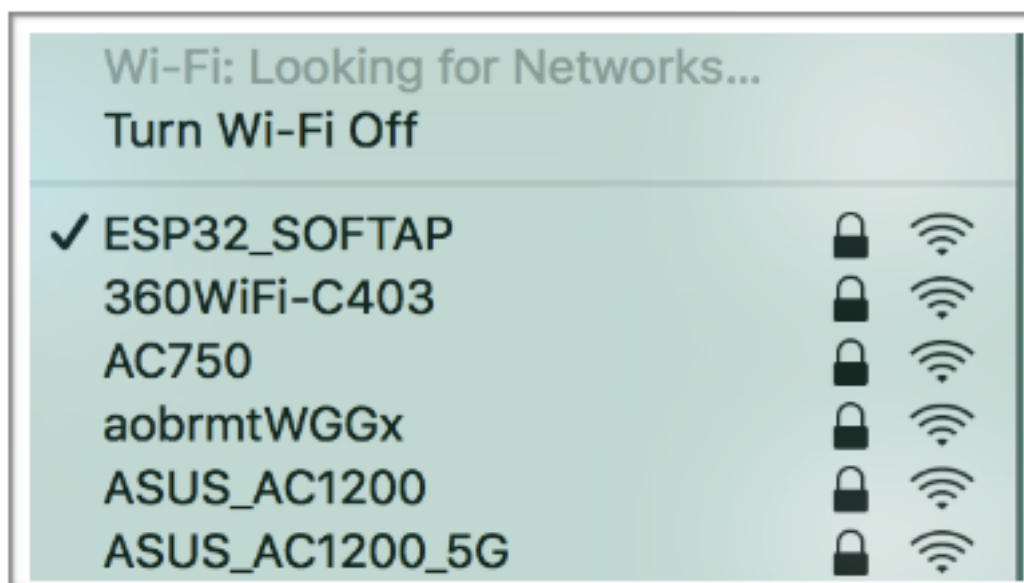
Command:

```
AT+CWSAP="ESP32_softAP","1234567890",5,3
```

Response:

```
OK
```

3. Connect the PC to the ESP32 softAP.



4. Create a UDP endpoint.

Use a network tool on PC to create a UDP endpoint. For example, the PC's IP address is 192.168.4.2 and the port is 8080.

5. Create a UDP transmission between ESP32 and the PC with a fixed remote IP address and port. The remote host's IP address is 192.168.4.2, the remote port is 8080, the local port is 2233, and the mode is 0.

Command:

```
AT+CIPSTART="UDP","192.168.4.2",8080,2233,0
```

Response:

```
CONNECT
```

```
OK
```

6. Enter the Network *Passthrough Receiving Mode*.

Command:

```
AT+CIPMODE=1
```

Response:

```
OK
```

7. Enter the Network *Passthrough Mode* and send data.

Command:

```
AT+CIPSEND
```

Response:

```
OK  
>
```

8. Stop sending data.

When receiving a packet that contains only +++, the Network passthrough transmission process will be stopped. Then please wait at least 1 second before sending the next AT command. Please note that if you input +++ directly by typing, the +++ may not be recognized as three consecutive + because of the prolonged typing duration. For more details, please refer to [\[Passthrough Mode Only\] +++](#).

Important: The aim of ending the packet with +++ is to exit [Passthrough Mode](#) and to accept normal AT commands, while TCP still remains connected. However, you can also use command AT+CIPSEND to go back into [Passthrough Mode](#).

9. Exit the Network [Passthrough Receiving Mode](#).

Command:

```
AT+CIPMODE=0
```

Response:

```
OK
```

10. Close TCP connection.

Command:

```
AT+CIPCLOSE
```

Response:

```
CLOSED  
OK
```

4.3.12 ESP32 obtains socket data in passive receiving mode

When a large amount of network data is expected to be received and the MCU cannot process it timely, you can refer to this example and use the passive receive data mode.

1. Set the Wi-Fi mode to station.

Command:

```
AT+CWMODE=1
```

Response:

```
OK
```

2. Connect to the router.

Command:

```
AT+CWJAP="espressif","1234567890"
```

Response:

```
WIFI CONNECTED  
WIFI GOT IP  
OK
```

Note:

- The SSID and password you entered may be different from those in the above command. Please replace the SSID and password with those of your router settings.

3. Query the device's IP address.

Command:

```
AT+CIPSTA?
```

Response:

```
+CIPSTA:ip:"192.168.3.112"  
+CIPSTA:gateway:"192.168.3.1"  
+CIPSTA:netmask:"255.255.255.0"  
  
OK
```

Note:

- The query results you obtained may be different from those in the above response.

4. Connect the PC to the same router which ESP32 is connected to.

Use a network tool on the PC to create a TCP server. For example, the TCP server on PC is 192.168.3.102, and the port is 8080.

5. Connect ESP32 to the TCP server as a client over TCP. The server's IP address is 192.168.3.102, and the port is 8080.

Command:

```
AT+CIPSTART="TCP", "192.168.3.102", 8080
```

Response:

```
CONNECT  
  
OK
```

6. ESP32 sets the socket receiving mode to passive mode.

Command:

```
AT+CIPRECVMODE=1
```

Response:

```
OK
```

7. The TCP server sends 4 bytes of data (data is test).

Note:

- The device replies with +IPD, 4. If it receives server data again later, please refer to the note section of [AT+CIPRECVMODE](#) for whether it will reply with +IPD, .

8. ESP32 obtains socket data length in passive receiving mode.

Command:

```
AT+CIPRECLEN?
```

Response:

```
+CIPRECLEN:4  
OK
```

9. ESP32 obtains socket data in passive receiving mode.

Command:

```
AT+CIPRECVDATA=4
```

Response:

```
+CIPRECVDATA:4,test  
OK
```


4.3.13 ESP32 enables mDNS service, PC connects to the device's TCP server using a domain name

1. Set the Wi-Fi mode to station.

Command:

```
AT+CWMODE=1
```

Response:

```
OK
```

2. Connect to the router.

Command:

```
AT+CWJAP="espressif","1234567890"
```

Response:

```
WIFI CONNECTED
WIFI GOT IP
OK
```

Note:

- The SSID and password you entered may be different from those in the above command. Please replace the SSID and password with those of your router settings.

3. Connect the PC to the same router which ESP32 is connected to.

PC can discover the ESP32 device only when they are in the same LAN.

4. Use an mDNS tool on the PC to enable service discovery. For example, use [avahi-browse](#) on linux (use [Bonjour](#) on macOS or Windows).

Command:

```
sudo avahi-browse -a -r
```

5. ESP32 device enables mDNS service.

Command:

```
AT+MDNS=1,"espressif","_printer",35,"my_instance","_tcp",2,"product","my_
printer","firmware_version","AT-V3.4.1.0"
```

Response:

```
OK
```

Note:

- This command enables mDNS service and specifies the device instance name as `my_instance`, service type as `_printer`, port as `35`, product as `my_printer`, and firmware version as `AT-V3.4.1.0`.

6. (Optional) PC discovers the ESP32 device.

The PC's `avahi-browse` tool will display:

```
...
+ enx000ec6dd4ebf IPv4 my_instance                                UNIX
→Printer          local
= enx000ec6dd4ebf IPv4 my_instance                                UNIX
→Printer          local
  hostname = [espressif.local]
  address  = [192.168.200.90]
  port     = [35]
  txt      = ["product=my_printer" "firmware_version=AT-V3.4.1.0"]
```

Note:

- This step is not necessary, it is just to verify the mDNS function of the ESP32 device.

7. ESP32 device enables multiple connections.

Command:

```
AT+CIPMUX=1
```

Response:

```
OK
```

8. ESP32 device runs a TCP server on port 35.

Command:

```
AT+CIPSERVER=1,35
```

Response:

```
OK
```

9. Use a TCP tool (For example, use [nc](#) on Linux or macOS, use [ncat](#) on Windows) on the PC to connect to ESP32 device's TCP server using the domain name.

Command:

```
nc espressif.local 35
```

ESP32 device responds:

```
0,CONNECT
```

Note:

- After the connection is established, data transmission can immediately take place between PC and ESP32 device.

10. ESP32 device disables mDNS function.

Command:

```
AT+MDNS=0
```

Response:

```
OK
```

Note:

- Disabling mDNS function can reduce the device's power consumption to some extent.

4.4 Bluetooth LE AT Examples

This document provides an introduction and detailed command examples to illustrate how to utilize [Bluetooth® Low Energy AT Commands](#) on ESP32.

- [Introduction](#)
- [Bluetooth LE client reads and write services](#)
- [Encrypt Bluetooth LE connection](#)
- [Establish SPP connection between two ESP32 development boards and transmit data in UART-Bluetooth LE Passthrough Mode](#)
- [Establish SPP connection between ESP32 and mobile phone and transmit data in UART-Bluetooth LE passthrough mode](#)
- [Establish Bluetooth LE connection and pairing between ESP32 and mobile phone](#)

4.4.1 Introduction

ESP-AT currently only supports **Bluetooth LE 4.2 protocol specification**, and the description in this document is only for **Bluetooth LE 4.2 protocol specification**. Please refer to [Core Specification 4.2](#) for more details.

Bluetooth LE protocol architecture

Bluetooth LE protocol stack is divided into several layers from bottom to top: Physical Layer (PHY), Link Layer (LL), Host Controller Interface (HCI), Logical Link Control and Adaptation Protocol Layer (L2CAP), Attribute Protocol (ATT), Security Manager Protocol (SMP), Generic Attribute Profile (GATT), Generic Access Profile (GAP).

- **PHY:** the PHY layer is responsible for transmitting and receiving packets of information on the physical channel. Bluetooth LE uses 40 RF channels. Frequency Range: 2402 MHz to 2480 MHz.
- **LL:** the LL layer is responsible for the creation, modification and release of logical links (and, if required, their associated logical transports), as well as the update of parameters related to physical links between devices. It controls the Link Layer state machine in one of the five states of `standby`, `advertising`, `scanning`, `initiating` and `connected`.
- **HCI:** the HCI layer provides a standardized interface to host and controller. This layer can be implemented by software API or controlled by hardware interfaces UART, SPI and USB.
- **L2CAP:** the L2CAP layer is responsible for protocol multiplexing capability, segmentation, and reassembly operation for data exchanged between the host and the protocol stack.
- **ATT:** the ATT layer implements the peer-to-peer protocol between an attribute server and an attribute client. The ATT client sends commands, requests, and confirmations to the ATT server. The ATT server sends responses, notifications and indications to the client.
- **SMP:** the SMP layer is the peer-to-peer protocol used to generate encryption keys and identity keys. The SMP also manages storage of the encryption keys and identity keys and is responsible for generating random addresses and resolving random addresses to known device identities.
- **GATT:** the GATT layer represents the functionality of the attribute server and, optionally, the attribute client. The profile describes the hierarchy of services, characteristics and attributes used in the attribute server. The layer provides interfaces for discovering, reading, writing and indicating of service characteristics and attributes.
- **GAP:** the GAP represents the base functionality common to all Bluetooth devices such as modes and access procedures used by the transports, protocols and application profiles. GAP services include device discovery, connection modes, security, authentication, association models and service discovery.

Bluetooth LE role division

There are different roles in different levels of the Bluetooth LE protocol architecture. These divisions are not affected by each other.

- **LL:** devices can be divided into `master` and `slave`, the `slave` advertises, and the `master` initiates a connection.
- **GAP:** GAP defines four specific roles: `broadcaster`, `observer`, `peripheral`, and `central`.
- **GATT:** devices can be divided into `server` and `client`.

Important:

- The Bluetooth LE server and Bluetooth LE client described in this document are both GATT layer roles.
 - At present, ESP-AT supports the existence of Bluetooth LE server and Bluetooth LE client at the same time.
 - No matter ESP-AT is initialized as Bluetooth LE server or Bluetooth LE client, the maximum number of devices connected at the same time is 3.
-

GATT is actually an attribute transmission protocol, which can be regarded as an application layer protocol for attribute transmission. The structure of this attribute is very simple. It is composed of `services`, each `service` is composed of different numbers of `characteristics`, and each `characteristic` is composed of many other elements.

GATT `server` and GATT `client` roles exist after the Bluetooth LE connection is established. The GATT server stores the data transported over the Attribute Protocol and accepts Attribute Protocol requests, commands and confirmations from the GATT client. In short, the end that provides data is called GATT `server`, and the end that accesses data is called GATT `client`.

4.4.2 Bluetooth LE client reads and write services

Below is an example of using two ESP32 development boards, one as a Bluetooth LE server (only as Bluetooth LE server role), the other one as a Bluetooth LE client (only as Bluetooth LE client role). The example shows how to use Bluetooth LE functions with AT commands.

Important: In the following steps, the operations starting with ESP32 Bluetooth LE server only need to be executed at ESP32 Bluetooth LE server, and the operations starting with ESP32 Bluetooth LE client only need to be executed at ESP32 Bluetooth LE client.

1. Bluetooth LE initialization.

ESP32 Bluetooth LE server:

Command:

```
AT+BLEINIT=2
```

Response:

```
OK
```

ESP32 Bluetooth LE client:

Command:

```
AT+BLEINIT=1
```

Response:

```
OK
```

2. ESP32 Bluetooth LE server gets Bluetooth LE address.

Command:

```
AT+BLEADDR?
```

Response:

```
+BLEADDR: "24:0a:c4:d6:e4:46"  
OK
```

Note:

- The address you obtain may be different from that in the above response. Keep yours handy as you will need it in one of the following steps.

1. ESP32 Bluetooth LE server creates services.

Command:

```
AT+BLEGATTSSRVCRE
```

Response:

```
OK
```

2. ESP32 Bluetooth LE server starts services.

Command:

```
AT+BLEGATTSSRVSTART
```

Response:

```
OK
```

1. ESP32 Bluetooth LE server discovers characteristics.

Command:

```
AT+BLEGATTTSCHAR?
```

Response:

```
+BLEGATTTSCHAR:"char",1,1,0xC300,0x02
+BLEGATTTSCHAR:"desc",1,1,1,0x2901
+BLEGATTTSCHAR:"char",1,2,0xC301,0x02
+BLEGATTTSCHAR:"desc",1,2,1,0x2901
+BLEGATTTSCHAR:"char",1,3,0xC302,0x08
+BLEGATTTSCHAR:"desc",1,3,1,0x2901
+BLEGATTTSCHAR:"char",1,4,0xC303,0x04
+BLEGATTTSCHAR:"desc",1,4,1,0x2901
+BLEGATTTSCHAR:"char",1,5,0xC304,0x08
+BLEGATTTSCHAR:"char",1,6,0xC305,0x10
+BLEGATTTSCHAR:"desc",1,6,1,0x2902
+BLEGATTTSCHAR:"char",1,7,0xC306,0x20
+BLEGATTTSCHAR:"desc",1,7,1,0x2902
+BLEGATTTSCHAR:"char",1,8,0xC307,0x02
+BLEGATTTSCHAR:"desc",1,8,1,0x2901
+BLEGATTTSCHAR:"char",2,1,0xC400,0x02
+BLEGATTTSCHAR:"desc",2,1,1,0x2901
+BLEGATTTSCHAR:"char",2,2,0xC401,0x02
+BLEGATTTSCHAR:"desc",2,2,1,0x2901
```

OK

2. ESP32 Bluetooth LE server starts advertising, then the ESP32 Bluetooth LE client starts scanning and lasts for 3 s.

ESP32 Bluetooth LE server:

Command:

```
AT+BLEADVSTART
```

Response:

OK

ESP32 Bluetooth LE client:

Command:

```
AT+BLESCAN=1,3
```

Response:

```
OK
+BLESCAN:"5b:3b:6c:51:90:49",-87,02011a020a0c0aff4c001005071c3024dc,,1
+BLESCAN:"c4:5b:be:93:ec:66",-84,0201060303111809095647543147572d58020a03,,0
+BLESCAN:"24:0a:c4:d6:e4:46",-29,,,0
```

Note:

- The scan results you obtain may be different from those in the above response.

3. Establish the Bluetooth LE connection.

ESP32 Bluetooth LE client:

Command:

```
AT+BLECONN=0,"24:0a:c4:d6:e4:46"
```

Response:

```
+BLECONN:0,"24:0a:c4:d6:e4:46"
```

OK

Note:

- When entering the above command, replace the address with your ESP32 Bluetooth LE server address.
- If the Bluetooth LE connection is established successfully, message +BLECONN:0,

"24:0a:c4:d6:e4:46 will be prompted.

- If the Bluetooth LE connection is broken, message +BLECONN:0,-1 will be prompted.

4. ESP32 Bluetooth LE client discovers services.

Command:

```
AT+BLEGATTCPRIMSRV=0
```

Response:

```
+BLEGATTCPRIMSRV:0,1,0x1801,1
+BLEGATTCPRIMSRV:0,2,0x1800,1
+BLEGATTCPRIMSRV:0,3,0xA002,1
+BLEGATTCPRIMSRV:0,4,0xA003,1

OK
```

Note:

- When discovering services, the ESP32 Bluetooth LE client will get two more default services (UUID: 0x1800 and 0x1801) than what the ESP32 Bluetooth LE server will get. So, for the same service, the <srv_index> received by the ESP32 Bluetooth LE client equals the <srv_index> received by the ESP32 Bluetooth LE server + 2. For example, for service 0xA002, the <srv_index> queried on the ESP32 Bluetooth LE client is 3, if the ESP32 Bluetooth LE server is queried through the command [AT+BLEGATTSSRV?](#), then <srv_index> is 1.

5. ESP32 Bluetooth LE client discovers characteristics.

Command:

```
AT+BLEGATTCCCHAR=0,3
```

Response:

```
+BLEGATTCCCHAR:"char",0,3,1,0xC300,0x02
+BLEGATTCCCHAR:"desc",0,3,1,1,0x2901
+BLEGATTCCCHAR:"char",0,3,2,0xC301,0x02
+BLEGATTCCCHAR:"desc",0,3,2,1,0x2901
+BLEGATTCCCHAR:"char",0,3,3,0xC302,0x08
+BLEGATTCCCHAR:"desc",0,3,3,1,0x2901
+BLEGATTCCCHAR:"char",0,3,4,0xC303,0x04
+BLEGATTCCCHAR:"desc",0,3,4,1,0x2901
+BLEGATTCCCHAR:"char",0,3,5,0xC304,0x08
+BLEGATTCCCHAR:"char",0,3,6,0xC305,0x10
+BLEGATTCCCHAR:"desc",0,3,6,1,0x2902
+BLEGATTCCCHAR:"char",0,3,7,0xC306,0x20
+BLEGATTCCCHAR:"desc",0,3,7,1,0x2902
+BLEGATTCCCHAR:"char",0,3,8,0xC307,0x02
+BLEGATTCCCHAR:"desc",0,3,8,1,0x2901

OK
```

6. ESP32 Bluetooth LE client reads a characteristic.

Command:

```
AT+BLEGATTCCRD=0,3,1
```

Response:

```
+BLEGATTCCRD:0,1,0

OK
```

Note:

- Please note that the target characteristic's property has to support the read operation.
- If the ESP32 Bluetooth LE client reads the characteristic successfully, message +READ:0,"7c:df:a1:b3:8d:de" will be prompted on the ESP32 Bluetooth LE Server side.

7. ESP32 Bluetooth LE client writes a characteristic.

Command:

```
AT+BLEGATTCWR=0,3,3,,2
```

Response:

```
>
```

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

```
OK
```

Note:

- If the ESP32 Bluetooth LE client writes the characteristic successfully, message +WRITE:<conn_index>,<srv_index>,<char_index>,[<desc_index>],<len>,<value> will be prompted on the ESP32 Bluetooth LE server side.

8. Indicate a characteristic.

ESP32 Bluetooth LE client:

Command:

```
AT+BLEGATTCWR=0,3,7,1,2
```

Response:

```
>
```

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

To receive data from ESP32 Bluetooth LE server (through the `notify` or the `indicate` method), the ESP32 Bluetooth LE client needs to register with the server in advance. Write the value 0x0001 to use the `notify` method, and 0x0002 to use the `indicate` method. This example writes the 0x0002 to use the `indicate` method.

```
OK
```

Note:

- If the ESP32 Bluetooth LE client writes the descriptor successfully, message +WRITE:<conn_index>,<srv_index>,<char_index>,<desc_index>,<len>,<value> will be prompted on the ESP32 Bluetooth LE server side.

ESP32 Bluetooth LE server:

Command:

```
AT+BLEGATTSIND=0,1,7,3
```

Response:

```
>
```

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

```
OK
```

Note:

- If the ESP32 Bluetooth LE client receives the indication, message +INDICATE:<conn_index>,<srv_index>,<char_index>,<len>,<value> will be prompted.
- For the same service, the <srv_index> on the ESP32 Bluetooth LE client side equals the <srv_index> on the ESP32 Bluetooth LE server side + 2.
- For the permissions of the characteristics in the services, please refer to [How to Customize Bluetooth® LE Services](#).

4.4.3 Bluetooth LE server read and write services

Below is an example of using two ESP32 development boards, one as a Bluetooth LE server (only as Bluetooth LE server role), the other one as a Bluetooth LE client (only as Bluetooth LE client role). The example shows how to

establish a Bluetooth LE connection, as well as the read and write characteristics of the server and client settings, and notification characteristics.

Important: In the step, the operations starting with ESP32 Bluetooth LE server only need to be executed at ESP32 Bluetooth LE server, and the operations starting with ESP32 Bluetooth LE client only need to be executed at ESP32 Bluetooth LE client.

1. Bluetooth LE initialization.

ESP32 Bluetooth LE server:

Command:

```
AT+BLEINIT=2
```

Response:

```
OK
```

ESP32 Bluetooth LE client:

Command:

```
AT+BLEINIT=1
```

Response:

```
OK
```

1. ESP32 Bluetooth LE server creates services.

Command:

```
AT+BLEGATTSSRVCRE
```

Response:

```
OK
```

2. ESP32 Bluetooth LE server starts services.

Command:

```
AT+BLEGATTSSRVSTART
```

Response:

```
OK
```

1. ESP32 Bluetooth LE server gets its MAC address.

Command:

```
AT+BLEADDR?
```

Response:

```
+BLEADDR: "24:0a:c4:d6:e4:46"  
OK
```

Note:

- The address you obtain may be different from that in the above response. Keep yours handy as you will need it in one of the following steps.

2. ESP32 Bluetooth LE server sets advertising parameters.

Command:

```
AT+BLEADVPARAM=50,50,0,0,7,0,,
```

Response:


```
OK
```

3. ESP32 Bluetooth LE server sets advertising data.

Command:

```
AT+BLEADVDATA="0201060A09457370726573736966030302A0"
```

Response:

```
OK
```

4. ESP32 Bluetooth LE server starts advertising.

Command:

```
AT+BLEADVSTART
```

Response:

```
OK
```

1. ESP32 Bluetooth LE client creates services.

Command:

```
AT+BLEGATTSSRVCRE
```

Response:

```
OK
```

2. ESP32 Bluetooth LE client starts services.

Command:

```
AT+BLEGATTSSRVSTART
```

Response:

```
OK
```

1. ESP32 Bluetooth LE client gets Bluetooth LE address.

Command:

```
AT+BLEADDR?
```

Response:

```
+BLEADDR: "24:0a:c4:03:a7:4e"  
OK
```

Note:

- The address you obtain may be different from that in the above response. Keep yours handy as you will need it in one of the following steps.

2. ESP32 Bluetooth LE client enables a scanning for three seconds.

Command:

```
AT+BLESCAN=1,3
```

Response:

```
OK  
+BLESCAN: "5b:3b:6c:51:90:49",-87,02011a020a0c0aff4c001005071c3024dc,,1  
+BLESCAN: "c4:5b:be:93:ec:66",-84,0201060303111809095647543147572d58020a03,,0  
+BLESCAN: "24:0a:c4:d6:e4:46",-29,,,0
```

Note:

- The scan results you obtain may be different from those in the above response.

3. Establish the Bluetooth LE connection.

ESP32 Bluetooth LE client:

Command:

```
AT+BLECONN=0, "24:0a:c4:d6:e4:46"
```

Response:

```
+BLECONN:0, "24:0a:c4:d6:e4:46"
```

OK

Note:

- When entering the above command, replace the address with your ESP32 Bluetooth LE server address.
- If the Bluetooth LE connection is established successfully, message +BLECONN:0, "24:0a:c4:d6:e4:46" will be prompted.
- If the Bluetooth LE connection is broken, message +BLECONN:0, -1 will be prompted.

ESP32 Bluetooth LE server:

Command:

```
AT+BLECONN=0, "24:0a:c4:03:a7:4e"
```

Response:

```
+BLECONN:0, "24:0a:c4:03:a7:4e"
```

OK

Note:

- When entering the above command, replace the address with your ESP32 Bluetooth LE server address.
- If the Bluetooth LE connection is established successfully, the message OK will be prompted and the message +BLECONN:0, "24:0a:c4:03:a7:4e" will not be prompted.
- If the Bluetooth LE connection is broken, the message ERROR will be prompted and the message +BLECONN:0, -1 will not be prompted.

4. ESP32 Bluetooth LE client discovers local services.

Command:

```
AT+BLEGATTSSRV?
```

Response:

```
+BLEGATTSSRV:1,1,0xA002,1
```

```
+BLEGATTSSRV:2,1,0xA003,1
```

OK

5. ESP32 Bluetooth LE client discovers local characteristics.

Command:

```
AT+BLEGATTCHAR?
```

Response:

```
+BLEGATTCHAR:"char",1,1,0xC300,0x02
```

```
+BLEGATTCHAR:"desc",1,1,1,0x2901
```

```
+BLEGATTCHAR:"char",1,2,0xC301,0x02
```

```
+BLEGATTCHAR:"desc",1,2,1,0x2901
```

```
+BLEGATTCHAR:"char",1,3,0xC302,0x08
```

```
+BLEGATTCHAR:"desc",1,3,1,0x2901
```

```
+BLEGATTCHAR:"char",1,4,0xC303,0x04
```

```
+BLEGATTCHAR:"desc",1,4,1,0x2901
```

```
+BLEGATTCHAR:"char",1,5,0xC304,0x08
```

```
+BLEGATTCHAR:"char",1,6,0xC305,0x10
```

```
+BLEGATTCHAR:"desc",1,6,1,0x2902
```

```
+BLEGATTCHAR:"char",1,7,0xC306,0x20
```

```
+BLEGATTCHAR:"desc",1,7,1,0x2902
```

```
+BLEGATTCHAR:"char",1,8,0xC307,0x02
```

(continues on next page)

(continued from previous page)

```
+BLEGATTCHAR:"desc",1,8,1,0x2901
+BLEGATTCHAR:"char",2,1,0xC400,0x02
+BLEGATTCHAR:"desc",2,1,1,0x2901
+BLEGATTCHAR:"char",2,2,0xC401,0x02
+BLEGATTCHAR:"desc",2,2,1,0x2901
```

OK

6. ESP32 Bluetooth LE server discovers primary services.

Command:

```
AT+BLEGATTCPRIMSRV=0
```

Response:

```
+BLEGATTCPRIMSRV:0,1,0x1801,1
+BLEGATTCPRIMSRV:0,2,0x1800,1
+BLEGATTCPRIMSRV:0,3,0xA002,1
+BLEGATTCPRIMSRV:0,4,0xA003,1
```

OK

Note:

- When discovering services, the ESP32 Bluetooth LE server will get two more default services (UUID: 0x1800 and 0x1801) than what the ESP32 Bluetooth LE client will get. So, for the same service, the <srv_index> received by the ESP32 Bluetooth LE server equals the <srv_index> received by the ESP32 Bluetooth LE client + 2. For example, for service 0xA002, the <srv_index> queried on the ESP32 Bluetooth LE client is 3, if the ESP32 Bluetooth LE server is queried through the command [AT+BLEGATTSSRV?](#), then <srv_index> is 1.

7. ESP32 Bluetooth LE server discovers primary characteristics.

Command:

```
AT+BLEGATTCCCHAR=0,3
```

Response:

```
+BLEGATTCCCHAR:"char",0,3,1,0xC300,0x02
+BLEGATTCCCHAR:"desc",0,3,1,1,0x2901
+BLEGATTCCCHAR:"char",0,3,2,0xC301,0x02
+BLEGATTCCCHAR:"desc",0,3,2,1,0x2901
+BLEGATTCCCHAR:"char",0,3,3,0xC302,0x08
+BLEGATTCCCHAR:"desc",0,3,3,1,0x2901
+BLEGATTCCCHAR:"char",0,3,4,0xC303,0x04
+BLEGATTCCCHAR:"desc",0,3,4,1,0x2901
+BLEGATTCCCHAR:"char",0,3,5,0xC304,0x08
+BLEGATTCCCHAR:"char",0,3,6,0xC305,0x10
+BLEGATTCCCHAR:"desc",0,3,6,1,0x2902
+BLEGATTCCCHAR:"char",0,3,7,0xC306,0x20
+BLEGATTCCCHAR:"desc",0,3,7,1,0x2902
+BLEGATTCCCHAR:"char",0,3,8,0xC307,0x02
+BLEGATTCCCHAR:"desc",0,3,8,1,0x2901
```

OK

8. ESP32 Bluetooth LE client sets characteristics.

Select the service characteristic that supports the write operation (characteristic) to set the characteristic.

Command:

```
AT+BLEGATTSSSETATTR=1,8,,1
```

Response:

>

Command:

```
Write 1 byte ``9``
```

Response:

```
OK
```

9. ESP32 Bluetooth LE server reads characteristics.

Command:

```
AT+BLEGATTCD=0,3,8,
```

Response:

```
+BLEGATTCD:0,1,9
```

```
OK
```

10. ESP32 Bluetooth LE client write characteristics.

Select the service characteristic that supports the write operation to write the characteristics.

Command:

```
AT+BLEGATTCWR=0,3,6,1,2
```

Response:

```
>
```

Command:

```
Write 2 bytes ``12``
```

Response:

```
OK
```

Note:

- If the Bluetooth LE server successfully writes the service characteristic value, the Bluetooth LE client will prompt `+WRITE:0,1,6,1,2,12`.

11. ESP32 Bluetooth LE client notify characteristics.

Command:

```
AT+BLEGATTSENTFY=0,1,6,10
```

Response:

```
>
```

Command:

```
Write 10 bytes ``1234567890``
```

Response:

```
OK
```

Note:

- If the ESP32 Bluetooth LE client's notify characteristic is successfully sent to the server, the Bluetooth LE server `+NOTIFY:0,3,6,10,1234567890` will be prompt.

4.4.4 Encrypt Bluetooth LE connection

Below is an example of using two ESP32 development boards, one as a Bluetooth LE server (only as Bluetooth LE server role), the other one as a Bluetooth LE client (only as Bluetooth LE client role). The example shows how to encrypt Bluetooth LE connection.

Important:

- In the following steps, the operations starting with ESP32 Bluetooth LE server only need to be executed at ESP32 Bluetooth LE server, and the operations starting with ESP32 Bluetooth LE client only need to be executed at ESP32 Bluetooth LE client.
 - Encryption and bonding are two different concepts. bonding is just a long-term key stored locally after successful encryption.
 - ESP-AT allows a maximum of 10 devices to be bonded.
-

1. Bluetooth LE initialization.

ESP32 Bluetooth LE server:

Command:

```
AT+BLEINIT=2
```

Response:

```
OK
```

ESP32 Bluetooth LE client:

Command:

```
AT+BLEINIT=1
```

Response:

```
OK
```

2. ESP32 Bluetooth LE server gets Bluetooth LE address.

Command:

```
AT+BLEADDR?
```

Response:

```
+BLEADDR: "24:0a:c4:d6:e4:46"  
OK
```

Note:

- The address you obtain may be different from that in the above response. Keep yours handy as you will need it in one of the following steps.

1. ESP32 Bluetooth LE server creates services.

Command:

```
AT+BLEGATTSSRVCRE
```

Response:

```
OK
```

2. ESP32 Bluetooth LE server starts services.

Command:

```
AT+BLEGATTSSRVSTART
```

Response:

```
OK
```

1. ESP32 Bluetooth LE server discovers characteristics.

Command:

```
AT+BLEGATTSSCHAR?
```

Response:

```

+BLEGATTCHAR:"char",1,1,0xC300,0x02
+BLEGATTCHAR:"desc",1,1,1,0x2901
+BLEGATTCHAR:"char",1,2,0xC301,0x02
+BLEGATTCHAR:"desc",1,2,1,0x2901
+BLEGATTCHAR:"char",1,3,0xC302,0x08
+BLEGATTCHAR:"desc",1,3,1,0x2901
+BLEGATTCHAR:"char",1,4,0xC303,0x04
+BLEGATTCHAR:"desc",1,4,1,0x2901
+BLEGATTCHAR:"char",1,5,0xC304,0x08
+BLEGATTCHAR:"char",1,6,0xC305,0x10
+BLEGATTCHAR:"desc",1,6,1,0x2902
+BLEGATTCHAR:"char",1,7,0xC306,0x20
+BLEGATTCHAR:"desc",1,7,1,0x2902
+BLEGATTCHAR:"char",1,8,0xC307,0x02
+BLEGATTCHAR:"desc",1,8,1,0x2901
+BLEGATTCHAR:"char",2,1,0xC400,0x02
+BLEGATTCHAR:"desc",2,1,1,0x2901
+BLEGATTCHAR:"char",2,2,0xC401,0x02
+BLEGATTCHAR:"desc",2,2,1,0x2901

OK

```

2. ESP32 Bluetooth LE server starts advertising, then the ESP32 Bluetooth LE client starts scanning and lasts for 3 s.

ESP32 Bluetooth LE server:

Command:

```
AT+BLEADVSTART
```

Response:

```
OK
```

ESP32 Bluetooth LE client:

Command:

```
AT+BLES SCAN=1,3
```

Response:

```

OK
+BLES SCAN:"5b:3b:6c:51:90:49",-87,02011a020a0c0aff4c001005071c3024dc,,1
+BLES SCAN:"c4:5b:be:93:ec:66",-84,0201060303111809095647543147572d58020a03,,0
+BLES SCAN:"24:0a:c4:d6:e4:46",-29,,,0

```

Note:

- The scan results you obtain may be different from those in the above response.

3. Establish the Bluetooth LE connection.

ESP32 Bluetooth LE client:

Command:

```
AT+BLECONN=0,"24:0a:c4:d6:e4:46"
```

Response:

```

+BLECONN:0,"24:0a:c4:d6:e4:46"

OK

```

Note:

- When entering the above command, replace the address with your ESP32 Bluetooth LE server address.
- If the Bluetooth LE connection is established successfully, message +BLECONN:0,"24:0a:c4:d6:e4:46" will be prompted.
- If the Bluetooth LE connection is broken, message +BLECONN:0,-1 will be prompted.

4. ESP32 Bluetooth LE client discovers services.

Command:

```
AT+BLEGATTCPRIMSRV=0
```

Response:

```
+BLEGATTCPRIMSRV:0,1,0x1801,1
+BLEGATTCPRIMSRV:0,2,0x1800,1
+BLEGATTCPRIMSRV:0,3,0xA002,1
+BLEGATTCPRIMSRV:0,4,0xA003,1

OK
```

Note:

- When discovering services, the ESP32 Bluetooth LE client will get two more default services (UUID: 0x1800 and 0x1801) than what the ESP32 Bluetooth LE server will get. So, for the same service, the <srv_index> received by the ESP32 Bluetooth LE client equals the <srv_index> received by the ESP32 Bluetooth LE server + 2. For example, for service 0xA002, the <srv_index> queried on the ESP32 Bluetooth LE client is 3, if the ESP32 Bluetooth LE server is queried through the command [AT+BLEGATTSSRV?](#), then <srv_index> is 1.
5. ESP32 Bluetooth LE client discovers characteristics.

Command:

```
AT+BLEGATTCCHAR=0,3
```

Response:

```
+BLEGATTCCHAR:"char",0,3,1,0xC300,0x02
+BLEGATTCCHAR:"desc",0,3,1,1,0x2901
+BLEGATTCCHAR:"char",0,3,2,0xC301,0x02
+BLEGATTCCHAR:"desc",0,3,2,1,0x2901
+BLEGATTCCHAR:"char",0,3,3,0xC302,0x08
+BLEGATTCCHAR:"desc",0,3,3,1,0x2901
+BLEGATTCCHAR:"char",0,3,4,0xC303,0x04
+BLEGATTCCHAR:"desc",0,3,4,1,0x2901
+BLEGATTCCHAR:"char",0,3,5,0xC304,0x08
+BLEGATTCCHAR:"char",0,3,6,0xC305,0x10
+BLEGATTCCHAR:"desc",0,3,6,1,0x2902
+BLEGATTCCHAR:"char",0,3,7,0xC306,0x20
+BLEGATTCCHAR:"desc",0,3,7,1,0x2902
+BLEGATTCCHAR:"char",0,3,8,0xC307,0x02
+BLEGATTCCHAR:"desc",0,3,8,1,0x2901

OK
```

6. Set Bluetooth LE encryption parameters. Set auth_req to SC_MITM_BOND, server' s iocap to KeyboardOnly, client' s iocap to KeyboardDisplay, key_size to 16, init_key to 3, rsp_key to 3.

ESP32 Bluetooth LE server:

Command:

```
AT+BLESECPARAM=13,2,16,3,3
```

Response:

```
OK
```

ESP32 Bluetooth LE client:

Command:

```
AT+BLESECPARAM=13,4,16,3,3
```

Response:

```
OK
```

Note:

- In this example, ESP32 Bluetooth LE server enters the pairing code and ESP32 Bluetooth LE client displays the pairing code.
- ESP-AT supports Legacy Pairing and Secure Connections encryption methods, but the latter has a higher priority. If the peer also supports Secure Connections, then Secure Connections will be used for encryption.

7. ESP32 Bluetooth LE client initiates encryption request.

Command:

```
AT+BLEENC=0,3
```

Response:

```
OK
```

Note:

If the ESP32 Bluetooth LE server successfully receives the encryption request, message +BLESECREQ:0 will be prompted on the ESP32 Bluetooth LE server side.

1. ESP32 Bluetooth LE server responds to the pairing request.

Command:

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

Response:

```
OK
```

Note:

- If the ESP32 Bluetooth LE client successfully receives the pairing response, a 6-digit pairing code will generate on the ESP32 Bluetooth LE client side.
- In this example, message +BLESECNTFYKEY:0,793718 will be prompted on the ESP32 Bluetooth LE client side. Pairing code is 793718.

1. ESP32 Bluetooth LE server replies pairing code.

Command:

```
AT+BLEKEYREPLY=0,793718
```

Response:

```
OK
```

After running this command, there will be corresponding messages prompt on both the ESP32 Bluetooth LE server and the ESP32 Bluetooth LE client.

ESP32 Bluetooth LE server:

```
+BLESECKEYTYPE:0,16
+BLESECKEYTYPE:0,1
+BLESECKEYTYPE:0,32
+BLESECKEYTYPE:0,2
+BLEAUTHCMPL:0,0
```

ESP32 Bluetooth LE client:

```
+BLESECNTFYKEY:0,793718
+BLESECKEYTYPE:0,2
+BLESECKEYTYPE:0,16
+BLESECKEYTYPE:0,1
+BLESECKEYTYPE:0,32
+BLEAUTHCMPL:0,0
```


You can ignore the message starting with +BLESECKEYTYPE. In terms of the second parameter in the message +BLEAUTHCMPL:0,0,0 means encryption is successful, and 1 means encryption fails.

4.4.5 Establish SPP connection between two ESP32 development boards and transmit data in UART-Bluetooth LE Passthrough Mode

Below is an example of using two ESP32 development boards, one as a Bluetooth LE server (only as Bluetooth LE server role), the other one as a Bluetooth LE client (only as Bluetooth LE client role). The example shows how to build Bluetooth LE SPP (Serial Port Profile, UART-Bluetooth LE passthrough mode) with AT commands.

Important: In the following steps, the operations starting with ESP32 Bluetooth LE server only need to be executed at ESP32 Bluetooth LE server, and the operations starting with ESP32 Bluetooth LE client only need to be executed at ESP32 Bluetooth LE client.

1. Bluetooth LE initialization.

ESP32 Bluetooth LE server:

Command:

```
AT+BLEINIT=2
```

Response:

```
OK
```

ESP32 Bluetooth LE client:

Command:

```
AT+BLEINIT=1
```

Response:

```
OK
```

1. ESP32 Bluetooth LE server creates services.

Command:

```
AT+BLEGATTSSRVCRE
```

Response:

```
OK
```

2. ESP32 Bluetooth LE server starts services.

Command:

```
AT+BLEGATTSSRVSTART
```

Response:

```
OK
```

1. ESP32 Bluetooth LE server gets Bluetooth LE address.

Command:

```
AT+BLEADDR?
```

Response:

```
+BLEADDR: "24:0a:c4:d6:e4:46"
```

```
OK
```

Note:

- The address you obtain may be different from that in the above response. Keep yours handy as you will need it in one of the following steps.
2. ESP32 Bluetooth LE server sets advertising parameters.

Command:

```
AT+BLEADVPARAM=50,50,0,0,7,0,,
```

Response:

```
OK
```

3. ESP32 Bluetooth LE server sets advertising data.

Command:

```
AT+BLEADVDATA="0201060A09457370726573736966030302A0"
```

Response:

```
OK
```

4. ESP32 Bluetooth LE server starts advertising.

Command:

```
AT+BLEADVSTART
```

Response:

```
OK
```

5. ESP32 Bluetooth LE client enables a scanning for three seconds.

Command:

```
AT+BLES SCAN=1,3
```

Response:

```
OK
+BLES SCAN:"24:0a:c4:d6:e4:46",-78,0201060a09457370726573736966030302a0,,0
+BLES SCAN:"45:03:cb:ac:aa:a0",-62,0201060aff4c001005441c61df7d,,1
+BLES SCAN:"24:0a:c4:d6:e4:46",-26,0201060a09457370726573736966030302a0,,0
```

Note:

- The scan results you obtain may be different from those in the above response.

6. Establish the Bluetooth LE connection.

ESP32 Bluetooth LE client:

Command:

```
AT+BLECONN=0,"24:0a:c4:d6:e4:46"
```

Response:

```
+BLECONN:0,"24:0a:c4:d6:e4:46"
```

```
OK
```

Note:

- When entering the above command, replace the address your ESP32 Bluetooth LE server address.
- If the Bluetooth LE connection is established successfully, message +BLECONN:0,"24:0a:c4:d6:e4:46" will be prompted.
- If the Bluetooth LE connection is broken, message +BLECONN:0,-1 will be prompted.

7. ESP32 Bluetooth LE server discovers local services.

Command:

```
AT+BLEGATTSSRV?
```

Response:

```
+BLEGATTSSRV:1,1,0xA002,1
+BLEGATTSSRV:2,1,0xA003,1
```

OK

8. ESP32 Bluetooth LE server discovers local characteristics.

Command:

```
AT+BLEGATTSSCHAR?
```

Response:

```
+BLEGATTSSCHAR:"char",1,1,0xC300,0x02
+BLEGATTSSCHAR:"desc",1,1,1,0x2901
+BLEGATTSSCHAR:"char",1,2,0xC301,0x02
+BLEGATTSSCHAR:"desc",1,2,1,0x2901
+BLEGATTSSCHAR:"char",1,3,0xC302,0x08
+BLEGATTSSCHAR:"desc",1,3,1,0x2901
+BLEGATTSSCHAR:"char",1,4,0xC303,0x04
+BLEGATTSSCHAR:"desc",1,4,1,0x2901
+BLEGATTSSCHAR:"char",1,5,0xC304,0x08
+BLEGATTSSCHAR:"char",1,6,0xC305,0x10
+BLEGATTSSCHAR:"desc",1,6,1,0x2902
+BLEGATTSSCHAR:"char",1,7,0xC306,0x20
+BLEGATTSSCHAR:"desc",1,7,1,0x2902
+BLEGATTSSCHAR:"char",1,8,0xC307,0x02
+BLEGATTSSCHAR:"desc",1,8,1,0x2901
+BLEGATTSSCHAR:"char",2,1,0xC400,0x02
+BLEGATTSSCHAR:"desc",2,1,1,0x2901
+BLEGATTSSCHAR:"char",2,2,0xC401,0x02
+BLEGATTSSCHAR:"desc",2,2,1,0x2901
```

OK

9. ESP32 Bluetooth LE client discovers services.

Command:

```
AT+BLEGATTCPRIMSRV=0
```

Response:

```
+BLEGATTCPRIMSRV:0,1,0x1801,1
+BLEGATTCPRIMSRV:0,2,0x1800,1
+BLEGATTCPRIMSRV:0,3,0xA002,1
+BLEGATTCPRIMSRV:0,4,0xA003,1
```

OK

Note:

- When discovering services, the ESP32 Bluetooth LE client will get two more default services (UUID: 0x1800 and 0x1801) than what the ESP32 Bluetooth LE server will get. So, for the same service, the <srv_index> received by the ESP32 Bluetooth LE client equals the <srv_index> received by the ESP32 Bluetooth LE server + 2. For example, for service 0xA002, the <srv_index> queried on the ESP32 Bluetooth LE client is 3, if the ESP32 Bluetooth LE server is queried through the command [AT+BLEGATTSSRV?](#), then <srv_index> is 1.

10. ESP32 Bluetooth LE client discovers characteristics.

Command:

```
AT+BLEGATTCCHAR=0,3
```

Response:

```
+BLEGATTCCHAR:"char",0,3,1,0xC300,0x02
+BLEGATTCCHAR:"desc",0,3,1,1,0x2901
```

(continues on next page)

(continued from previous page)

```
+BLEGATTCCCHAR:"char",0,3,2,0xC301,0x02
+BLEGATTCCCHAR:"desc",0,3,2,1,0x2901
+BLEGATTCCCHAR:"char",0,3,3,0xC302,0x08
+BLEGATTCCCHAR:"desc",0,3,3,1,0x2901
+BLEGATTCCCHAR:"char",0,3,4,0xC303,0x04
+BLEGATTCCCHAR:"desc",0,3,4,1,0x2901
+BLEGATTCCCHAR:"char",0,3,5,0xC304,0x08
+BLEGATTCCCHAR:"char",0,3,6,0xC305,0x10
+BLEGATTCCCHAR:"desc",0,3,6,1,0x2902
+BLEGATTCCCHAR:"char",0,3,7,0xC306,0x20
+BLEGATTCCCHAR:"desc",0,3,7,1,0x2902
+BLEGATTCCCHAR:"char",0,3,8,0xC307,0x02
+BLEGATTCCCHAR:"desc",0,3,8,1,0x2901
```

OK

11. ESP32 Bluetooth LE client Configures Bluetooth LE SPP.

Set a characteristic that enables writing permission to TX channel for sending data. Set another characteristic that supports notification or indication to RX channel for receiving data.

Command:

```
AT+BLESPPCFG=1,3,5,3,7
```

Response:

OK

12. ESP32 Bluetooth LE client enables Bluetooth LE SPP.

Command:

```
AT+BLESPP
```

Response:

OK

>

This response indicates that AT has entered Bluetooth LE SPP mode and can send and receive data.

Note:

- After the ESP32 Bluetooth LE client enables Bluetooth LE SPP, data received from serial port will be transmitted to the Bluetooth LE server directly.

13. ESP32 Bluetooth LE server Configures Bluetooth LE SPP.

Set a characteristic that supports notification or indication to TX channel for sending data. Set another characteristic that enables writing permission to RX channel for receiving data.

Command:

```
AT+BLESPPCFG=1,1,7,1,5
```

Response:

OK

14. ESP32 Bluetooth LE server enables Bluetooth LE SPP.

Command:

```
AT+BLESPP
```

Response:

OK

>

This response indicates that AT has entered Bluetooth LE SPP mode and can send and receive data.

Note:

- After the ESP32 Bluetooth LE server enables Bluetooth LE SPP, the data received from serial port will be transmitted to the Bluetooth LE client directly.
- If the ESP32 Bluetooth LE client does not enable Bluetooth LE SPP first, or uses other device as Bluetooth LE client, then the Bluetooth LE client needs to listen to the notification or indication first. For example, if the ESP32 Bluetooth LE client does not enable Bluetooth LE SPP first, then it should use command `AT+BLEGATTCWR=0,3,7,1,1` to enable listening function first, so that the ESP32 Bluetooth LE server can transmit successfully.
- For the same service, the `<srv_index>` on the ESP32 Bluetooth LE client side equals the `<srv_index>` on the ESP32 Bluetooth LE server side + 2.

4.4.6 Establish SPP connection between ESP32 and mobile phone and transmit data in UART-Bluetooth LE passthrough mode

The example shows how to establish SPP connection between an ESP32 development board (only serving as the Bluetooth LE server role) and a mobile phone (only serve as the Bluetooth LE client role) and how to transmit data between them in UART-Bluetooth LE passthrough mode.

Important: In the following steps, the operations starting with ESP32 Bluetooth LE server only need to be executed at ESP32 Bluetooth LE server, and those starting with ESP32 Bluetooth LE client only need to be executed on the Bluetooth debugging assistant of the mobile phone.

1. First, you need to download the Bluetooth LE debugging assistant on the mobile phone, such as LightBlue.
2. Bluetooth LE initialization.

ESP32 Bluetooth LE server:

Command:

```
AT+BLEINIT=2
```

Response:

```
OK
```

1. ESP32 Bluetooth LE server creates services.

Command:

```
AT+BLEGATTSSRVCRE
```

Response:

```
OK
```

2. ESP32 Bluetooth LE server starts services.

Command:

```
AT+BLEGATTSSRVSTART
```

Response:

```
OK
```

1. ESP32 Bluetooth LE server gets its MAC address.

Command:

```
AT+BLEADDR?
```

Response:

```
+BLEADDR: "24:0a:c4:d6:e4:46"  
OK
```

Note:

- The address you obtain may be different from that in the above response. Keep yours handy as you will need it in one of the following steps.
2. ESP32 Bluetooth LE server sets advertising parameters.

Command:

```
AT+BLEADVPARAM=50,50,0,0,7,0,,
```

Response:

```
OK
```

3. ESP32 Bluetooth LE server sets advertising data.

Command:

```
AT+BLEADVDATA="0201060A09457370726573736966030302A0"
```

Response:

```
OK
```

4. ESP32 Bluetooth LE server starts advertising.

Command:

```
AT+BLEADVSTART
```

Response:

```
OK
```

5. Establish the Bluetooth LE connection.

Open the LightBlue application on your mobile phone, and click **SCAN** to start scanning. When you find the MAC address of the ESP32 Bluetooth LE server, click **CONNECT**. Then, ESP32 should print the log similar to `+BLECONN:0,"60:51:42:fe:98:aa"`, which indicates that Bluetooth LE connection has been established.

6. ESP32 Bluetooth LE server discovers local services.

Command:

```
AT+BLEGATTSSRV?
```

Response:

```
+BLEGATTSSRV:1,1,0xA002,1
+BLEGATTSSRV:2,1,0xA003,1
```

```
OK
```

7. ESP32 Bluetooth LE server discovers local characteristics.

Command:

```
AT+BLEGATTSSCHAR?
```

Response:

```
+BLEGATTSSCHAR:"char",1,1,0xC300,0x02
+BLEGATTSSCHAR:"desc",1,1,1,0x2901
+BLEGATTSSCHAR:"char",1,2,0xC301,0x02
+BLEGATTSSCHAR:"desc",1,2,1,0x2901
+BLEGATTSSCHAR:"char",1,3,0xC302,0x08
+BLEGATTSSCHAR:"desc",1,3,1,0x2901
+BLEGATTSSCHAR:"char",1,4,0xC303,0x04
+BLEGATTSSCHAR:"desc",1,4,1,0x2901
+BLEGATTSSCHAR:"char",1,5,0xC304,0x08
+BLEGATTSSCHAR:"char",1,6,0xC305,0x10
+BLEGATTSSCHAR:"desc",1,6,1,0x2902
+BLEGATTSSCHAR:"char",1,7,0xC306,0x20
+BLEGATTSSCHAR:"desc",1,7,1,0x2902
```

(continues on next page)

(continued from previous page)

```
+BLEGATTCHAR:"char",1,8,0xC307,0x02
+BLEGATTCHAR:"desc",1,8,1,0x2901
+BLEGATTCHAR:"char",2,1,0xC400,0x02
+BLEGATTCHAR:"desc",2,1,1,0x2901
+BLEGATTCHAR:"char",2,2,0xC401,0x02
+BLEGATTCHAR:"desc",2,2,1,0x2901
```

OK

8. ESP32 Bluetooth LE client discovers characteristics.

Click the button of the service feature whose Properties is NOTIFY or INDICATE in the mobile phone LightBlue client (here ESP-AT default Properties The service characteristics of NOTIFY or INDICATE are 0xC305 and 0xC306) and start to listen for the service characteristics of NOTIFY or INDICATE.

9. ESP32 Bluetooth LE server configures Bluetooth LE SPP.

Set a characteristic that supports notification or indication to TX channel for sending data. Set another characteristic that enables writing permission to RX channel for receiving data.

Command:

```
AT+BLESPPCFG=1,1,7,1,5
```

Response:

OK

10. ESP32 Bluetooth LE server enables Bluetooth LE SPP.

Command:

```
AT+BLESPP
```

Response:

OK

>

This response indicates that AT has entered Bluetooth LE SPP mode and can send and receive data.

11. Bluetooth LE client sends data.

In the LightBlue client, select the 0xC304 service characteristic value and send the data `test` to the ESP32 Bluetooth LE server. Then, the ESP32 Bluetooth LE server can receive the `test`.

12. ESP32 Bluetooth LE server sends data.

The ESP32 Bluetooth LE server sends `test`, and then the LightBlue client can receive `test`.

4.4.7 Establish Bluetooth LE connection and pairing between ESP32 and mobile phone

This example shows how to establish a Bluetooth LE connection between the ESP32 development board (only as a Bluetooth LE server role) and a mobile phone (only as a Bluetooth LE client role) and enter the passkey to complete the pairing.

Important: In the following steps, the operations starting with `ESP32 Bluetooth LE server` only need to be executed at ESP32 Bluetooth LE server, and those starting with `ESP32 Bluetooth LE client` only need to be executed on the Bluetooth debugging assistant of the mobile phone.

1. First, you need to download the Bluetooth LE debugging assistant on the mobile phone, such as LightBlue.
2. Bluetooth LE initialization.

ESP32 Bluetooth LE server:

Command:

```
AT+BLEINIT=2
```

Response:

```
OK
```

1. ESP32 Bluetooth LE server creates services.

Command:

```
AT+BLEGATTSSRVCRE
```

Response:

```
OK
```

2. ESP32 Bluetooth LE server starts services.

Command:

```
AT+BLEGATTSSRVSTART
```

Response:

```
OK
```

1. ESP32 Bluetooth LE server gets its MAC address.

Command:

```
AT+BLEADDR?
```

Response:

```
+BLEADDR: "24:0a:c4:d6:e4:46"
```

```
OK
```

Note:

- The address you obtain may be different from that in the above response. Keep yours handy as you will need it in one of the following steps.

2. ESP32 Bluetooth LE server sets advertising parameters.

Command:

```
AT+BLEADVPARAM=50,50,0,0,7,0,,
```

Response:

```
OK
```

3. ESP32 Bluetooth LE server sets advertising data.

Command:

```
AT+BLEADVDATA="0201060A09457370726573736966030302A0"
```

Response:

```
OK
```

4. ESP32 Bluetooth LE server sets encryption parameters.

Command:

```
AT+BLESECPARAM=13,2,16,3,3
```

Response:

```
OK
```

5. ESP32 Bluetooth LE server starts advertising.

Command:

```
AT+BLEADVSTART
```

Response:

OK

6. Establish the Bluetooth LE connection.

Open the LightBlue application on your mobile phone, and click `SCAN` to start scanning. When you find the MAC address of the ESP32 Bluetooth LE server, click `CONNECT`. Then, ESP32 should print the log similar to `+BLECONN:0,"60:51:42:fe:98:aa"`, which indicates that Bluetooth LE connection has been established.

7. ESP32 Bluetooth LE server initiates encryption request.

Command:

AT+BLEENC=0,3

Response:

OK

8. Bluetooth LE client accepts pairing.

The pairing information (including the pairing key, for example: 231518) will pop up on the Bluetooth LE connection page that has just been successfully created on the LightBlue, and then click “Pairing”. At this time, the ESP32 Bluetooth LE server should print a log similar to `+BLESECKEYREQ:0`, indicating that the phone has responded to pairing, and the ESP32 Bluetooth LE server can enter the key.

9. ESP32 Bluetooth LE server replies with the pairing key.

At this time, the key that the Bluetooth LE server should reply with is the key (231518) contained in the pairing information that popped up on LightBlue in the previous step.

Command:

AT+BLEKEYREPLY=0,231518

Response:

OK

At this time, the ESP32 Bluetooth LE server will print a log similar to the following, which means that the ESP32 Bluetooth LE server has successfully paired with the phone Bluetooth LE client.

+BLESECKEYTYPE:0,16
+BLESECKEYTYPE:0,1
+BLESECKEYTYPE:0,32
+BLESECKEYTYPE:0,2
+BLEAUTHCMPL:0,0

4.5 MQTT AT Examples

This document provides detailed command examples to illustrate how to utilize *MQTT AT Commands* on ESP32.

- *MQTT over TCP (with a local MQTT broker)(suitable for a small amount of data)*
 - *MQTT over TCP (with a local MQTT broker)(suitable for large amounts of data)*
 - *MQTT over TLS (with a local MQTT broker)*
 - *MQTT over WSS*

Important:

- The examples described in this document are based on the situation that Wi-Fi has been connected.
-

4.5.1 MQTT over TCP (with a local MQTT broker)(suitable for a small amount of data)

Below is an example of using two ESP32 development boards, one as a MQTT publisher (only as MQTT publisher role), the other one as a MQTT subscriber (only as MQTT subscriber role).

The example shows how to establish MQTT connections over TCP. You need to first create a local MQTT broker. For example, the MQTT broker's IP address is 192.168.3.102, and the port is 8883.

Important: In the step, the operations starting with ESP32 MQTT publisher only need to be executed at ESP32 MQTT publisher, and the operations starting with ESP32 MQTT subscriber only need to be executed at ESP32 MQTT subscriber. If the operation is not specified on which side it is executed, it needs to be executed on both the publisher side and the subscriber side.

1. Set MQTT user configuration.

ESP32 MQTT publisher:

Command:

```
AT+MQTTUSERCFG=0,1,"publisher","espressif","123456789",0,0,""
```

Response:

```
OK
```

ESP32 MQTT subscriber:

Command:

```
AT+MQTTUSERCFG=0,1,"subscriber","espressif","123456789",0,0,""
```

Response:

```
OK
```

2. Connect to MQTT brokers.

Command:

```
AT+MQTTCONN=0,"192.168.3.102",8883,1
```

Response:

```
+MQTTCONNECTED:0,1,"192.168.3.102","8883","",1
```

```
OK
```

Note:

- The MQTT broker domain or MQTT broker IP address you enter may be different from those in the above command.

3. Subscribe to MQTT topics.

ESP32 MQTT subscriber:

Command:

```
AT+MQTTSUB=0,"topic",1
```

Response:

```
OK
```

4. Publish MQTT messages in string.

ESP32 MQTT publisher:

Command:

```
AT+MQTTPUB=0,"topic","test",1,0
```

Response:

```
OK
```

Note:

- If the ESP32 MQTT publisher successfully publishes the message, following message will be prompted on the ESP32 MQTT subscriber.

```
+MQTTSUBRECV:0,"topic",4,test
```

5. Close MQTT connections.

Command:

```
AT+MQTTCLEAN=0
```

Response:

```
OK
```

4.5.2 MQTT over TCP (with a local MQTT broker)(suitable for large amounts of data)

Below is an example of using two ESP32 development boards, one as a MQTT publisher (only as MQTT publisher role), the other one as a MQTT subscriber (only as MQTT subscriber role).

The example shows how to establish MQTT connections over TCP. You need to first create a local MQTT broker. For example, the MQTT broker's IP address is 192.168.3.102, and the port is 8883.

If the amount of data you publish is relatively large, and the length of a single AT command has exceeded the threshold of 256, it is recommended that you use the [AT+MQTTPUBRAW](#) command.

Important: In the step, the operations starting with ESP32 MQTT publisher only need to be executed at ESP32 MQTT publisher, and the operations starting with ESP32 MQTT subscriber only need to be executed at ESP32 MQTT subscriber. If the operation is not specified on which side it is executed, it needs to be executed on both the publisher side and the subscriber side.

1. Set MQTT user configuration.

ESP32 MQTT publisher:

Command:

```
AT+MQTTUSERCFG=0,1,"publisher","espressif","123456789",0,0,""
```

Response:

```
OK
```

ESP32 MQTT subscriber:

Command:

```
AT+MQTTUSERCFG=0,1,"subscriber","espressif","123456789",0,0,""
```

Response:

```
OK
```

2. Connect to MQTT brokers.

Command:

```
AT+MQTTCONN=0,"192.168.3.102",8883,1
```

Response:

```
+MQTTCONNECTED:0,1,"192.168.3.102","8883","",1
```

```
OK
```

Note:

- The MQTT broker domain or MQTT broker IP address you enter may be different from those in the above command.

3. Subscribe to MQTT topics.

ESP32 MQTT subscriber:

Command:

```
AT+MQTTSUB=0,"topic",1
```

Response:

```
OK
```

4. Publish MQTT messages in string.

Assume the data you want to publish is as follows, length is 427 bytes.

```
{ "headers": { "Accept": "application/json", "Accept-Encoding": "gzip, deflate",
  ↳ "Accept-Language": "en-US,en;q=0.9,zh-CN;q=0.8,zh;q=0.7", "Content-Length": "0
  ↳ ", "Host": "httpbin.org", "Origin": "http://httpbin.org", "Referer": "http://
  ↳ httpbin.org/", "User-Agent": "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.
  ↳ 36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36", "X-Amzn-Trace-Id
  ↳ ": "Root=1-6150581e-1ad4bd5254b4bf5218070413" }}
```

ESP32 MQTT publisher:

Command:

```
AT+MQTTPUBRAW=0,"topic",427,0,0
```

Response:

```
OK
```

```
>
```

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

```
+MQTTPUB:OK
```

Note:

- After AT outputs the > character, the special characters in the data does not need to be escaped through the escape character, and it does not need to end with a new line(CR-LF).
- If the ESP32 MQTT publisher successfully publishes the message, following message will be prompted on the ESP32 MQTT subscriber.

```
+MQTTSUBRECV:0,"topic",427,{ "headers": { "Accept": "application/json",
  ↳ "Accept-Encoding": "gzip, deflate", "Accept-Language": "en-US,en;q=0.9,zh-
  ↳ CN;q=0.8,zh;q=0.7", "Content-Length": "0", "Host": "httpbin.org", "Origin":
  ↳ "http://httpbin.org", "Referer": "http://httpbin.org/", "User-Agent":
  ↳ "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
  ↳ Chrome/91.0.4472.114 Safari/537.36", "X-Amzn-Trace-Id": "Root=1-6150581e-
  ↳ 1ad4bd5254b4bf5218070413" }}
```

5. Close MQTT connections.

Command:

```
AT+MQTTCLEAN=0
```

Response:

```
OK
```

4.5.3 MQTT over TLS (with a local MQTT broker)

Below is an example of using two ESP32 development boards, one as a MQTT publisher (only as MQTT publisher role), the other one as a MQTT subscriber (only as MQTT subscriber role).

The example shows how to establish MQTT connections over TLS. You need to first create a local MQTT broker. For example, the MQTT broker's IP address is 192.168.3.102, and port is 8883.

Important: In the step, the operations starting with `ESP32 MQTT publisher` only need to be executed at ESP32 MQTT publisher, and the operations starting with `ESP32 MQTT subscriber` only need to be executed at ESP32 MQTT subscriber. If the operation is not specified on which side it is executed, it needs to be executed on both the publisher side and the subscriber side.

1. Set the time zone and the SNTP server.

Command:

```
AT+CIPSNTPCFG=1,8,"ntp1.aliyun.com"
```

Response:

```
OK
```

2. Query the SNTP time.

Command:

```
AT+CIPSNTPTIME?
```

Response:

```
+CIPSNTPTIME:Thu Sep 2 18:57:03 2021
OK
```

Note:

- The time you obtained may be different from that in the above response.
- Please make sure that the SNTP time must be a real and valid time and cannot be the time in 1970 or before.
- The purpose of setting the time is to verify the validity period of the certificates during TLS authentication.

3. Set MQTT user configuration.

ESP32 MQTT publisher:

Command:

```
AT+MQTTUSERCFG=0,4,"publisher","espressif","123456789",0,0,""
```

Response:

```
OK
```

ESP32 MQTT subscriber:

Command:

```
AT+MQTTUSERCFG=0,4,"subscriber","espressif","123456789",0,0,""
```

Response:

```
OK
```

4. Set configuration of MQTT connection.

Command:

```
AT+MQTTCONNCFG=0,0,0,"lwtt","lwtm",0,0
```

Response:

```
OK
```

5. Connect to MQTT brokers.

Command:

```
AT+MQTTCONN=0,"192.168.3.102",8883,1
```

Response:

```
+MQTTCONNECTED:0,4,"192.168.3.102","8883","",1
```

OK

Note:

- The MQTT broker domain or MQTT broker IP address you enter may be different from those in the above command.

6. Subscribe to MQTT topics.

ESP32 MQTT subscriber:

Command:

```
AT+MQTTSUB=0,"topic",1
```

Response:

OK

7. Publish MQTT messages in string.

ESP32 MQTT publisher:

Command:

```
AT+MQTTPUB=0,"topic","test",1,0
```

Response:

OK

Note:

- If the ESP32 MQTT publisher successfully publishes the message, following message will be prompted on the ESP32 MQTT subscriber.

```
+MQTTSUBRECV:0,"topic",4,test
```

8. Close MQTT connections.

Command:

```
AT+MQTTCLEAN=0
```

Response:

OK

4.5.4 MQTT over WSS

Below is an example of using two ESP32 development boards, one as a MQTT publisher (only as MQTT publisher role), the other one as a MQTT subscriber (only as MQTT subscriber role).

The example shows how to establish MQTT connections over WSS and how to communicate with a MQTT broker. For example, the MQTT broker's domain name is `test.mosquitto.org`, the path is `mqtt`, and the port is 8081.

Important: In the step, the operations starting with `ESP32 MQTT publisher` only need to be executed at ESP32 MQTT publisher, and the operations starting with `ESP32 MQTT subscriber` only need to be executed at ESP32 MQTT subscriber. If the operation is not specified on which side it is executed, it needs to be executed on both the publisher side and the subscriber side.

1. Set the time zone and the SNTP server.

Command:

```
AT+CIPSNTPCFG=1,8,"ntp1.aliyun.com"
```

Response:

```
OK
```

2. Query the SNTP time.

Command:

```
AT+CIPSNTPTIME?
```

Response:

```
+CIPSNTPTIME:Thu Sep  2 18:57:03 2021
OK
```

Note:

- The time you obtained may be different from that in the above response.
 - Please make sure that the SNTP time must be a real and valid time and cannot be the time in 1970 or before.
 - The purpose of setting the time is to verify the validity period of the certificates during TLS authentication.
3. Set MQTT user configuration.

ESP32 MQTT publisher:

Command:

```
AT+MQTTUSERCFG=0,7,"publisher","",0,0,"mqtt"
```

Response:

```
OK
```

ESP32 MQTT subscriber:

Command:

```
AT+MQTTUSERCFG=0,7,"subscriber","",0,0,"mqtt"
```

Response:

```
OK
```

4. Connect to MQTT brokers.

Command:

```
AT+MQTTCONN=0,"test.mosquitto.org",8081,1
```

Response:

```
+MQTTCONNECTED:0,7,"test.mosquitto.org","8081","/mqtt",1
OK
```

Note:

- The MQTT broker domain or MQTT broker IP address you enter may be different from those in the above command.
5. Subscribe to MQTT topics.

ESP32 MQTT subscriber:

Command:

```
AT+MQTTSUB=0,"topic",1
```

Response:

```
OK
```

6. Publish MQTT messages in string.

ESP32 MQTT publisher:

Command:

```
AT+MQTTPUB=0,"topic","test",1,0
```

Response:

```
OK
```

Note:

- If the ESP32 MQTT publisher successfully publishes the message, following message will be prompted on the ESP32 MQTT subscriber.

```
+MQTTSUBRECV:0,"topic",4,test
```

7. Close MQTT connections.

Command:

```
AT+MQTTCLEAN=0
```

Response:

```
OK
```

4.6 MQTT AT Examples for Cloud

This document mainly describes how to connect your ESP32 to AWS IoT with MQTT AT commands.

Important: For details on how to use MQTT AT commands, please refer to [MQTT AT Commands](#). You need to become familiar with the AWS IoT by reading the [AWS IoT Development Guide](#).

Please follow the steps below to connect your ESP32 to AWS IoT with ESP-AT.

- [Obtain certificates and endpoints from AWS IoT](#)
- [Connect to AWS IoT based on mutual authentication with MQTT AT commands](#)

4.6.1 Obtain certificates and endpoints from AWS IoT

1. Sign in to your AWS IoT Console account and switch to the IoT Core services.
2. Create an AWS IoT policy, thing, and certificates following the instructions in [Create AWS IoT Resources](#).

Make sure you have got the following certificate and key files:

- device.pem.crt (Device certificate)
 - private.pem.key (Private key)
 - Amazon-root-CA-1.pem (Root CA certificate)
3. Get the endpoint and bind the thing to the policy through the certificate according to the documentation [Set up the policy](#).

The endpoint value has the format of `xxx-ats.iot.us-east-2.amazonaws.com`.

Note: It is strongly recommended to familiarize yourself with the [AWS IoT Developer Guide](#). Below are some key points from this Guide that are worth noting.

- All devices must have a device certificate, private key, and root CA certificate installed in order to communicate with AWS IoT.

- Information on how to activate certificates.
 - Select Ohio as your region.
-

4.6.2 Connect to AWS IoT based on mutual authentication with MQTT AT commands

Configure certificates

There are three ways to replace MQTT certificates:

Method 1: Recompile firmware

Open your local ESP-AT project and do the following:

- Replace `mqtt_ca.crt` with `Amazon-root-CA-1.pem`.
- Replace `mqtt_client.crt` with `device.pem.crt`.
- Replace `mqtt_client.key` with `private.pem.key`.

Then compile the ESP-AT project to build the AT firmware, and flash the firmware to your ESP32. For more information, please refer to [Compile ESP-AT Project Locally](#).

Method 2: Update certificates at runtime

If you don't want to recompile the firmware, you can directly use the `AT+SYSMFG` command to update MQTT certificates at runtime. For detailed operation steps, please refer to [PKI Configuration in AT+SYSMFG command examples](#). The certificate configuration method is the same as SSL certificates, just change the namespace to `mqtt_cert`, `mqtt_key`, and `mqtt_ca`.

Method 3: Update only the certificate bin file

If you already have AT firmware and only need to pre-flash your own certificates, you can directly update the `mfg_nvs.bin` file. For detailed operation steps, please refer to [How to Update PKI Configuration](#).

Use AT commands to connect to AWS IoT

1. Set the Wi-Fi mode to station.

Command:

```
AT+CWMODE=1
```

Response:

```
OK
```

2. Connect to the AP.

Command:

```
AT+CWJAP=<"ssid">,<"password">
```

Response:

```
OK
```

3. Set the SNTP server.

Command:

```
AT+CIPSNTPCFG=1,8,"pool.ntp.org"
```

Response:

```
OK
```

4. Query the SNTP time.

Command:

```
AT+CIPSNTPTIME?
```

Response:

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

Note:

- The <asctime style time> obtained at this time must be the real-time time of the set time zone, otherwise the connection will fail due to the validity period of the certificate.

5. Set MQTT user properties.

Command:

```
AT+MQTTUSERCFG=0,5,"esp32","espressif","1234567890",0,0,""
```

Response:

```
OK
```

Note:

- If the second parameter of AT+MQTTUSERCFG is 5, it is authenticated by both sides and cannot be changed.

6. Connect to AWS IoT.

Command:

```
AT+MQTTCONN=0,"<endpoint>",8883,1
```

Response:

```
+MQTTCONNECTED:0,5,<endpoint>,"8883","",1  
OK
```

Note:

- Please fill in your endpoint value in the <endpoint> parameter.
- The port 8883 cannot be changed.

7. Subscribe to messages.

Command:

```
AT+MQTTSUB=0,"topic/esp32at",1
```

Response:

```
OK
```

8. Publish a message.

Command:

```
AT+MQTTPUB=0,"topic/esp32at","hello aws!",1,0
```

Response:

```
+MQTTSUBRECV:0,"topic/esp32at",10,hello aws!  
OK
```

Example log

Log for normal interaction is as follows:

1. Log on the ESP32 side
2. Log on the AWS side

```
[20:12:43:217] AT+CWMODE=1 Ⓢ Ⓛ Ⓡ
[20:12:43:217] Ⓢ Ⓛ Ⓡ
[20:12:43:217] OK Ⓢ Ⓛ Ⓡ
[20:12:56:684] AT+CWJAP="MERCURY_407","" Ⓢ Ⓛ Ⓡ
[20:12:58:234] WIFI CONNECTED Ⓢ Ⓛ Ⓡ
[20:12:58:937] WIFI GOT IP Ⓢ Ⓛ Ⓡ
[20:12:58:937] Ⓢ Ⓛ Ⓡ
[20:12:58:937] OK Ⓢ Ⓛ Ⓡ
[20:13:01:892] AT+CIPSNTPCFG=1,8,"ntp1.aliyun.com" Ⓢ Ⓛ Ⓡ
[20:13:01:892] Ⓢ Ⓛ Ⓡ
[20:13:01:892] OK Ⓢ Ⓛ Ⓡ
[20:13:10:854] AT+CIPSNTPTIME? Ⓢ Ⓛ Ⓡ
[20:13:10:854] +CIPSNTPTIME:Wed Jan 19 20:13:10 2022 Ⓢ Ⓛ Ⓡ
[20:13:10:854] OK Ⓢ Ⓛ Ⓡ
[20:13:15:716] AT+MQTTUSERCFG=0,5,"esp32","espressif","1234567890",0,0,"" Ⓢ Ⓛ Ⓡ
[20:13:15:716] Ⓢ Ⓛ Ⓡ
[20:13:15:716] OK Ⓢ Ⓛ Ⓡ
[20:13:23:030] AT+MQTTCONN=0,"a108b8vm1g634q-ats.iot.us-east-2.amazonaws.com",8883,1 Ⓢ Ⓛ Ⓡ
[20:13:31:479] +MQTTCONNECTED:0,5,"a108b8vm1g634q-ats.iot.us-east-2.amazonaws.com",8883,"",1 Ⓢ Ⓛ Ⓡ
[20:13:31:479] Ⓢ Ⓛ Ⓡ
[20:13:31:479] OK Ⓢ Ⓛ Ⓡ
[20:13:34:275] AT+MQTTSUB=0,"topic/esp32at",1 Ⓢ Ⓛ Ⓡ
[20:13:35:521] Ⓢ Ⓛ Ⓡ
[20:13:35:521] OK Ⓢ Ⓛ Ⓡ
[20:13:41:959] AT+MQTTPUB=0,"topic/esp32at","hello aws!",1,0 Ⓢ Ⓛ Ⓡ
[20:13:42:422] +MQTTSUBRECV:0,"topic/esp32at",10,hello aws! Ⓢ Ⓛ Ⓡ
[20:13:42:422] Ⓢ Ⓛ Ⓡ
[20:13:42:422] OK Ⓢ Ⓛ Ⓡ
[20:13:49:335] +MQTTSUBRECV:0,"topic/esp32at",45,{ Ⓢ Ⓛ Ⓡ
[20:13:49:335]   "message": "Hello from AWS IoT console" Ⓢ Ⓛ Ⓡ
[20:13:49:335] } Ⓢ Ⓛ Ⓡ
```

MQTT client [Info](#) Connected as **lotconsole-1642593579651-0**

Subscriptions

[Subscribe to a topic](#)
[Publish to a topic](#)
topic/esp32at ✕

topic/esp32at

Export Clear Pause

Publish

Specify a topic and a message to publish with a QoS of 0.

topic/esp32at

Publish to topic

```
1 {
2   "message": "Hello from AWS IoT console"
3 }
```

topic/esp32at

January 19, 2022, 20:13:49 (UTC+0800)

Export Hide

```
{
  "message": "Hello from AWS IoT console"
}
```

topic/esp32at

January 19, 2022, 20:13:42 (UTC+0800)

Export Hide

We cannot display the message as JSON, and are instead displaying it as UTF-8 String.

hello aws!

Espressif Systems

322
[Submit Document Feedback](#)

Release latest

4.7 ESP32 Ethernet AT Examples

This document provides an introduction and detailed command examples to illustrate how to utilize *ESP32 Ethernet AT Commands* on ESP32.

- *Establish a TCP connection on Ethernet network*

Important:

- Before you run any Ethernet AT commands, please make sure you have followed the *Prerequisite*.
- The examples described in this document are based on the situation that network cable has been plugged in.

4.7.1 Establish a TCP connection on Ethernet network

1. Enable multiple connections.

Command:

```
AT+CIPMUX=1
```

Response:

```
OK
```

2. Create a TCP server.

Command:

```
AT+CIPSERVER=1,8081
```

Response:

```
OK
```

3. Obtain the IP address of the server.

Command:

```
AT+CIPETH?
```

Response:

```
+CIPETH:ip:192.168.105.24
+CIPETH:gateway:192.168.105.1
+CIPETH:netmask:255.255.255.0
OK
```

Note:

- The address you obtain may be different from that in the above response.

4. Use a network tool on PC to create a TCP client and connect to the TCP server by the step 2, which IP address is 192.168.105.24, port is 8081.
5. Send 4 bytes of data to transmission link 0 in *Normal Transmission Mode*.

Command:

```
AT+CIPSEND=0,4
```

Response:

```
OK
```

```
>
```

Input 4 bytes, for example, test, then AT will respond the following messages.

```
Recv 4 bytes
```

```
SEND OK
```

Note:

- If the number of bytes input exceeds the length (n) set by AT+CIPSEND, the system will reply busy p . . . , and send the first n bytes. After sending the first n bytes, the system will reply SEND OK.
6. Receive 4 bytes of data from transmission link 0 in *Normal Transmission Mode*.
Assume that the TCP server received 4 bytes of data (data is test), the system would be prompt as:

```
+IPD,0,4:test
```

7. Close TCP connection.

Command:

```
AT+CIPCLOSE=0
```

Response:

```
0,CLOSED
```

```
OK
```

8. Delete the TCP server.

Command:

```
AT+CIPSERVER=0
```

Response:

```
OK
```

Note:

- The AT+CIPSERVER=0 command will only shutdown the server, but will keep the existing connection. If you want to close all client connections to the server at the same time, please execute the command AT+CIPSERVER=0,1.

4.8 Web Server AT Example

This document mainly introduces the use of AT web server, mainly involving the following applications:

- *Wi-Fi Provisioning Using a Browser*
- *OTA Firmware Upgrade Using a Browser*
- *Wi-Fi Provisioning Using a WeChat Applet*
- *OTA Firmware Upgrade Using a WeChat Applet*
- *ESP32 Using Captive Portal*

Note: The default firmware does not support web server AT commands, please refer to [Web Server AT Commands](#) to enable the web server.

4.8.1 Wi-Fi Provisioning Using a Browser

Introduction

With the web server, mobile phone or PC is able to control ESP32's Wi-Fi provisioning service. You can use a mobile phone or computer to connect to the SoftAP of the ESP32, open the web pages via browser, start provisioning

service, and then the ESP32 can connect to the target router as you set.

Introduction to Operation Steps

The whole process can be divided into the following three steps:

- *Use STA Device to Connect to ESP32 Device*
- *Use the Browser to Send Wi-Fi Connection Information*
- *Get the Result of Wi-Fi Connection*

Use STA Device to Connect to ESP32 Device Firstly, ESP32 needs to be configured to softAP + STA mode, and creates a web server to wait for Wi-Fi provisioning messages. In this case, a mobile phone or a PC can connect to the ESP32 softAP as a station. The corresponding AT commands are as follows:

1. Clear the previous Wi-Fi provisioning information.

- Command

```
AT+RESTORE
```

2. Set the Wi-Fi mode to Station+SoftAP.

- Command

```
AT+CWMODE=3
```

3. Set the configuration of an ESP32 SoftAP. (For example, set the default connection ssid to “pos_softap” , Wi-Fi without password.)

- Command

```
AT+CWSAP="pos_softap","",11,0,3
```

4. Enable multiple connections.

- Command

```
AT+CIPMUX=1
```

5. Create a web server, port: 80, connection timeout: 25 s (default maximum is 60 s).

- Command

```
AT+WEBSERVER=1,80,25
```

After starting the web sever according to the above commands, you can turn on the Wi-Fi connection function on your STA device, and connect it to the softAP of the ESP32:

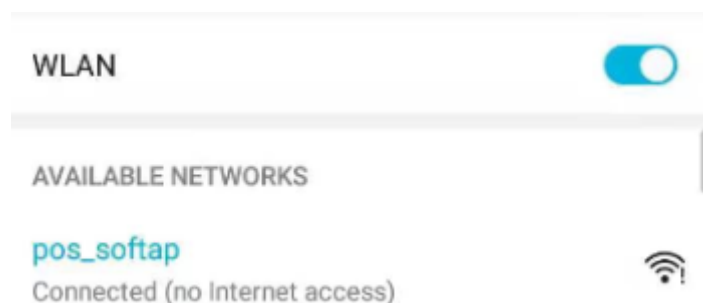


Fig. 1: Connection to the ESP32 AP

Use the Browser to Send Wi-Fi Connection Information After your STA device connected to the ESP32 softAP, it can send Wi-Fi connection information to ESP32 in an HTTP request. Please note that if your target AP is the hotspot of the device which opens the web pages, you may not receive the Wi-Fi connection result. You can enter the

default IP address of the web server in the browser (the default IP is 192.168.4.1, or you can query the current SoftAP IP address by command AT+CIPAP?), open the Wi-Fi provisioning interface, and enter the ssid and password of the router to be connected, click “Connect” to let ESP32 start connecting to the router:

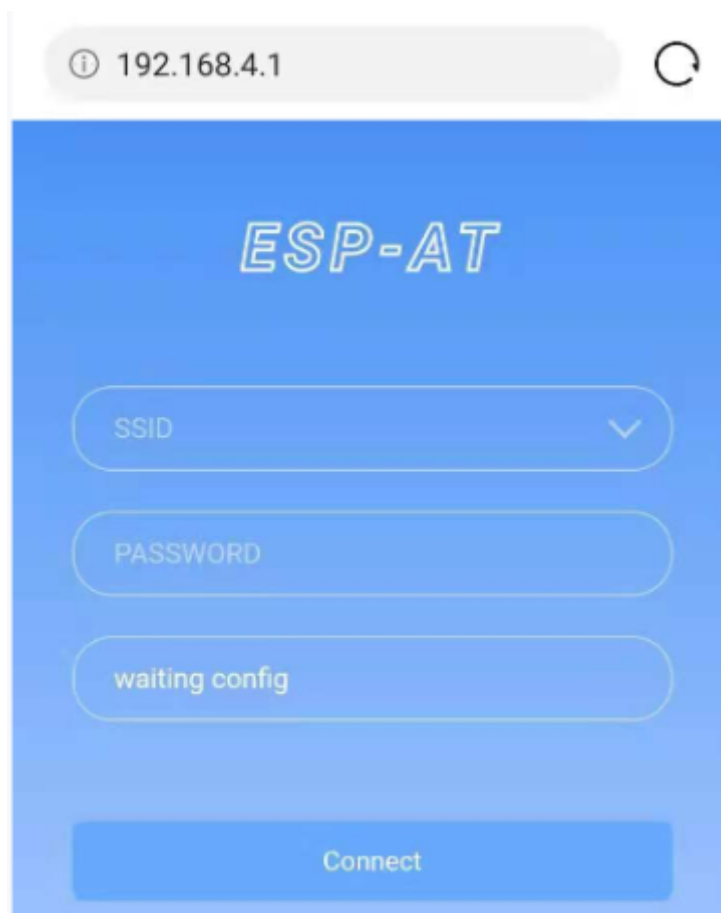


Fig. 2: Opening the Wi-Fi Provisioning Interface

Or you can click the drop-down box of SSID to list all APs nearby, select the target AP and enter the password, and then click “Connect” to let the ESP32 start connecting to the router:

Get the Result of Wi-Fi Connection After the Wi-Fi connection is established successfully, the web page will be displayed as follows:

Note 1: After the Wi-Fi connection is established successfully, the webpage will be closed automatically. If you want to continue to access the webpage, please re-enter the IP address of the ESP32 and reopen the webpage.

At the same time, the following messages will be returned from the ESP-AT command port:

```
+WEBSERVERRSP:1      // meaning that ESP32 has received Wi-Fi connection.
↪information
WIFI CONNECTED       // meaning that ESP32 is connecting
WIFI GOT IP          // meaning that ESP32 connect successfully to the destination.
↪router
+WEBSERVERRSP:2      // meaning that STA device has received Wi-Fi connection.
↪result, and web resources can be released
```

If the ESP32 fails to connect to the router, the web page will display:

At the same time, the following messages will be returned from the ESP-AT command port:

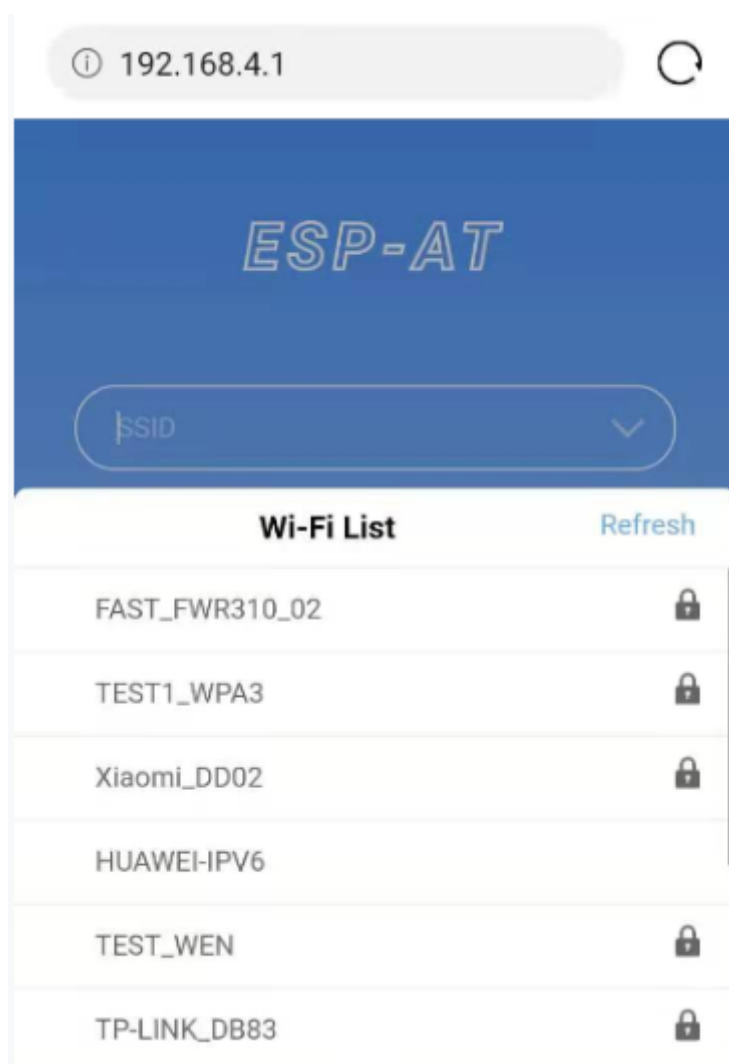


Fig. 3: Schematic Diagram Of Browser Obtaining Wi-Fi AP List

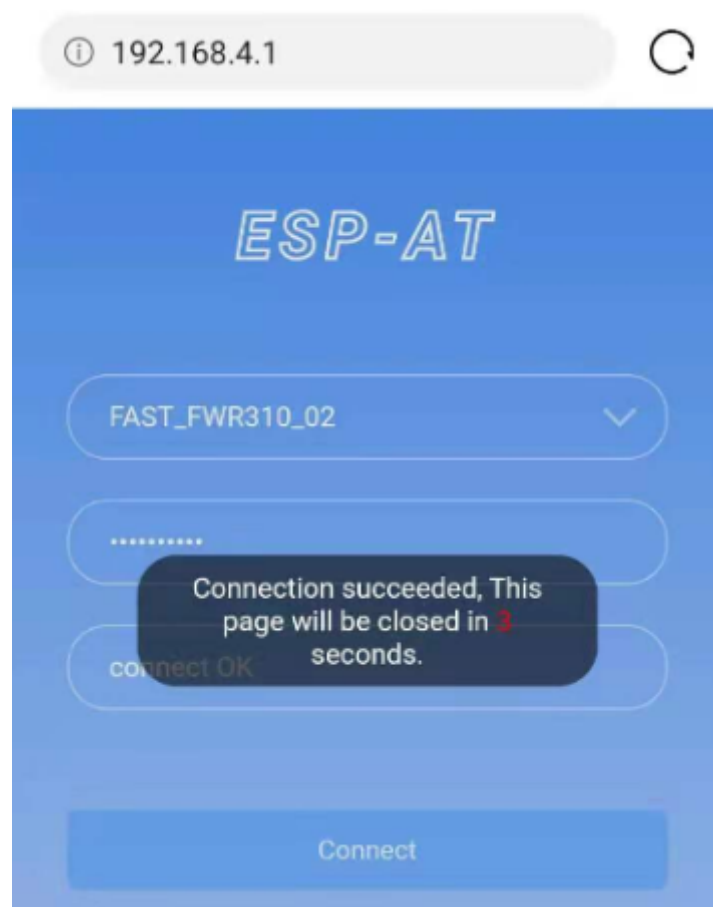


Fig. 4: Wi-Fi Connection Established Successfully

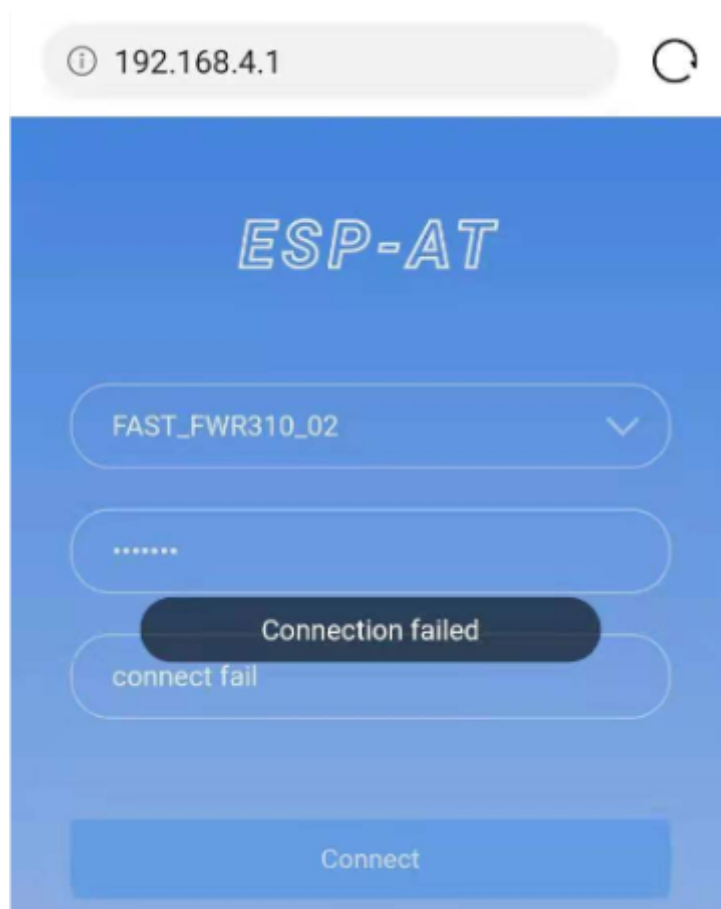


Fig. 5: Failed Connection to the Router

```
+WEBSERVERRSP:1          // meaning that ESP32 has received Wi-Fi connection.
↪information, but failed to connect to the router.
```

Troubleshooting

Note 1: The network configuration page received a prompt “Connection failed”. Please check whether the Wi-Fi AP of the ESP32 module is correctly turned on, and the relevant configuration of the AP, and confirm that the correct AT command has been entered to successfully enable the web server.

4.8.2 OTA Firmware Upgrade Using a Browser

Introduction

After the browser opens the web page of the web server, you can choose to enter the OTA upgrade page to upgrade the firmware in the app partitions or the certificate binaries in other partitions (please refer to [How to Update PKI Configuration](#) for more about certificate information).

Introduction to Operation Steps

- [Open the OTA Configuration Page](#)
- [Selecting the Partition to Upgrade](#)
- [Send the New Firmware](#)
- [Get the Result of OTA](#)

Open the OTA Configuration Page As shown in the figure, click on the “OTA” option in the lower right corner of the web page, and after opening the OTA configuration page, you can view the current firmware version and AT Core version:

Note 1: The configuration interface can only be opened when the STA device is connected to the AP of the ESP32, or the STA device accessing the OTA configuration page is connected to the ESP32 in the same subnet.

Note 2: The “current app version” displayed on the webpage is the version number of the application. You can change the version number through `./build.py menuconfig > Component config > AT > AT firmware version` (see [Compile ESP-AT Project Locally](#)). In this case, you can manage your application firmware version.

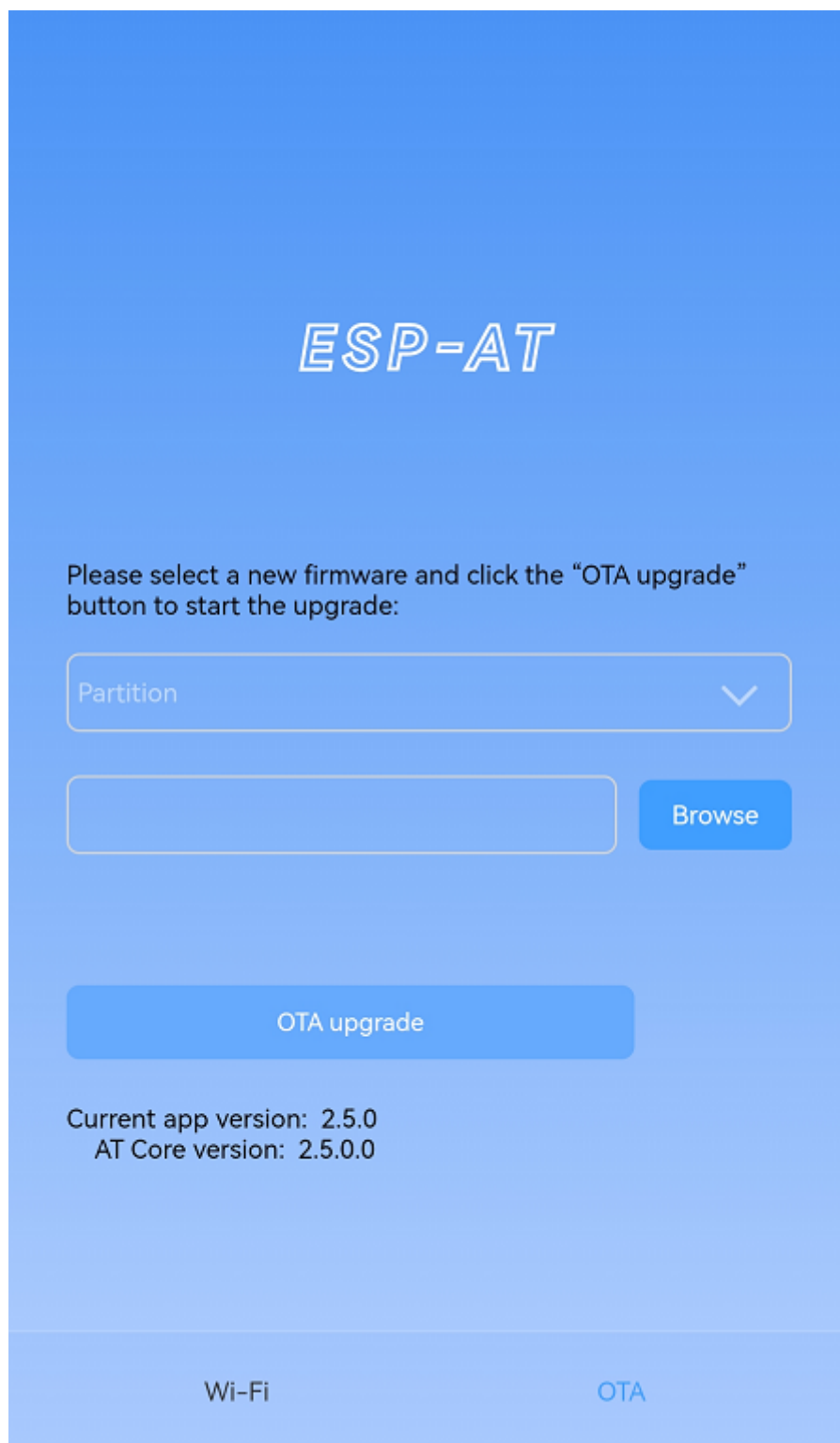
Selecting the Partition to Upgrade As shown in the figure, click the drop-down box of “Partition” to obtain all partitions that can be upgraded:

Send the New Firmware As shown in the figure, click the “Browse” button on the page and select the new firmware to be sent:

Then you can click “OTA upgrade” button to send the firmware.

Note 1: For the ota partition, the web page will check the selected firmware. The suffix of the firmware name must be `.bin`. Please make sure that the firmware size does not exceed the size of the ota partition defined in the `partitions_at.csv` file. For more information on this file, please refer to [How to Add Support for a Module](#).

Note 2: For other partitions, the web page will check the selected firmware. The suffix of the firmware name must be `.bin`. Please make sure that the firmware size does not exceed the size defined in the `at_customize.csv` file. For more information on this file, please refer to [How to Customize Partitions](#).



The image shows a web interface for OTA configuration. It has a blue gradient background. At the top, the text 'ESP-AT' is displayed in a white, stylized font. Below this, a message in black text says 'Please select a new firmware and click the "OTA upgrade" button to start the upgrade:'. There are two main input areas: a dropdown menu labeled 'Partition' with a downward arrow, and a file selection area consisting of a text box and a blue 'Browse' button. Below these is a large blue button labeled 'OTA upgrade'. At the bottom left, the current versions are listed: 'Current app version: 2.5.0' and 'AT Core version: 2.5.0.0'. At the very bottom, there is a navigation bar with 'Wi-Fi' and 'OTA' links, where 'OTA' is highlighted in blue.

ESP-AT

Please select a new firmware and click the "OTA upgrade" button to start the upgrade:

Partition 

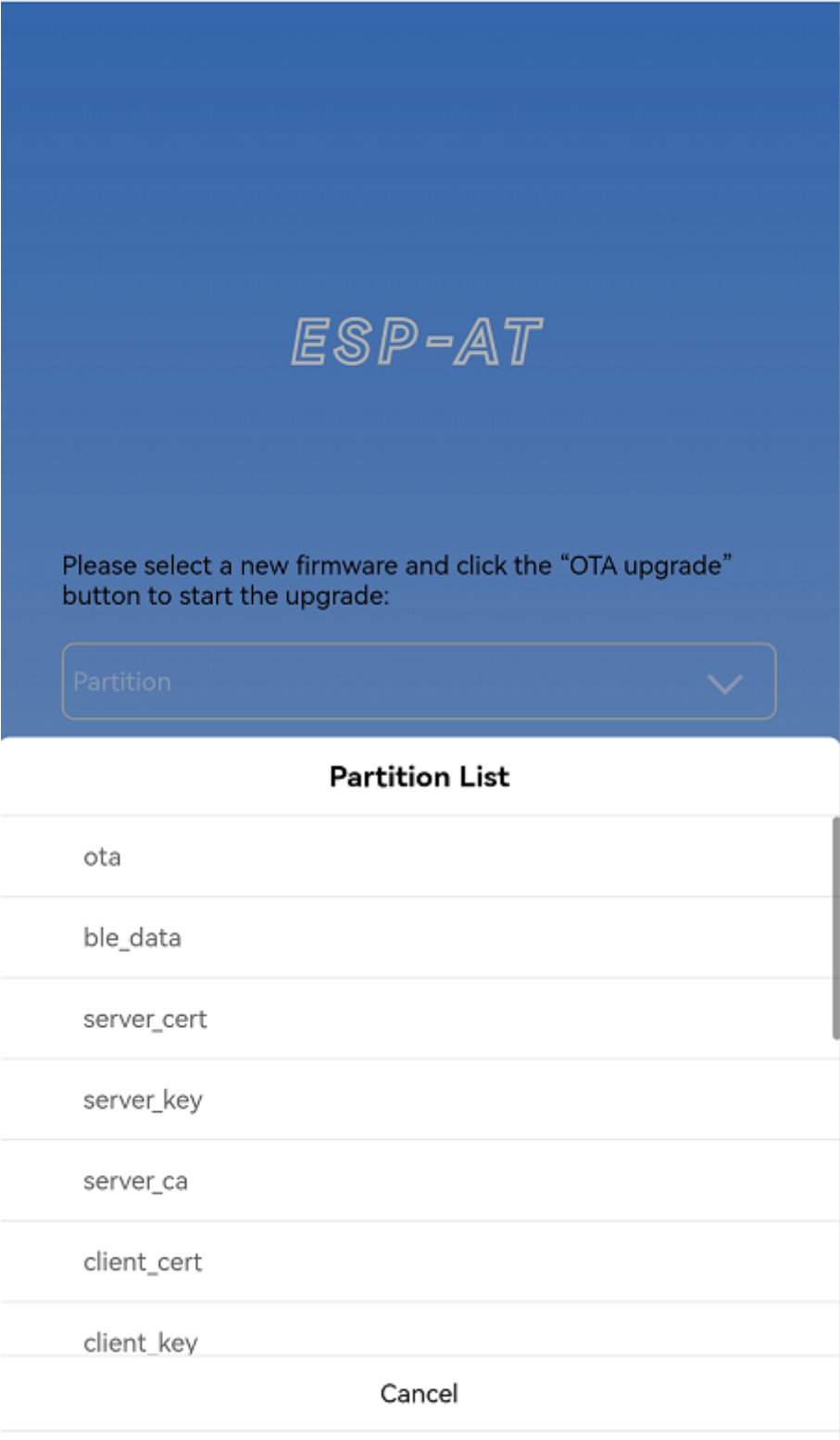
[Browse](#)

[OTA upgrade](#)

Current app version: 2.5.0
AT Core version: 2.5.0.0

Wi-Fi [OTA](#)

Fig. 6: OTA Configuration Page



The image shows a web interface for the ESP-AT module. At the top, the text "ESP-AT" is displayed in a stylized font. Below it, a message instructs the user to select a new firmware and click the "OTA upgrade" button. A dropdown menu labeled "Partition" is shown, which is currently open, displaying a list of partitions: ota, ble_data, server_cert, server_key, server_ca, client_cert, and client_key. A "Cancel" button is located at the bottom of the list.

ESP-AT

Please select a new firmware and click the "OTA upgrade" button to start the upgrade:

Partition ▼

Partition List

ota
ble_data
server_cert
server_key
server_ca
client_cert
client_key

Cancel

Fig. 7: Obtaining All Partitions That Can Be Upgraded

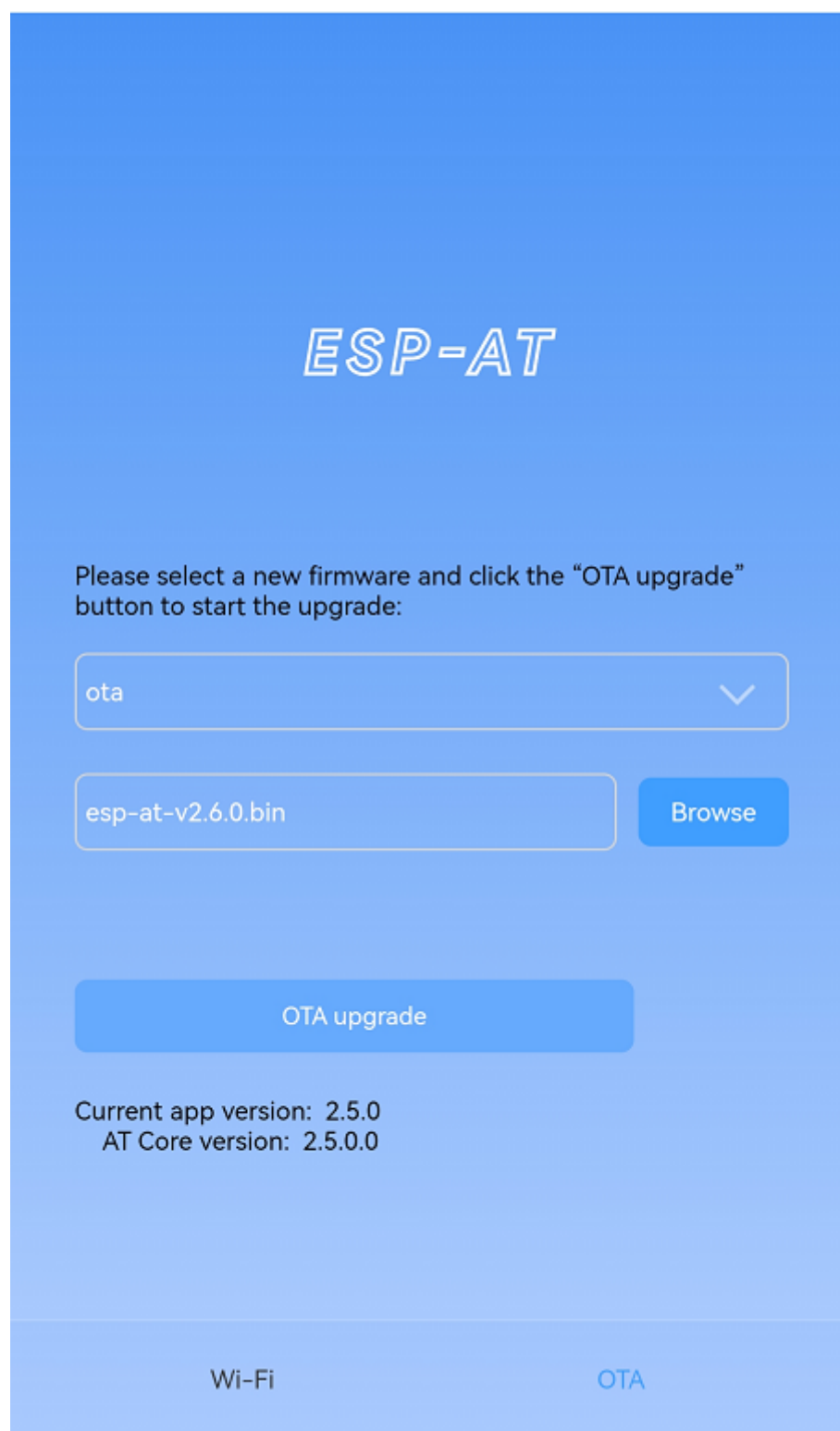


Fig. 8: Selecting the New Firmware to Be Sent

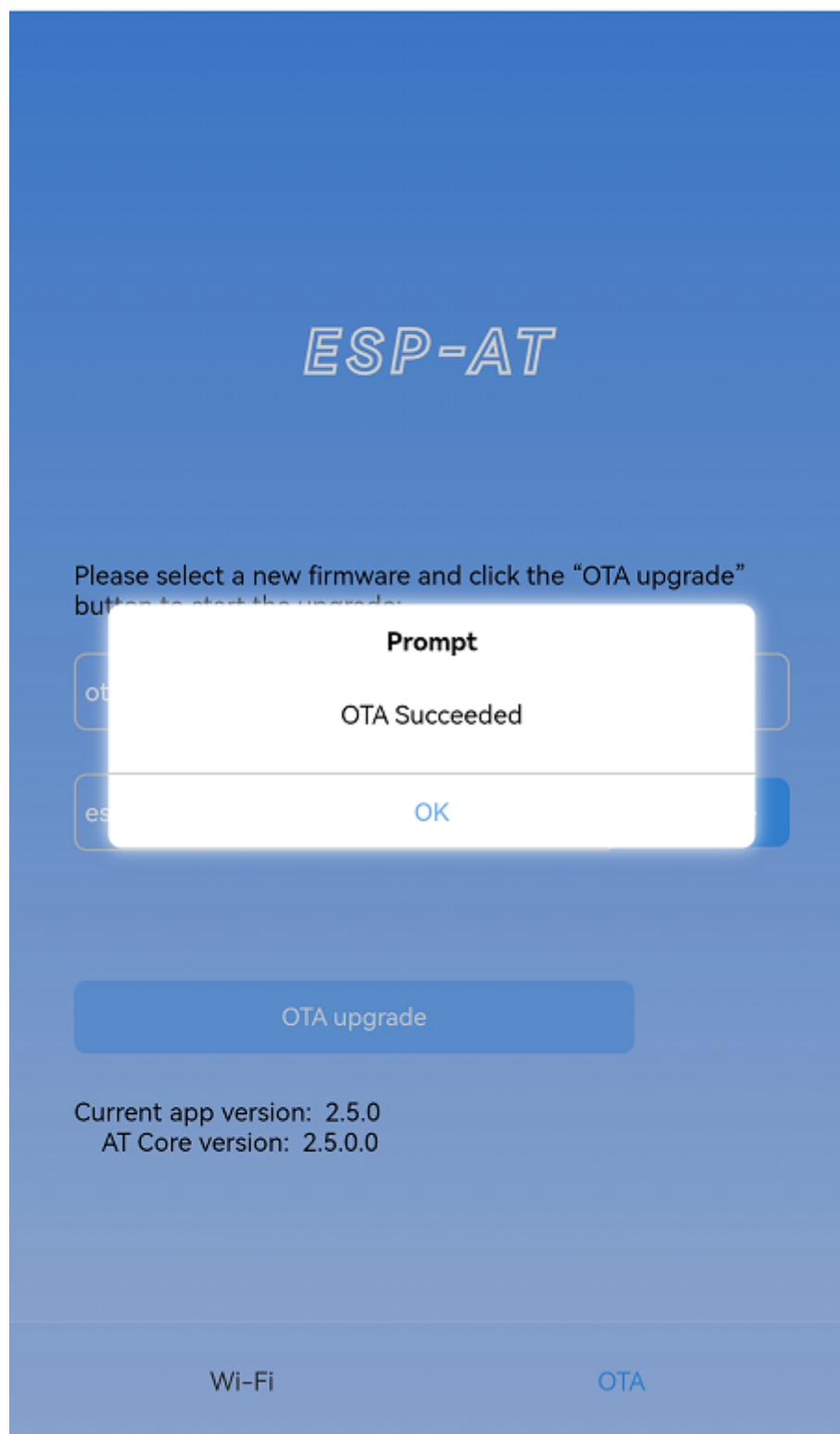


Fig. 9: The New Firmware Sent Successfully

Get the Result of OTA As shown in the figure, if the ESP32 OTA successfully, it will prompt “OTA Succeeded” :

At the same time, the following messages will be returned from the ESP-AT command port:

```
+WEBSERVERRSP:3      // meaning that ESP32 begin to receive OTA data
+WEBSERVERRSP:4      // meaning that ESP32 has received all firmware data
```

If the ESP32 OTA failed, it will prompt “OTA Failed” :

At the same time, the following message will be received on the serial port:

```
+WEBSERVERRSP:3      // meaning that ESP32 begin to receive OTA data
+WEBSERVERRSP:5      // meaning a failure of receiving OTA data failed. You can
↳ choose to reopen the OTA configuration interface and follow the above steps to
↳ restart the firmware upgrade
```

Note 1: For the ota partition, you need to execute *AT+RST* to restart the ESP32 to apply the new firmware.

Note 2: ESP32 will only verify the received ota firmware content. The firmware data received by other partitions will not be verified, so please make sure its content is correct.

4.8.3 Wi-Fi Provisioning Using a WeChat Applet

Introduction

The WeChat applet can automatically connect to the WiFi AP of the ESP32, and then send the ssid and password required by the ESP32 to connect to the network.

Important: The ESP-AT WeChat applet will be discontinued on December 31, 2024. During this period, please make arrangements for the transition of its functionalities. If you still have related needs after this date, please contact Espressif’s business team. We will be dedicated to providing you with support and solutions. Thank you for your understanding and support.

Introduction to Operation Steps

The whole process can be divided into the following four steps:

- *Configure ESP32 Device Parameters*
- *Load WeChat Applet*
- *Target AP Selection*
- *Use the WeChat Applet to Send Wi-Fi Connection Information*

Configure ESP32 Device Parameters Firstly, ESP32 needs to be configured to softAP + STA mode, and creates a web server to wait for Wi-Fi provisioning messages. In this case, a mobile phone or a PC can connect to the ESP32 softAP as a station. The corresponding AT commands are as follows:

1. Clear the previous Wi-Fi provisioning information.
 - Command

```
AT+RESTORE
```

2. Set the Wi-Fi mode to Station+SoftAP.
 - Command

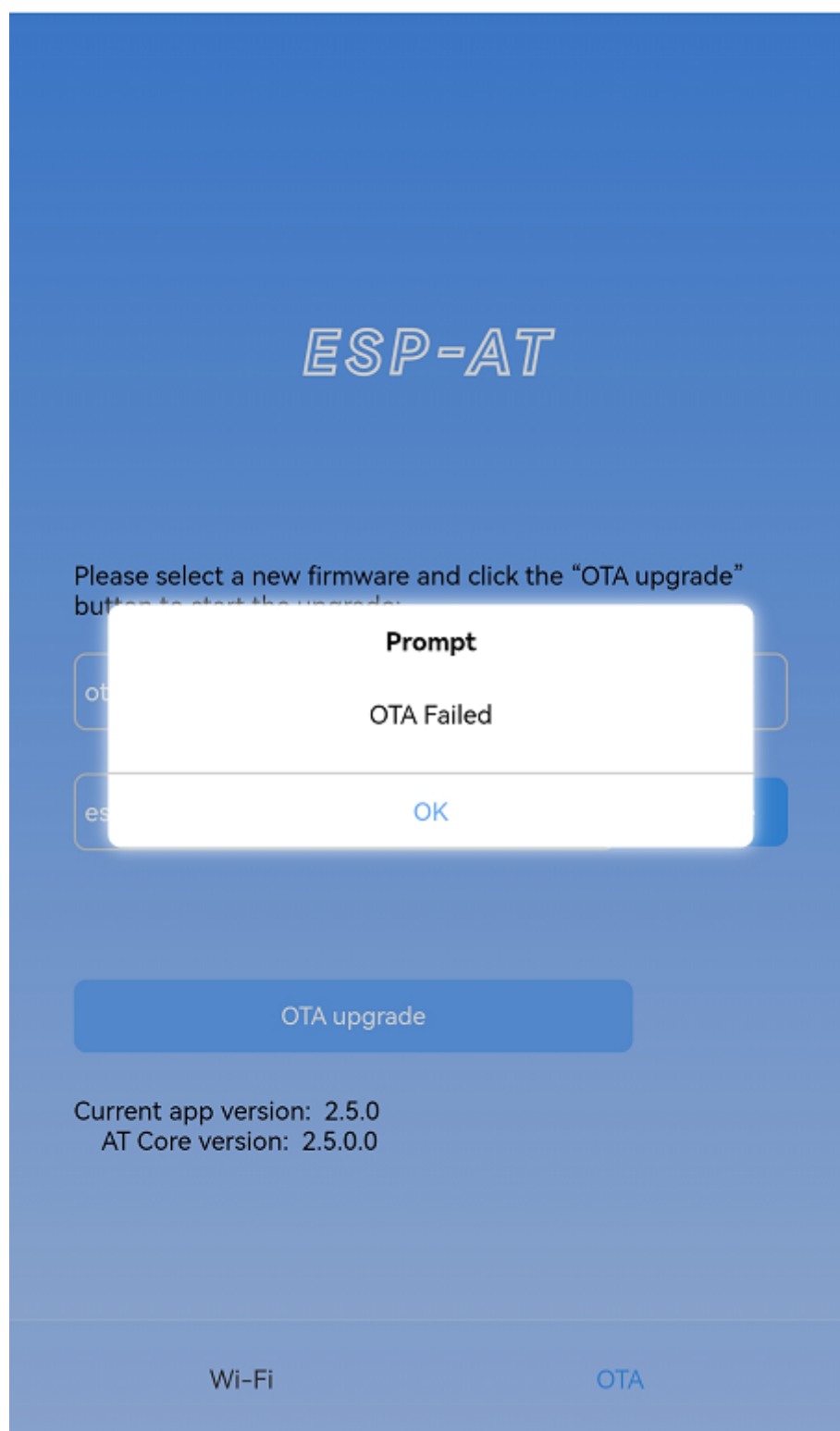


Fig. 10: Failed Sending of the New Firmware

```
AT+CWMODE=3
```

- Set the configuration of an ESP32 SoftAP. (For example, set the default connection ssid to “pos_softap”, and password to “espressif” .)
 - Command

```
AT+CWSAP="pos_softap","espressif",11,3,3
```

Note: By default, the WeChat applet initiates a connection to the SoftAP whose ssid is *pos_softap* and password is *espressif*. Please make sure to set the parameters of the ESP32 according to the above configuration.

- Enable multiple connections.

- Command

```
AT+CIPMUX=1
```

- Create a web server, port: 80, connection timeout: 40 s (default maximum is 60 s).

- Command

```
AT+WEBSERVER=1,80,40
```

Load WeChat Applet Open the mobile phone WeChat, scan the following QR code:



Fig. 11: Getting the QR Code of the Applet

Open the WeChat applet and enter the Wi-Fi provisioning interface:

Target AP Selection After loading the WeChat applet, there are two situations according to different target AP:

Situation 1. If your target AP is the hotspot of the mobile phone which running the WeChat applet, please select the “Local phone hotspot” option box on the WeChat applet page.

Situation 2. If your target AP is just another AP, not as the special situation one as above, then please do not select the “Local phone hotspot” option box.

Use the WeChat Applet to Send Wi-Fi Connection Information

The target AP to be accessed is not the hotspot provided by the mobile phone which loading the WeChat applet. Here, take connecting to a router as an example, the process of Wi-Fi Connection configuration is introduced:

- Turn on the mobile Wi-Fi and connect to the router:

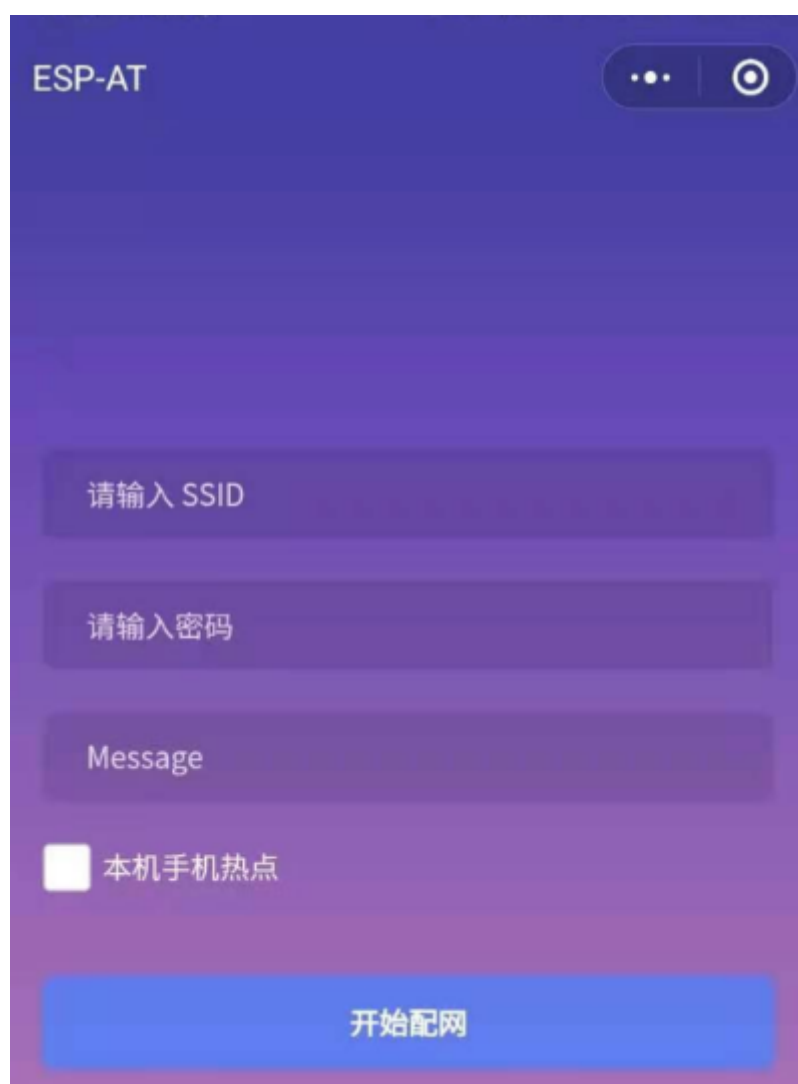


Fig. 12: Wi-Fi Provisioning Interface



Fig. 13: Connection to the Router

2. Open the WeChat applet, you can see that the applet page has automatically displayed the ssid of the current router as “FAST_FWR310_02” .



Fig. 14: Getting Router Information

Note: If the ssid of the connected router is not displayed on the current page, please click “Re-enter applet” in the following figure to refresh the current page:

3. After entering the password of the router, click “Connect” .

4. After the Wi-Fi connection is established successfully, the web page will be displayed as follows:

At the same time, the following messages will be returned from the ESP-AT command port:

```
+WEBSERVERRSP:1      // meaning that ESP32 has received Wi-Fi connection.
↪information
WIFI CONNECTED       // meaning that ESP32 is connecting
WIFI GOT IP          // meaning that ESP32 connect successfully to the destination.
↪router
+WEBSERVERRSP:2      // meaning that STA device has received Wi-Fi connection.
↪result, and web resources can be released
```

5. If the ESP32 fails to connect to the router, the page will display:

At the same time, the following messages will be returned from the ESP-AT command port:



Fig. 15: Re-entering the Applet



Fig. 16: Connection to the Router via the Applet



Fig. 17: Successfully connection to the Router via the Applet



Fig. 18: Failed Connection to the Router via the Applet

```
+WEBSERVERRSP:1 // meaning that ESP32 has received Wi-Fi connection  
↳ information, but failed to connect to the router.
```

The target AP to be accessed is the hotspot provided by the mobile phone which loading the WeChat applet.

If the target AP to be accessed is the hotspot provided by the mobile phone which loading the WeChat applet, it is not necessary to enter the ssid, but only needs to enter the password of the AP, and turn on the mobile AP in time according to the prompts.

Note: To use this function, keep at least the first five bytes of the phone's Personal Hotspot MAC address the same as those of the WLAN MAC address.

1. Select the “Local phone hotspot” option box on the WeChat applet page, enter the password of the local hotspot, and click “Connect” .

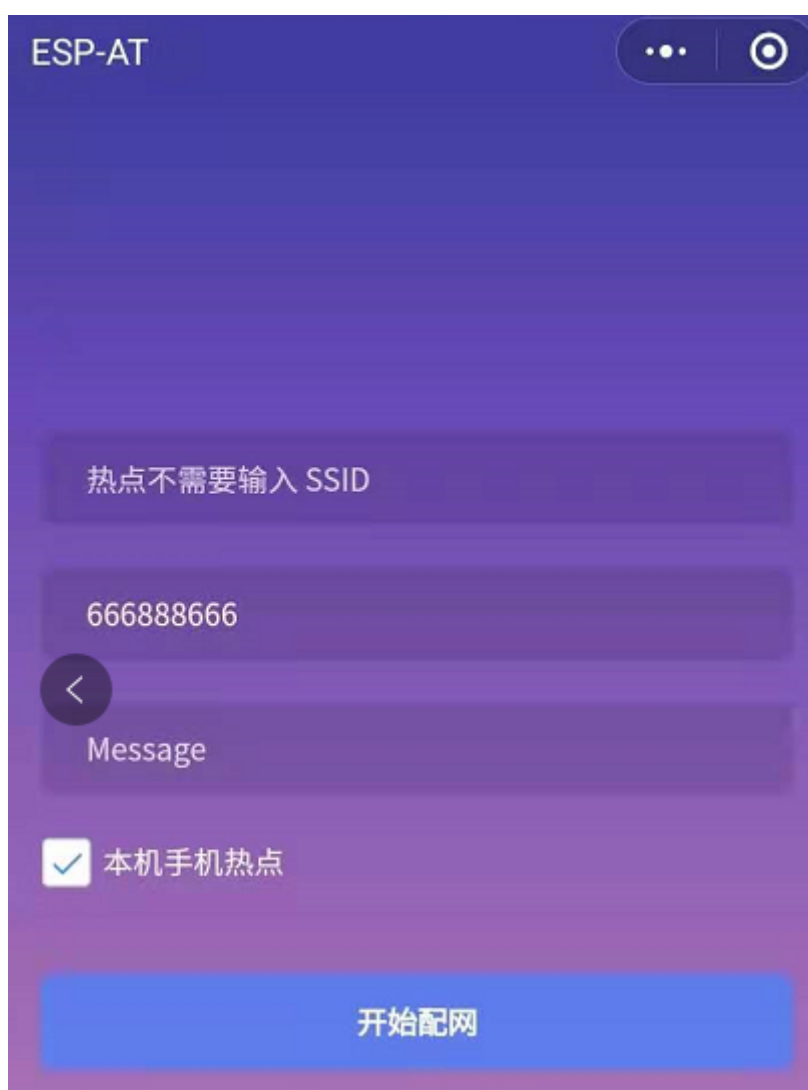


Fig. 19: Entering the Password of the AP

2. After receiving the prompt “Connecting to the mobile phone hotspot” , please check that the local mobile phone hotspot is turned on. At this time, the ESP32 will automatically scan the surrounding hotspots and initiate a connection.

3. The display of the WiFi connection results on the applet page and the data output from the serial port are the same

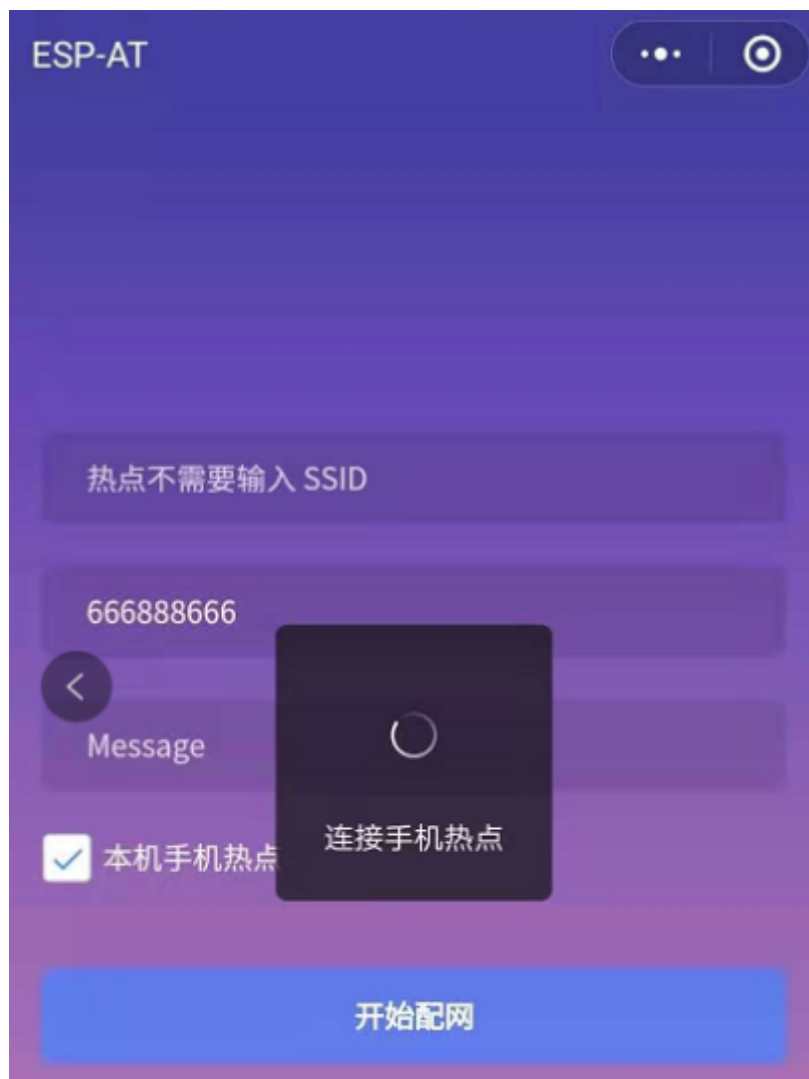


Fig. 20: Starting Connection to the AP

as the above-mentioned “The target AP to be accessed is not the hotspot provided by the mobile phone which loading the WeChat applet.” , please refer to the above.

Troubleshooting

Note 1: The Wi-Fi provisioning page received a prompt “Data transmission failed” . Please check whether the Wi-Fi AP of the ESP32 is correctly turned on, and the relevant configuration of the AP, and confirm that the correct AT command has been entered to successfully enable the web server.

Note 2: The Wi-Fi provisioning page receives a prompt “Failed to connect to the AP” . Please check whether the Wi-Fi connection function of the mobile phone is turned on, check whether the Wi-Fi AP of the ESP32 is correctly turned on, and whether the ssid and password of the AP are configured according to the above steps.

Note 3: The Wi-Fi provisioning page receives a prompt “The Wi-Fi provisioning saved by the system expired” . Please manually connect the ESP32 AP with a mobile phone, and confirm that the ssid and password of the ESP32 module have been configured according to the above steps.

4.8.4 OTA Firmware Upgrade Using a WeChat Applet

The WeChat applet support online firmware upgrade, please refer to the above-described [Configure ESP32 Device Parameters](#) specific steps performed ESP32 configuration (if the configuration has been completed, do not repeat configuration). Once configured, the device performs OTA firmware upgrade processes is similar as [OTA Firmware Upgrade Using a Browser](#) .

4.8.5 ESP32 Using Captive Portal

Introduction

Captive Portal is commonly used to present a specified page to newly connected devices of a Wi-Fi or wired network. For more information about Captive Portal, please refer to [Captive Portal Wiki](#) .

Note: The default firmware does not support web server Captive Portal, you may enable it by `./build.py menuconfig>Component config>AT>AT WEB Server command support>AT WEB captive portal support` and build the project (see [Compile ESP-AT Project Locally](#)). In addition, enabling this feature may cause page skipping when using wechat applet for Wi-Fi provisioning or OTA firmware upgrade. It is recommended that this feature be enabled only when accessing at web using browser.

Introduction to Operation Steps

After Enable Captive Portal support, please refer to [Use STA Device to Connect to ESP32 Device](#) to complete the configuration of the ESP32, and then connect to the AP of the ESP32:

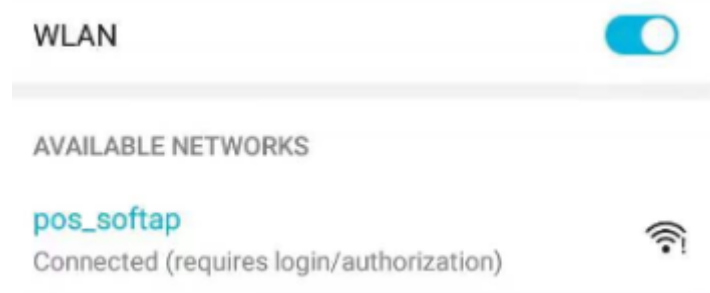


Fig. 21: Connection to the AP with Captive Portal Enabled

As shown in the figure above, after the Station device is connected to the AP of the ESP32 with the Captive Portal function enabled, it will prompt “requires login/authentication”, and then the browser will automatically open and jump to the main interface of AT Web. If it cannot be redirected automatically, please follow the instructions of the Station device, click “Authentication” or click the name of the “pos_softap” hotspot in the figure above to manually trigger the Captive Portal to automatically open the browser and enter the main interface of AT Web.

Troubleshooting

Note 1: Both Station device and AP device support the Captive Portal function to ensure the normal use of this function. Therefore, if the device is connected to the AP of the ESP32, but it does not prompt “Login/Authentication”, it may be that the Station device does not support this function. In this case, please refer to the specific steps of [Use the Browser to Send Wi-Fi Connection Information](#) above to open the main interface of AT Web.

4.9 HTTP AT Examples

This document provides detailed command examples to illustrate how to utilize [HTTP AT Commands](#) on ESP32.

- *The HEAD method of HTTP client*
- *The GET method of HTTP client*
- *The POST method of HTTP client (suitable for POST small amount of data)*
- *The POST method of HTTP client (recommended method)*
- *The PUT method of HTTP client (suitable for no data put)*
- *The PUT method of HTTP client (recommended method)*
- *The DELETE method of HTTP client*

Important: Currently ESP-AT only supports some HTTP client functions.

4.9.1 The HEAD method of HTTP client

In this example, the HTTP server is <http://httpbin.org>.

1. Restore factory default settings of the module.

Command:

```
AT+RESTORE
```

Response:

```
OK
```

2. Set the Wi-Fi mode to station.

Command:

```
AT+CWMODE=1
```

Response:

```
OK
```

3. Connect to the router.

Command:

```
AT+CWJAP="espressif","1234567890"
```

Response:

```
WIFI CONNECTED
WIFI GOT IP

OK
```

Note:

- The SSID and password you entered may be different from those in the above command. Please replace the SSID and password with those of your router settings.
4. Send an HTTP HEAD request. Set opt to 1 (HEAD method), url to `http://httpbin.org/get` and `transport_type` to 1 (HTTP_TRANSPORT_OVER_TCP).

Command:

```
AT+HTTPCLIENT=1,0,"http://httpbin.org/get",,,1
```

Response:

```
+HTTPCLIENT:35, Date: Sun, 26 Sep 2021 06:59:13 GMT
+HTTPCLIENT:30, Content-Type: application/json
+HTTPCLIENT:19, Content-Length: 329
+HTTPCLIENT:22, Connection: keep-alive
+HTTPCLIENT:23, Server: gunicorn/19.9.0
+HTTPCLIENT:30, Access-Control-Allow-Origin: *
+HTTPCLIENT:38, Access-Control-Allow-Credentials: true

OK
```

Note:

- The HTTP header information you obtain may be different from those in the above response.

4.9.2 The GET method of HTTP client

This example describes how to download an image file in JPG format. The image link is <https://www.espressif.com/sites/all/themes/espressif/images/about-us/solution-platform.jpg>.

1. Restore factory default settings of the module.

Command:

```
AT+RESTORE
```

Response:

```
OK
```

2. Set the Wi-Fi mode to station.

Command:

```
AT+CWMODE=1
```

Response:

```
OK
```

3. Connect to the router.

Command:

```
AT+CWJAP="espressif","1234567890"
```

Response:

```
WIFI CONNECTED
WIFI GOT IP

OK
```

Note:

- The SSID and password you entered may be different from those in the above command. Please replace the SSID and password with those of your router settings.
4. Send an HTTP GET request. Set opt to 2 (GET method), url to <https://www.espressif.com/sites/all/themes/espressif/images/about-us/solution-platform.jpg> and transport_type to 2 (HTTP_TRANSPORT_OVER_SSL).

Command:

```
AT+HTTPCLIENT=2,0,"https://www.espressif.com/sites/all/themes/espressif/images/
→about-us/solution-platform.jpg",,,2
```

Response:

```
+HTTPCLIENT:512,<0xff><0xd8><0xff><0xe2><0x0c>XICC_PROFILE<break>
<0x01><0x01><break>
<break>
<0x0c>HLino<0x02><0x10><break>
<break>
mnrRGB XYZ <0x07><0xce><break>
<0x02><break>
...
+HTTPCLIENT:512,<0xeb><0xe2>v<0xcb><0x98>-<0xf8><0x8a><0xae><0xf3><0xc8><0xb6>
→<0xa8><0x86><0x02>j<0x06><0xe2>
"<0xaa>*p<0x7f>2",h<0x12>N<0xa5><0x1e><0xd2>bp<0xea><0x1e><0xf5><0xa3>x<0xa6>J
→<0x14>Ti<0xc3>m<0x1a>m<0x94>T<0xe1>I<0xb6><0x90><0xdc>_<0x11>QU;<0x94><0x97>
→<0xcb><0xdd><0xc7><0xc6><0x85><0xd7>U<0x02><0xc9>W<0xa4><0xaa><0xa1><0xa1>
→<0x08>hB<0x1a><0x10><0x86><0x84>!<0xa1><0x08>hB<0x1a><0x10><0x9b><0xb9>K
→<0xf5>5<0x95>5-=<0x8a><0xcb><0xce><0xe0><0x91><0xf0>m<0xa9><0x04>C<0xde>k
→<0xe7><0xc2>v<0xaf><0xb8>L<0x91>=<0xda>_<0x94><0xde><0xd0><0xa9><0xc0>
→<0xdd>8<0x9a>B<0xaa><0x1a><0x10><0x86><0x84>$<0xf4><0xd6><0xf2><0xa3><0x92>
→<0xe7><0x0a>I<0xa3>b<0x1f>)<0xe1>z<0xc4>y<0xae><0xca><0xed><0xec><0x1e>|^
→<0xd7>E<0xa2>_<0x13><0x9e>;{|<0xb5>Q<0x97><0xa5>P<0xdf><0xa1>#3vn<0x1b><0xc3>
→-<0x92><0xe2>dIn<0x9c><0xb8>
<0xc7><0xa9><0xc6>(<0xe0><0xd3>i-<0x9e>@<0xbb><0xcc><0x88><0xd5>K<0xe3><0xf0>O
→<0x9f>Km<0xb3>h<0xa8>omR<0xfe><0x8b><0xf9><0xa4><0xa6><0xff><break>
aU<0xdf><0xf3><0xa3>:A<0xe2>UG<0x04>k<0xaa>*<0xa1><0xa1><0x0b><0xca><0xec>
→<0xd8>Q<0xfb><0xbc>yqY<0xec><0xfb>?<0x16>CM<0xf6>|}<0xae><0xf3><0x1e><0xdf>%
→<0xf8><0xe8><0xb1>B<0x8f>[<0xb3>><0x04><0xec><0xeb>f<0x06><0x1c><0xe8><0x92>
→<0xc9><0x8c><0xb0>I<0xd1><0x8b>%<0x99><0x04><0xd0><0xbb>s<0x8b>xj<0xe2>4f
→<0xa0><0x8e>+E<0xda><0xab><0xc7>=<0xab><0xc7><0xb9>xz1f<0xba><0xfd>_e6<0xff>
→<break>
(w<0xa7>b<0xe3>m<0xf0>|<0x82><0xc9><0xfb><0x8b><0xac>r<0x95><0x94><0x96><0xd9>i
→<0xe9>RVA<0x91><0x83><0x8b>M'<0x86><0x8f><0xa3>A<0xd8><0xd8>"r"<0x8a><0xa8>
→<0x9e>z1=<0xcd><0x16><0x07>D<0xa2><0xd0>u(<0xc2><0x8b><0x0b><0xc4><0xf1>
→<0x87><0x9c><0x93><0x8f><0xe3><0xd5>U<0x12>]<0x8e><0x91>]<0x91><0x06>#1<0xbe>
→<0xf4>t0?<0xd7><0x85><GEM<0xb1>%<0xee>UUT<0xe7><0xdf><0xa0><0xb9><0xce><0xe2>
→U@<0x03><0x82>S<0xe9>*<0xa8>hB<0x1a><0x10><0xa1><0xaf>V<0x19>U<0x9d><0xb3>x
→<0xa6><0xc7><0xe2><0x86><0x8e>d[<0x89>e<0x05>l<0x80>H<0x91>#<0xd2><0x8c>
→<0xe1>j<0x1b>rH<0x04><0x89><0x98><0xd3>lZW]q<0xc2><'><0x93><0xb4><0xf5>&
→<0x9d><0xa0>^Wp<0xa9>6`<0xe2>T<0xa2><0xc2><0xb1>*<0xbc><0x13><0x13><0xa0>
→<0xc4>)<0x83><0xb6><0xbe><0x86><0xb9><0x88>-<0x1a>
```

OK

4.9.3 The POST method of HTTP client (suitable for POST small amount of data)

In this example, the HTTP server is <http://httpbin.org> and the data type is application/json.

1. Restore factory default settings of the module.

Command:

```
AT+RESTORE
```

Response:

```
OK
```

2. Set the Wi-Fi mode to station.

Command:

```
AT+CWMODE=1
```

Response:

```
OK
```

3. Connect to the router.

Command:

```
AT+CWJAP="espressif","1234567890"
```

Response:

```
WIFI CONNECTED
WIFI GOT IP
```

```
OK
```

Note:

- The SSID and password you entered may be different from those in the above command. Please replace the SSID and password with those of your router settings.

4. Send an HTTP POST request. Set `opt` to 3 (POST method), `url` to `http://httpbin.org/post`, `content-type` to 1 (application/json) and `transport_type` to 1 (HTTP_TRANSPORT_OVER_TCP).

Command:

```
AT+HTTPCLIENT=3,1,"http://httpbin.org/post",,,1,"{\"form\":{\"purpose\":\"test\
→\"}}"
```

Response:

```
+HTTPCLIENT:282,{
  "args": {},
  "data": "{\"form\":{\"purpose\":\"test\"}}",
  "files": {},
  "form": {},
  "headers": {
    "Content-Length": "27",
    "Content-Type": "application/json",
    "Host": "httpbin.org",
    "User-Agent": "ESP32 HTTP Client/1.0",
    "X-Amzn-Trace-Id": "Root=
+HTTPCLIENT:173,1-61503a3f-4b16b71918855b97614c5dfb"
  },
  "json": {
    "form": {
      "purpose": "test"
    }
  },
  "origin": "20.187.154.207",
  "url": "http://httpbin.org/post"
}

OK
```

Note:

- The results you obtain may be different from those in the above response.

4.9.4 The POST method of HTTP client (recommended method)

If the amount of data you post is relatively large, and the length of a single AT command has exceeded the threshold of 256, it is recommended that you use the `AT+HTTPCPOST` command.

In this example, the HTTP server is <http://httpbin.org> and the data type is `application/json`.

1. Restore factory default settings of the module.

Command:

```
AT+RESTORE
```

Response:

```
OK
```

2. Set the Wi-Fi mode to station.

Command:

```
AT+CWMODE=1
```

Response:

```
OK
```

3. Connect to the router.

Command:

```
AT+CWJAP="espressif","1234567890"
```

Response:

```
WIFI CONNECTED
WIFI GOT IP
OK
```

Note:

- The SSID and password you entered may be different from those in the above command. Please replace the SSID and password with those of your router settings.
4. Post HTTP data of specified length. The command specifies that the number of HTTP header fields is 2, which are connection field and content-type field respectively. connection is keep-alive and content-type is application/json. Assume the JSON data you want to post is of 427 bytes as follows.

```
{ "headers": { "Accept": "application/json", "Accept-Encoding": "gzip, deflate",
→ "Accept-Language": "en-US,en;q=0.9,zh-CN;q=0.8,zh;q=0.7", "Content-Length": "0
→", "Host": "httpbin.org", "Origin": "http://httpbin.org", "Referer": "http://
→httpbin.org/", "User-Agent": "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.
→36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36", "X-Amzn-Trace-Id
→": "Root=1-6150581e-1ad4bd5254b4bf5218070413" } }
```

Command:

```
AT+HTTPCPOST="http://httpbin.org/post",427,2,"connection: keep-alive","content-
→type: application/json"
```

Response:

```
OK
>
```

This response indicates that AT is ready for receiving serial data. You should enter the 427-byte data mentioned above, and when the data length reaches the `<length>` value, data transmission starts.

```

+HTTPCPOST:281,{
  "args": {},
  "data": "{ \"headers\": { \"Accept\": \"application/json\", \"Accept-Encoding\
→\": \"gzip, deflate\", \"Accept-Language\": \"en-US,en;q=0.9,zh-CN;q=0.8,zh;
→q=0.7\", \"Content-Length\": \"0\", \"Host\": \"httpbin.org\", \"Origin\": \"
→http://httpbin.org\", \"Referer\": \"htt
+HTTPCPOST:512,p://httpbin.org/\", \"User-Agent\": \"Mozilla/5.0 (X11; Linux_
→x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/
→537.36\", \"X-Amzn-Trace-Id\": \"Root=1-6150581e-1ad4bd5254b4bf5218070413\"} }
→\",
  "files": {},
  "form": {},
  "headers": {
    "Content-Length": "427",
    "Content-Type": "application/json",
    "Host": "httpbin.org",
    "User-Agent": "ESP32 HTTP Client/1.0",
    "X-Amzn-Trace-Id": "Root=1-61505e76-278b5c267aaf55842bd58b32"
  },
  "json": {
    "headers": {

+HTTPCPOST:512,"Accept": "application/json",
  "Accept-Encoding": "gzip, deflate",
  "Accept-Language": "en-US,en;q=0.9,zh-CN;q=0.8,zh;q=0.7",
  "Content-Length": "0",
  "Host": "httpbin.org",
  "Origin": "http://httpbin.org",
  "Referer": "http://httpbin.org/",
  "User-Agent": "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
→ like Gecko) Chrome/91.0.4472.114 Safari/537.36",
  "X-Amzn-Trace-Id": "Root=1-6150581e-1ad4bd5254b4bf5218070413"
  }
  },
  "origin": "20.187.154
+HTTPCPOST:45,.207",
  "url": "http://httpbin.org/post"
}

SEND OK

```

Note:

- After AT outputs the > character, the special characters in the HTTP body do not need to be escaped using the escape character, and the data do not need to end with a new line (CR-LF).

4.9.5 The PUT method of HTTP client (suitable for no data put)

In this example, the HTTP server is <http://httpbin.org>. PUT request supports [Query String Parameters](#) mode.

1. Restore factory default settings of the module.

Command:

```
AT+RESTORE
```

Response:

```
OK
```

2. Set the Wi-Fi mode to station.

Command:

```
AT+CWMODE=1
```

Response:

```
OK
```

3. Connect to the router.

Command:

```
AT+CWJAP="espressif","1234567890"
```

Response:

```
WIFI CONNECTED
WIFI GOT IP
OK
```

Note:

- The SSID and password you entered may be different from those in the above command. Please replace the SSID and password with those of your router settings.
4. Send an HTTP PUT request. Set opt to 4 (PUT method), url to <http://httpbin.org/put>, and transport_type to 1 (HTTP_TRANSPORT_OVER_TCP).

Command:

```
AT+HTTPCLIENT=4,0,"http://httpbin.org/put?user=foo",,,1
```

Response:

```
+HTTPCLIENT:282,{
  "args": {
    "user": "foo"
  },
  "data": "",
  "files": {},
  "form": {},
  "headers": {
    "Content-Length": "0",
    "Content-Type": "application/x-www-form-urlencoded",
    "Host": "httpbin.org",
    "User-Agent": "ESP32 HTTP Client/1.0",
    "X-Amzn-Trace-Id": "R
+HTTPCLIENT:140,oot=1-61503d41-1dd8cbe0056190f721ab1912"
  },
  "json": null,
  "origin": "20.187.154.207",
  "url": "http://httpbin.org/put?user=foo"
}
OK
```

Note:

- The results you obtain may be different from those in the above response.

4.9.6 The PUT method of HTTP client (recommended method)

In this example, the HTTP server is <http://httpbin.org> and the data type is `application/json`.

1. Restore factory default settings of the module.

Command:

```
AT+RESTORE
```

Response:

```
OK
```

2. Set the Wi-Fi mode to station.

Command:

```
AT+CWMODE=1
```

Response:

```
OK
```

3. Connect to the router.

Command:

```
AT+CWJAP="espressif","1234567890"
```

Response:

```
WIFI CONNECTED
WIFI GOT IP
```

```
OK
```

Note:

- The SSID and password you entered may be different from those in the above command. Please replace the SSID and password with those of your router settings.

4. PUT HTTP data of specified length. The command specifies that the number of HTTP header fields is 2, which are connection field and content-type field respectively. connection is keep-alive and content-type is application/json.

Assume the JSON data you want to post is of 427 bytes as follows.

```
{ "headers": { "Accept": "application/json", "Accept-Encoding": "gzip, deflate",
→ "Accept-Language": "en-US,en;q=0.9,zh-CN;q=0.8,zh;q=0.7", "Content-Length": "0
→", "Host": "httpbin.org", "Origin": "http://httpbin.org", "Referer": "http://
→httpbin.org/", "User-Agent": "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.
→36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.36", "X-Amzn-Trace-Id
→": "Root=1-6150581e-1ad4bd5254b4bf5218070413"} }
```

Command:

```
AT+HTTPCPUT="http://httpbin.org/put",427,2,"connection: keep-alive","content-
→type: application/json"
```

Response:

```
OK
```

```
>
```

This response indicates that AT is ready for receiving serial data. You should enter the 427-byte data mentioned above, and when the data length reaches the <length> value, data transmission starts.

```
+HTTPCPUT:281,{
  "args": {},
  "data": "{ \"headers\": { \"Accept\": \"application/json\", \"Accept-Encoding\
→\": \"gzip, deflate\", \"Accept-Language\": \"en-US,en;q=0.9,zh-CN;q=0.8,zh;
→q=0.7\", \"Content-Length\": \"0\", \"Host\": \"httpbin.org\", \"Origin\": \
→\"http://httpbin.org\", \"Referer\": \"htt
+HTTPCPUT:512,p://httpbin.org/\", \"User-Agent\": \"Mozilla/5.0 (X11; Linux x86_
→64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.114 Safari/537.
→36\", \"X-Amzn-Trace-Id\": \"Root=1-6150581e-1ad4bd5254b4bf5218070413\"} }",
  "files": {},
  "form": {},
  "headers": {
```

(continues on next page)

(continued from previous page)

```

    "Content-Length": "427",
    "Content-Type": "application/json",
    "Host": "httpbin.org",
    "User-Agent": "ESP32 HTTP Client/1.0",
    "X-Amzn-Trace-Id": "Root=1-635f7009-681be2d5478504dc5b83624a"
  },
  "json": {
    "headers": {
+HTTPCPUT:512,"Accept": "application/json",
      "Accept-Encoding": "gzip, deflate",
      "Accept-Language": "en-US,en;q=0.9,zh-CN;q=0.8,zh;q=0.7",
      "Content-Length": "0",
      "Host": "httpbin.org",
      "Origin": "http://httpbin.org",
      "Referer": "http://httpbin.org/",
      "User-Agent": "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
→ like Gecko) Chrome/91.0.4472.114 Safari/537.36",
      "X-Amzn-Trace-Id": "Root=1-6150581e-1ad4bd5254b4bf5218070413"
    }
  },
  "origin": "52.246.135"
+HTTPCPUT:43,.57",
  "url": "http://httpbin.org/put"
}

SEND OK

```

Note:

- After AT outputs the > character, the special characters in the HTTP body do not need to be escaped using the escape character, and the data do not need to end with a new line (CR-LF).

4.9.7 The DELETE method of HTTP client

In this example, the HTTP server is <http://httpbin.org>. The DELETE method is used to delete resources on a server. The exact use of DELETE requests depends on the server implementation.

1. Restore factory default settings of the module.

Command:

```
AT+RESTORE
```

Response:

```
OK
```

2. Set the Wi-Fi mode to station.

Command:

```
AT+CWMODE=1
```

Response:

```
OK
```

3. Connect to the router.

Command:

```
AT+CWJAP="espressif","1234567890"
```

Response:

```
WIFI CONNECTED
WIFI GOT IP

OK
```

Note:

- The SSID and password you entered may be different from those in the above command. Please replace the SSID and password with those of your router settings.
4. Send an HTTP DELETE request. Set `opt` to 5 (DELETE method), `url` to `http://httpbin.org/delete`, and `transport_type` to 1 (HTTP_TRANSPORT_OVER_TCP).

Command:

```
AT+HTTPCLIENT=5,0,"https://httpbin.org/delete",,,1
```

Response:

```
+HTTPCLIENT:282,{
  "args": {},
  "data": "",
  "files": {},
  "form": {},
  "headers": {
    "Content-Length": "0",
    "Content-Type": "application/x-www-form-urlencoded",
    "Host": "httpbin.org",
    "User-Agent": "ESP32 HTTP Client/1.0",
    "X-Amzn-Trace-Id": "Root=1-61504289-468a41
+HTTPCLIENT:114,737b0d251672acec9d"
  },
  "json": null,
  "origin": "20.187.154.207",
  "url": "https://httpbin.org/delete"
}

OK
```

Note:

- The results you obtain may be different from those in the above response.

4.10 WebSocket AT Examples

This document shows how to use *WebSocket AT Commands* on ESP32 with detailed examples.

- [WebSocket Connection over TCP](#)
- [WebSocket Connection over TLS \(Mutual Authentication\)](#)

Important: The default AT firmware does not support the WebSocket functionality. If you need ESP32 to support WebSocket functionality, please refer to [introduction](#).

4.10.1 WebSocket Connection over TCP

1. Set the Wi-Fi mode to station.

Command:

```
AT+CWMODE=1
```

Response:

```
OK
```

2. Connect to the router.

Command:

```
AT+CWJAP="espressif","1234567890"
```

Response:

```
WIFI CONNECTED
WIFI GOT IP
OK
```

Note:

- The SSID and password you entered may be different from those in the above command. Please replace the SSID and password with those of your router settings.
3. Connect the PC to the same router which ESP32 is connected to.
 4. Create a WebSocket server over TCP on PC using python. The example code is as follows:

```
#!/usr/bin/env python

import asyncio
from websockets.server import serve

host = "192.168.200.249"
port = 8765

async def echo(websocket):
    async for message in websocket:
        await websocket.send(message)

async def main():
    async with serve(echo, host, port):
        print(f"Server started at ws://{host}:{port}")
        await asyncio.get_running_loop().create_future() # run forever

asyncio.run(main())
```

Please modify the code above by replacing the host with the IP address of your PC, and save it as ws-server.py file. Then run the program.

```
python ws-server.py
```

Note:

- If you do not have the websockets library installed on your PC, please install it using `pip install websockets`.
 - The WebSocket server port is set to 8765 by default, but you can also set it to a different port.
5. Connect ESP32 device as a client to the WebSocket server.

Command:

```
AT+WSOPEN=0,"ws://192.168.200.249:8765"
```

Response:

```
+WS_CONNECTED:0
OK
```

6. Send 4 bytes of data.

Command:

```
AT+WSEND=0,4
```

Response:

```
OK
>
```

Input 4 bytes (e.g., `test`), and then AT will respond with the following message.

```
SEND OK
```

Note:

- If the input exceeds the length (n) specified by `AT+CIPSEND`, the system will respond with `busy p. . .`, and send the first n bytes. Once those bytes are sent, the system will respond with `SEND OK`.
7. Receive 4 bytes of data.
Since the above WebSocket server is an echo server, after the data is sent, the server will also return the data as it is, so AT will output:

```
+WS_DATA:0,4,test
```

4.10.2 WebSocket Connection over TLS (Mutual Authentication)

1. Set the Wi-Fi mode to station.

Command:

```
AT+CWMODE=1
```

Response:

```
OK
```

2. Connect to the router.

Command:

```
AT+CWJAP="espressif","1234567890"
```

Response:

```
WIFI CONNECTED
WIFI GOT IP
OK
```

Note:

- The SSID and password you entered may be different from those in the above command. Please replace the SSID and password with those of your router settings.
3. Set the SNTP server.

Command:

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

Response:

```
OK
```

Note:

- You can set the SNTP server according to your country's time zone.
4. Query the SNTP time.

Command:

```
AT+CIPSNTPTIME?
```

Response:


```
+CIPSNTPTIME:Mon Oct 18 20:12:27 2021
OK
```

Note:

- You can verify if the SNTP server is functioning correctly by checking whether the SNTP-synchronized time matches the actual current time.
 - The purpose of setting the time is to verify the validity period of the certificates during TLS authentication.
5. Connect the PC to the same router which ESP32 is connected to.
 6. Get the CA, certificate, and private key of the WebSocket server.

You can use the `openssl` tool to generate the CA, certificate, and private key. If you encounter any difficulties, consider using the following configuration for testing:

wss_ca.crt

```
-----BEGIN CERTIFICATE-----
MIIDNjCCAh6gAwIBAgIUDe6T+Pu0BqmmTjw3s4snVRNFicMwDQYJKoZIhvcNAQEL
BQAwNjELMAkGA1UEBhMCQ04xFTATBgNVBAoMDEVtUUFJFU1NJRiBBVDEQMA4GA1UE
AwWHU9vdCBDQTAeFw0yNDA5MTkwOTMzNDBaFw0zNDA5MTcwOTMzNDBaMDYxCzAJ
BgNVBAYTAkNOMRUwEwYDVQKDAxFTU1BSRVNTSUyGQVQxEDA0BgNVBAMMB1Jvb3Qg
Q0EwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCylQstJGNFHLf7F+gG
oMSZSrWV4+/5Qxw1Cw3y5N1OVkMSxppYHVxj6MbOwWoCqQcx/WMtqnhg9ATDiZOE
bXFVB0aZd6EEBz24k7UvQ1ilfIw5tzmjl8SSbe7CiFq302WVokBVFhSeY2jrG+sI
6PWg0WsvxzierDL9hef708IJERlX0ScBIslZnVU4UBPtrbG2bgl3L6T5iQ95tSEX
LsmfZ4l+/Q1xSC7VGH1K1WGBnUzGpv9vLc9uFGZVcAEKx1V9Y7DsyJvTosf00Mmb
yUicBk5nVCHrfhmrfrAWWD9YaNc0gMPVpO6cHzGd/Fgw6v06QshMYUOE6wxpVjb
8JY1AgMBAAGjPDA6MAwGA1UdEwQFMAMBAf8wCwYDVR0PBAQDAgGmMB0GA1UdDgQW
BBRdPJ7Nq+WXSiLn4ZLWJlQQfjrm7zANBgkqhkiG9w0BAQsFAAOCAQEANYpErE2L
IpISbzITvG6A67YmYMyadWSNGQ2VjdLK2s3zkggF96bIZziygOa9mgAKD/yTw10t
V0xF1GUDz43DdtJZ1wxo8FPrxH41cP63vjtp28+ZNkylv9Hrx8De2JjiYwRpZmlQY
EHwDEFpK26LdwPPalwdMZSyMzCtRNJ+o8Bk5kSc3V9QJmh9lLe4PdfCJcfzwPskC
dNgUMEcRa4k3VFOYUumyFofG8/kINcRogfPzZuUE20j1wCHqSpOaP2CJN9QTK0q
Fn0Itrjikv6z1aTp+SJPOzRPbympL9KhhhhcYJQdPaXHWYfDhEU/abnnJQpd+1Y
HPnBCb4tK/pS9w==
-----END CERTIFICATE-----
```

wss_server.crt

```
-----BEGIN CERTIFICATE-----
MIIDVTCCAj2gAwIBAgIUPLhviJh1UJDQ5Md6tHibK11jP24wDQYJKoZIhvcNAQEL
BQAwNjELMAkGA1UEBhMCQ04xFTATBgNVBAoMDEVtUUFJFU1NJRiBBVDEQMA4GA1UE
AwWHU9vdCBDQTAeFw0yNDA5MTkwOTMzNDBaFw0zNDA5MTcwOTMzNDBaMDQxCzAJ
BgNVBAYTAkNOMRUwEwYDVQKDAxFTU1BSRVNTSUyGQVQxZjAMBgNVBAMMBUwxIENB
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAtKALI+zRbUamJukAi2ai
dzPdNdy+Tuv2K+GyB15yomIr0e1c1pfztk68zdHKcFFt0RQfz1GYtxrZzqv07o4t
K9ijfAw+k498SctmTqHwLUKc7B9mK6UZfZSkXUnufXKE5+N96u+49e3wbk7yoNfQ
kzdtwXUtM6ee5w7HvjwdKQcJXYWv/c/vZLWmAUg3AEaUknS8r4LDg/X+L/bxN+9
zycLL5tGgZg22KENW4QWwsUY6ntuKQDlohiNxi+wXyM3mVNOc94umICj7a10hPst
UmuLYD/gUCnM4JRvQYAVmPQC88K0Lj7aIwJedQ7TJwhJGDS6WDVEMeRfJvXswUk
4QIDAQAB010wWzAMBgNVHRMEBTADAQH/MASGA1UdDwQEAwIBpjAdBgNVHQ4EFgQU
e1wZbnvJnZr8js60EyIWip6hRFawHwYDVR0jBBgwFoAUNrQtNxD0tapUczc8Mwln
hPZE84YwDQYJKoZIhvcNAQELBQADggEBAICnosdYBc6enaKB77AWXFzMWyDDLvd5
JSJa1IRJ+Rhs1gBxjIH66/+6QyZcJu5C/RZ+RZ04Mky3nMndc3kSCFqauBCK9xoG
/fZ5wrAfH54o4P+3ZliGoK8EltKulHuQYJTW8JKbLWCRJ8PnGkrDZBgSXgn8tyL4
U3hu1MdmkH//fiV0z6itfyJcZdu387l8bGBhrAD3Z7gWcdIJbXxmu6m7FJadUe/N
0vsLtgkOpOyUa2rThOnJpSyXB5QnKifRGYjuPrnOIWtmADYQRUEd2zdggRUUuYJR
ee4Vz2BFXhpZdGeD3bVAop+/YebTa0iDxXSLWkPLQfCyIkdtPXmKQPQ=
-----END CERTIFICATE-----
```

wss_server.key

```
-----BEGIN PRIVATE KEY-----
MIIEvAIBADANBgkqhkiG9w0BAQEFAASCBKYYggSiAgEAAoIBAQC0oAsj7NftRoyO
6QCLZqJ3M9013L5NS/Yr4bIHxNKiYivR7VzWl/O2TrzN0cpwUW3RFB/PUzi3GtnO
q87uji0r2KMUDD6Tj3xIJOZOofCVQpzsh2YrpRl9lKRdSe59coTn433q77j17fBu
```

(continues on next page)

(continued from previous page)

```

TvKg19CTN23DBdS0zp57nDse+PB0pBwldha/9z/NutaYBQbcARpSSdLyvgt0b9f4
v9vE373PJwsvm0aBmDbYoQ1bhBaxJRjqe24pAOWiGI3GL7BfIzeZU05z3i6YgKPt
rU6E+y1Sa4tgP+BQKczglG9BgBWY9AKLzwqguPtoJAl51DtMnCEkYNLpYNUQx5F8
lVezBSThAgMBAAECgEASkRF4FsWjyJDV91Y4nhsU6vpCCT/wCN8D/XoL9x3MOpB
jzrUAc4PoIWGXvAkFwN8LkviemlX6+2n4bDF0FN4Ij+caflQ33ZPSRCW+3zdQVnW
0MvmSorDTN3JqSvlWgI0wG3Kz8cKW2AejBR88YJbGbTgNiBXIZKVGkkWC/maULKa
L2Omd5ZzF9qkWWoBk0GxrlME+V9726L6SJHIVESkPcbojXZPM06zmz84+Zzf0UdG
pl+SocmQzo8yzhjXR8UHxtzfZbiiJZGCsWixvizqM4OTarfiBPhmmSNwp07Vk3lD
t6XISALqMbIHv/co5LOWs5WnyQBud28MV7RpCD/tawKBgQDnGuF5milpMR6KqRrd
O8amarseklND9NiWsUcXLYYHQkNC1Ttr5Gqtm3EbJbR0wLwUzL7PwgeuNmN6e6vz
IreS7wd+uO3HeVVsFPeOpTlXzALMG6wuYN/Ipfe/T8v/vq9CkoZ6VDaDN6OdeBAq
g6pAUQFL+uamLfrKcCiZtiOR/wKBgQDIFRhC5MyLaFlw3XumgAsh37tmlGOQ/T+K
GAWXiRhXW340EVEzNUUXeJcGig4BgNVwXBvjYTk0unyIw72bUNrIeOqp4ZgO2vW
NUgyDgIGKZuIIGI51FeezG6Qu2s93qZTYtdjH0M4ISKgrKOvY51fuXBNFSu7BOq
i5MQamuJHwKBgHGJhiszq6aPSCbtH1KTHGwDwXwqfRfEwWd/HqLnbZJBXpPmhvPh
mvtBg5bHtlkpmv1I/XFKLXM2KCDA54Gb1OTdQYvyjmWhX386wY8a+iTRMiLy9JZ
K3gC+aOWge1Z+/Zj0AdnOgTLH+14y8hnOQwx/8YZNJltu2kbIwcpMV53AoGAFMzE
meepL/DoI2iS+ysifSICHFBexurc2SFIK4mwBgY3cX9Nrt6qZBSifIqnlbNiU17p
r18a6qLWeTqVyp5vPMroHQyPVp+2xS0C1VlJcpSOu6cKLxLZDQQZlmg1bNghmFeV
JpPQbBxdujBYT9peON5RQ2IpBNI/9SHPZwx5I2cCgYBdCmby0loBi5lCbCXKV5c3
mzlPpM6zR3aLYsFyknWrtLv3UPMGzOLsRDkwfQEMZWt/VFjXehajyqd8X7r3V0lX
cz5LwmHOMSKDYgdzrJjvE3lPxDDKtmyPr5nnF/8/SF6Lawj7v6eGnpiZNqHC/wkK
9EmCWXJc/9GVWSZtEOz/Q==
-----END PRIVATE KEY-----

```

7. Create a WebSocket server over TLS on PC using python. The example code is as follows:

```

#!/usr/bin/env python

import asyncio
import ssl
import websockets

host = '192.168.200.249'
port = 8766
wss_ca = '/your_path/wss_ca.crt'
wss_cert = '/your_path/wss_server.crt'
wss_key = '/your_path/wss_server.key'

async def echo(websocket, path):
    async for message in websocket:
        await websocket.send(message)

ssl_context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)
ssl_context.load_cert_chain(certfile=wss_cert, keyfile=wss_key)
ssl_context.verify_mode = ssl.CERT_REQUIRED
ssl_context.load_verify_locations(wss_ca)

start_server = websockets.serve(echo, host, port, ssl=ssl_context)

asyncio.get_event_loop().run_until_complete(start_server)
print(f"Server started at wss://{host}:{port}")

asyncio.get_event_loop().run_forever()

```

Please modify the code above by replacing the host with the IP address of your PC, and the `wss_ca`, `wss_cert`, and `wss_key` with the paths to the CA, certificate, and private key of the server. Save the modified code as `wss-server.py` and run the program.

```
python wss-server.py
```

Note:

- If you do not have the `websockets` library installed on your PC, please install it using `pip install`

websockets.

- The WebSocket server port is set to 8766, but you can also set it to a different port.
8. Configure the WebSocket connection for mutual authentication.

Command:

```
AT+WSCFG=0,30,60,4096,3,0,0
```

Response:

```
OK
```

Note:

- If you want to use your own certificates or use multiple sets of certificates, there are two methods:
 - Runtime update: You can directly use the [AT+SYSMFG](#) command to update certificates without recompiling the firmware. For detailed operation steps, please refer to [PKI Configuration in AT+SYSMFG command examples](#). The certificate configuration method is the same as SSL certificates, just change the namespace to `wss_cert`, `wss_key`, and `wss_ca`.
 - Pre-flash certificates: If you want to pre-flash certificates during compilation, please refer to [How to Update PKI Configuration](#).
9. Connect ESP32 device as a client to the WebSocket server.

Command:

```
AT+WSOOPEN=0,"wss://192.168.200.249:8766"
```

Response:

```
+WS_CONNECTED:0
```

```
OK
```

10. Send 4 bytes of data.

Command:

```
AT+WSENSEND=0,4
```

Response:

```
OK
```

```
>
```

Input 4 bytes, for example, `test`, then AT will respond the following message.

```
SEND OK
```

Note:

- If the number of bytes inputted are more than the length (n) set by `AT+CIPSEND`, the system will reply `busy p...`, and send the first n bytes. And after sending the first n bytes, the system will reply `SEND OK`.
11. Receive 4 bytes of data.
- Since the above WebSocket server is an echo server, after the data is sent, the server will also return the data as it is, so AT will output:

```
+WS_DATA:0,4,test
```

4.11 ESP32 Classic Bluetooth AT Examples

This document provides detailed command examples to illustrate how to utilize [ESP32 Classic Bluetooth® AT Commands](#) on ESP32.

- *Establish SPP connection between phone (or PC) and ESP32 in Normal Transmission mode with IO capability set to NoInputNoOutput*
- *Establish SPP connection between phone (or PC) and ESP32 in Passthrough mode with IO capability set to NoInputNoOutput*
- *Establish SPP connection between phone (or PC) and ESP32 with IO capability set to KeyboardOnly*
- *Establish SPP connection between two ESP32 development boards*
- *Establish A2DP connection and enable A2DP Sink to play music*
- *Query and clear Classic Bluetooth encryption device list*

Important: The default firmware does not support Classic Bluetooth commands. For how to enable support for those commands, please refer to [Introduction to Classic Bluetooth](#).

4.11.1 Establish SPP connection between phone (or PC) and ESP32 in Normal Transmission mode with IO capability set to NoInputNoOutput

In this example, mobile phone or PC is master and ESP32 is slave. The example shows how to establish SPP connection.

1. Classic Bluetooth initialization.

Command:

```
AT+BTINIT=1
```

Response:

```
OK
```

2. Classic Bluetooth SPP profile initialization, and the role is set to slave.

Command:

```
AT+BTSPPINIT=2
```

Response:

```
OK
```

3. Set Classic Bluetooth device name.

Command:

```
AT+BTNAME="EXAMPLE"
```

Response:

```
OK
```

4. Set Classic Bluetooth scan mode to discoverable and connectable.

Command:

```
AT+BTSCANMODE=2
```

Response:

```
OK
```

5. Set Classic Bluetooth security parameters. Set `io_cap` to NoInputNoOutput, `pin_type` to fixed, `pin_code` to 9527.

Command:

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

Response:

OK

6. Start Classic Bluetooth SPP profile.

Command:

AT+BTSPSTART

Response:

OK

7. The mobile phone or PC initiates the connection.

The mobile phone or PC should be able to find the Bluetooth device with name “EXAMPLE”. If the mobile phone or PC initiates a connection and the connection is established successfully, ESP32 will prompt:

+BTSPPCONN:0,"e0:24:81:47:90:bc"

Note:

- The address you obtained may be different from that in the above response.

8. Send 4 bytes of data.

Command:

AT+BTSPSEND=0,4

Response:

>

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

Input 4 bytes, for example, test, then AT will respond the following message.

OK

Note:

- If the number of bytes inputted are more than the length (n) set by AT+BTSPSEND, the system will reply busy p . . . , and send the first n bytes. And after sending the first n bytes, the system will reply OK.
- After AT outputs the > character, the special characters in the data does not need to be escaped through the escape character, and it does not need to end with a new line (CR-LF).

9. Receive 4 bytes of data.

Assume that mobile phone or PC sends 4 bytes of data (data is test), the system will prompt:

+BTDATA:4,test

10. Terminate Classic Bluetooth SPP connection.

Command:

AT+BTSPDISCONN=0

Response:

+BTSPDISCONN:0,"e0:24:81:47:90:bc"

OK

Note:

- The address you obtained may be different from that in the above response.

4.11.2 Establish SPP connection between phone (or PC) and ESP32 in Passthrough mode with IO capability set to NoInputNoOutput

In this example, mobile phone or PC is master and ESP32 is slave. The example shows how to establish SPP connection.

1. Classic Bluetooth initialization.

Command:

```
AT+BTINIT=1
```

Response:

```
OK
```

2. Classic Bluetooth SPP profile initialization, and the role is set to slave.

Command:

```
AT+BTSPPINIT=2
```

Response:

```
OK
```

3. Set Classic Bluetooth device name.

Command:

```
AT+BTNAME="EXAMPLE"
```

Response:

```
OK
```

4. Set Classic Bluetooth scan mode to discoverable and connectable.

Command:

```
AT+BTSCANMODE=2
```

Response:

```
OK
```

5. Set Classic Bluetooth security parameters. Set `io_cap` to `NoInputNoOutput`, `pin_type` to `fixed`, `pin_code` to `9527`.

Command:

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

Response:

```
OK
```

6. Start Classic Bluetooth SPP profile.

Command:

```
AT+BTSPSTART
```

Response:

```
OK
```

7. The mobile phone or PC initiates the connection.

The mobile phone or PC should be able to find the Bluetooth device with name “EXAMPLE”. If the mobile phone or PC initiates a connection and the connection is established successfully, ESP32 will prompt:

```
+BTSPPCONN:0,"e0:24:81:47:90:bc"
```

Note:

- The address you obtained may be different from that in the above response.

8. Send data in Passthrough Mode.

Command:

```
AT+BTSPSEND
```

Response:

```
OK
```

```
>
```

This response indicates that AT has entered Passthrough Mode.

Note:

- After the AT enters Passthrough Mode, data received from serial port will be transmitted to the mobile phone or PC.

9. Stop sending data.

When receiving a packet that contains only +++, the Passthrough Mode will be stopped. Then please wait at least 1 second before sending next AT command. Please be noted that if you input +++ directly by typing, the +++ may not be recognised as three consecutive + because of the prolonged typing duration. For more details, please refer to [AT+BTSPSEND](#).

Important: The aim of ending the packet with +++ is to exit Passthrough Mode and to accept normal AT commands. However, you can also use command AT+BTSPSEND to go back into Passthrough Mode.

10. Terminate Classic Bluetooth SPP connection.

Command:

```
AT+BTSPDISCONN=0
```

Response:

```
+BTSPDISCONN:0,"e0:24:81:47:90:bc"
```

```
OK
```

Note:

- The address you obtained may be different from that in the above response.

4.11.3 Establish SPP connection between phone (or PC) and ESP32 with IO capability set to KeyboardOnly

The process is basically the same as in the [Establish SPP connection between phone \(or PC\) and ESP32 in Normal Transmission mode with IO capability set to NoInputNoOutput](#). The only difference lies in security parameters settings.

1. Classic Bluetooth initialization.

Command:

```
AT+BTINIT=1
```

Response:

```
OK
```

2. Classic Bluetooth SPP profile initialization, and the role is set to slave.

Command:

```
AT+BTSPINIT=2
```

Response:

```
OK
```

3. Set Classic Bluetooth device name.

Command:

```
AT+BTNAME="EXAMPLE"
```

Response:

```
OK
```

4. Set Classic Bluetooth scan mode to discoverable and connectable.

Command:

```
AT+BTSCANMODE=2
```

Response:

```
OK
```

5. Set Classic Bluetooth security parameters. Set `io_cap` to `KeyboardOnly`, `pin_type` to `variable`, `pin_code` to `9527`.

Command:

```
AT+BTSECPARAM=2,0,"9527"
```

Response:

```
OK
```

6. Start Classic Bluetooth SPP profile.

Command:

```
AT+BTSPSTART
```

Response:

```
OK
```

7. The mobile phone or PC initiates the connection.

The mobile phone or PC can initiate a connection and generate a PIN code, then you can enter the PIN code on the ESP32.

```
AT+BTKEYREPLY=0,676572
```

If the connection is established successfully, the system will prompt:

```
+BTSPPCONN:0,"e0:24:81:47:90:bc"
```

Note:

- The PIN code you entered may be different from those in the above command. Please use the real PIN instead.
- The address you obtained may be different from that in the above response.

8. Terminate Classic Bluetooth SPP connection.

Command:

```
AT+BTSPDISCONN=0
```

Response:

```
+BTSPDISCONN:0,"e0:24:81:47:90:bc"
```

```
OK
```

Note:

- The address you obtained may be different from that in the above response.

4.11.4 Establish SPP connection between two ESP32 development boards

Below is an example of using two ESP32 development boards, one as master, the other one as slave.

Important: In the following steps, the operations starting with `Master` only need to be executed at master, and the operations starting with `Slave` only need to be executed at slave. If the operation is not specified on which side

it is executed, it needs to be executed on both the master side and the slave side.

1. Classic Bluetooth initialization.

Command:

```
AT+BTINIT=1
```

Response:

```
OK
```

2. Classic Bluetooth SPP profile initialization.

Master:

Command:

```
AT+BTSPPINIT=1
```

Response:

```
OK
```

Slave:

Command:

```
AT+BTSPPINIT=2
```

Response:

```
OK
```

3. Set Classic Bluetooth device name.

Slave:

Command:

```
AT+BTNAME="EXAMPLE"
```

Response:

```
OK
```

4. Set Classic Bluetooth scan mode to discoverable and connectable.

Slave:

Command:

```
AT+BTSCANMODE=2
```

Response:

```
OK
```

5. Set Classic Bluetooth security parameters. Set `io_cap` to `NoInputNoOutput`, `pin_type` to `fixed`, `pin_code` to `9527`.

Slave:

Command:

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

Response:

```
OK
```

6. Start Classic Bluetooth SPP profile.

Slave:

Command:

```
AT+BTSPPSTART
```

Response:

OK

7. Start Classic Bluetooth discovery. Set inquiry duration to 10 s, number of inquiry responses to 10.

Master:

Command:

AT+BTSTARTDISC=0,10,10

Response:

```
+BTSTARTDISC:"10:f6:05:f9:bc:4f",realme V11 5G,0x2,0x3,0x2d0,-34
+BTSTARTDISC:"24:0a:c4:d6:e4:46",EXAMPLE,,,,-27
+BTSTARTDISC:"10:f6:05:f9:bc:4f",realme V11 5G,0x2,0x3,0x2d0,-33
+BTSTARTDISC:"24:0a:c4:d6:e4:46",EXAMPLE,,,,-25
+BTSTARTDISC:"ac:d6:18:47:0c:ae",,0x2,0x3,0x2d0,-72
+BTSTARTDISC:"24:0a:c4:d6:e4:46",EXAMPLE,,,,-26
+BTSTARTDISC:"10:f6:05:f9:bc:4f",,0x2,0x3,0x2d0,-41
+BTSTARTDISC:"24:0a:c4:2c:a8:a2",,,,,-50
+BTSTARTDISC:"24:0a:c4:d6:e4:46",EXAMPLE,,,,-26
+BTSTARTDISC:"10:f6:05:f9:bc:4f",realme V11 5G,0x2,0x3,0x2d0,-39
+BTSTARTDISC:"24:0a:c4:d6:e4:46",EXAMPLE,,,,-23
+BTSTARTDISC:"10:f6:05:f9:bc:4f",realme V11 5G,0x2,0x3,0x2d0,-36
+BTSTARTDISC:"10:f6:05:f9:bc:4f",realme V11 5G,0x2,0x3,0x2d0,-41
+BTSTARTDISC:"b4:a5:ac:16:14:8c",,0x2,0x3,0x2d0,-57
+BTSTARTDISC:"24:0a:c4:2c:a8:a2"
+BTSTARTDISC:"b4:a5:ac:16:14:8c"
```

OK

Note:

- The discovery results you obtain may be different from those in the above response.

8. Establish SPP connection.

Master:

Command:

AT+BTSPPCONN=0,0,"24:0a:c4:d6:e4:46"

Response:

+BTSPPCONN:0,"24:0a:c4:d6:e4:46"

OK

Note:

- When entering the above command, replace the address with slave address.
- If the SPP connection is established successfully, message +BTSPPCONN:0,"30:ae:a4:80:06:8e" will be prompted on the slave.

9. Terminate Classic Bluetooth SPP connection.

Slave:

Command:

AT+BTSPDISCONN=0

Response:

+BTSPDISCONN:0,"30:ae:a4:80:06:8e"

OK

Note:

- Both master and slave can actively terminate the SPP connection.
- If the SPP connection is ended successfully, message +BTSPDISCONN:0,"24:0a:c4:d6:e4:46" will be prompted on the master.

4.11.5 Establish A2DP connection and enable A2DP Sink to play music

Important:

- To use A2DP Sink, you need to add the code of the I2S part by yourself. For the code to initialize the I2S part, please refer to [a2dp sink example](#).
 - The driver code of the decoder chip part also needs to be added by yourself or use an off-the-shelf development board.
-

1. Classic Bluetooth initialization.

Command:

```
AT+BTINIT=1
```

Response:

```
OK
```

2. Classic Bluetooth A2DP profile initialization, and the role is set to sink.

Command:

```
AT+BTA2DPINIT=2
```

Response:

```
OK
```

3. Set Classic Bluetooth device name.

Command:

```
AT+BTNAME="EXAMPLE"
```

Response:

```
OK
```

4. Set Classic Bluetooth scan mode to discoverable and connectable.

Command:

```
AT+BTSCANMODE=2
```

Response:

```
OK
```

5. Establish connection.

The source role should be able to find the Bluetooth device with name “EXAMPLE”. In this example, you can use your mobile phone to initiate a connection. If the connection is established successfully, ESP32 will prompt:

```
+BTA2DPCONN:0,"e0:24:81:47:90:bc"
```

Note:

- The address you obtained may be different from that in the above response.

6. Start playing music.

Command:

```
AT+BTA2DPCTRL=0,1
```

Response:

```
OK
```

Note:

- For more types of control, please refer to [AT+BTA2DPCTRL](#).

7. Stop playing music.

Command:

```
AT+BTA2DPCTRL=0,0
```

Response:

```
OK
```

Note:

- For more types of control, please refer to [AT+BTA2DPCTRL](#).

8. Terminate A2DP connection.

Command:

```
AT+BTA2DPDISCONN=0
```

Response:

```
OK
+BTA2DPDISCONN:0,"e0:24:81:47:90:bc"
```

4.11.6 Query and clear Classic Bluetooth encryption device list

1. Get the encryption device list.

Command:

```
AT+BTENCDEV?
```

Response:

```
+BTA2DPDISCONN:0,"e0:24:81:47:90:bc"
OK
```

Note:

- If no device has been successfully bound before, AT will only prompt OK.

2. Clear Classic Bluetooth encryption device list.

There are two ways to clear encryption device list.

1. Remove a device from the encryption device list with a specific index.

Command:

```
AT+BTENCCLEAR=0
```

Response:

```
OK
```

2. Remove all devices from the encryption device list.

Command:

```
AT+BTENCCLEAR
```

Response:

```
OK
```

4.12 Sleep AT Examples

This document provides an introduction and detailed command examples to illustrate how to utilize AT commands to set sleep modes on ESP32 series of products.

- [Introduction](#)
- [Set Modem-sleep mode in Wi-Fi mode](#)
- [Set Light-sleep mode in Wi-Fi mode](#)
- [Set Light-sleep mode in Bluetooth LE advertising mode](#)
- [Set Light-sleep mode in Bluetooth LE connection mode](#)
- [Set Deep-sleep mode](#)

4.12.1 Introduction

With the use of advanced power-management technologies, ESP32 series can switch between different power modes. Currently, ESP-AT supports the following four power consumption modes (for more sleep modes, please refer to the datasheet):

1. **Active Mode:** CPU and chip radio are powered on. The chip can receive, transmit, or listen.
2. **Modem-sleep Mode:** The CPU is operational and the clock speed can be reduced. Wi-Fi baseband, Bluetooth LE baseband, and radio are disabled, but Wi-Fi and Bluetooth LE connection can remain active.
3. **Light-sleep Mode:** The CPU is paused. Any wake-up events (MAC, host, RTC timer, or external interrupts) will wake up the chip. Wi-Fi and Bluetooth LE connection can remain active.
4. **Deep-sleep Mode:** CPU and most peripherals are powered down. Only the RTC memory is powered on.

By default, ESP32 will enter **Active** mode after system reset. When the CPU does not need to work all the time, such as waiting for external activities to wake up, the system can enter low-power modes.

For current consumption of ESP32, please refer to [ESP32 Series Datasheet](#).

Note:

- Setting ESP32 to sleep modes in Wi-Fi mode and Bluetooth LE mode will be described separately.
 - In single Wi-Fi mode, only **station** mode supports **Modem-sleep** mode and **Light-sleep** mode.
 - In single Bluetooth mode, when operating in **Light-sleep** mode, the power consumption can be reduced to the μ A level if an external 32 kHz crystal oscillator is configured. If no external 32 kHz crystal is present, the system automatically uses the internal main crystal oscillator as the RTC clock source, and the power consumption is approximately several mA.
-

Measurement Method

In order to avoid some unnecessary interference during the power consumption test process, it is recommended to use the Espressif modules that integrate the chip for the test.

Refer to the following figure for hardware connection. (Note that the development board below only has the ESP32 module on board, and all other peripheral components have been removed.)

4.12.2 Set Modem-sleep mode in Wi-Fi mode

1. Set the Wi-Fi mode to station mode.

Command:

```
AT+CWMODE=1
```

Response:

```
OK
```

2. Connect to an router.

Command:

```
AT+CWJAP="espressif","1234567890"
```

Response:

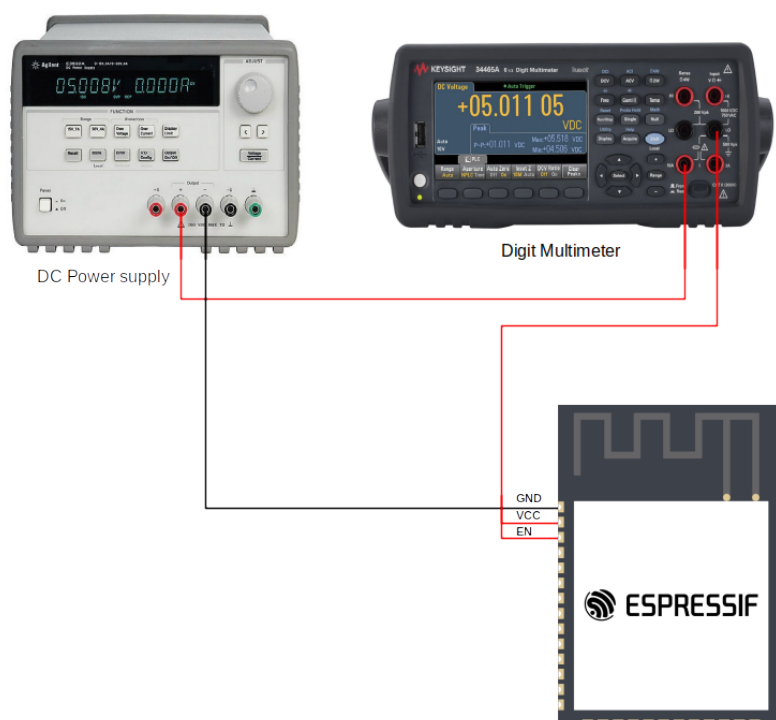


Fig. 22: ESP32 Hardware Connection

```
WIFI CONNECTED
WIFI GOT IP
OK
```

Note:

- The SSID and password you entered may be different from those in the above command. Please replace the SSID and password with those of your router settings.
3. Set the sleep mode to Modem-sleep mode.

Command:

```
AT+SLEEP=1
```

Response:

```
OK
```

Note: RF will be periodically closed according to AP DTIM (routers generally set DTIM to 1).

4.12.3 Set Light-sleep mode in Wi-Fi mode

1. Set the Wi-Fi mode to station mode.

Command:

```
AT+CWMODE=1
```

Response:

```
OK
```

2. Connect to an router. Set listen interval to 3.

Command:

```
AT+CWJAP="espressif","1234567890",,,3
```

Response:

```
WIFI CONNECTED
```

```
WIFI GOT IP
```

```
OK
```

Note:

- The SSID and password you entered may be different from those in the above command. Please replace the SSID and password with those of your router settings.

3. Set the sleep mode to Light-sleep mode.

Command:

```
AT+SLEEP=2
```

Response:

```
OK
```

Note: CPU will automatically sleep and RF will be periodically closed according to listen interval set by [AT+CWJAP](#).

4.12.4 Set Light-sleep mode in Bluetooth LE advertising mode

1. Initialize the role of Bluetooth LE as server.

Command:

```
AT+BLEINIT=2
```

Response:

```
OK
```

2. Set parameters of Bluetooth LE advertising. Set Bluetooth LE advertising interval to 1 s.

Command:

```
AT+BLEADVPARAM=1600,1600,0,0,7,0,0,"00:00:00:00:00:00"
```

Response:

```
OK
```

3. Start Bluetooth LE advertising.

Command:

```
AT+BLEADVSTART
```

Response:

```
OK
```

4. Disable Wi-Fi.

Command:

```
AT+CWINIT=0
```

Response:

```
OK
```

5. Set the sleep mode to Light-sleep mode.

Command:

```
AT+SLEEP=2
```

Response:

```
OK
```

4.12.5 Set Light-sleep mode in Bluetooth LE connection mode

1. Initialize the role of Bluetooth LE as server.

Command:

```
AT+BLEINIT=2
```

Response:

```
OK
```

2. Start Bluetooth LE advertising.

Command:

```
AT+BLEADVSTART
```

Response:

```
OK
```

3. Waiting for connection.

If the connection is established successfully, AT will prompt:

```
+BLECONN:0,"47:3f:86:dc:e4:7d"  
+BLECONNPARAM:0,0,0,6,0,500  
+BLECONNPARAM:0,0,0,24,0,500  
  
OK
```

Note:

- In this example, Bluetooth LE client address is 47:3f:86:dc:e4:7d.
- For prompt information (+BLECONN and +BLECONNPARAM), please refer to [AT+BLECONN](#) and [AT+BLECONNPARAM](#) for more details.

4. Update parameters of Bluetooth LE connection. Set Bluetooth LE connection interval to 1 s.

Command:

```
AT+BLECONNPARAM=0,800,800,0,500
```

Response:


```
OK
```

If the connection parameters are updated successfully, AT will output:

```
+BLECONNPARAM:0,800,800,800,0,500
```

Note:

- For prompt information (+BLECONNPARAM), please refer to [AT+BLECONNPARAM](#) for more details.

5. Disable Wi-Fi.

Command:

```
AT+CWINIT=0
```

Response:

```
OK
```

6. Set the sleep mode to Light-sleep mode.

Command:

```
AT+SLEEP=2
```

Response:

```
OK
```

4.12.6 Set Deep-sleep mode

1. Set the sleep mode to Deep-sleep mode. Set the deep-sleep time to 3600000 ms.

Command:

```
AT+GSLP=3600000
```

Response:

```
OK
```

Note:

- When the time is up, the device automatically wakes up, calls Deep-sleep wake stub, and then proceeds to load the application.
- For Deep-sleep mode, the only wake-up method is timed wake-up.

4.13 AT Network Provisioning Examples

This document mainly introduces the following network provisioning methods supported by ESP-AT:

- [BluFi Provisioning](#)
- [SmartConfig Provisioning](#)
- [SoftAP Provisioning \(WEB Provisioning\)](#)
- [WPS Provisioning](#)

Table 1: Summary of Key Parameters for Provisioning Methods

Provisioning Method	BluFi Provisioning	Provisioning	SmartConfig Provisioning	SoftAP Provisioning	Provisioning	WPS Provisioning
Default Firmware Support	✓		✓	✗ (Note 1)		✓
Requires BLE	✓		✗	✗		✗
Requires Additional Mobile App	✓		✓ (Note 2)	✓		✗
Provisioning Success Rate	High		Moderate	High		High
Operation Complexity	Simple		Simple	Complex		Complex
Recommendation Level	Recommended		Moderate	Moderate		Moderate

- Note 1: If you want to enable SoftAP provisioning in the AT firmware, please refer to: [Web Server AT Commands](#).
- Note 2: The AirKiss provisioning in SmartConfig requires the WeChat app, which may be unavailable to users outside China.

4.13.1 BluFi Provisioning

BluFi is a Wi-Fi configuration feature based on the Bluetooth channel. For more information, please refer to the [BluFi documentation](#).

1. Install the EspBluFi app on your mobile device.
 - Android: [Download EspBluFi for Android \(Android Source Code\)](#)
 - iOS: [Download EspBluFi for iOS \(iOS Source Code\)](#)
2. Set the BluFi name on the ESP32 device.

Command:

```
AT+BLUFINAME="blufi_test"
```

Response:

```
OK
```

3. Enable BluFi on the ESP32 device.

Command:

```
AT+BLUFI=1
```

Response:

```
OK
```

4. Establish BluFi connection on your mobile device and perform Wi-Fi provisioning.

4.1 BluFi Connection

Enable the system's Bluetooth and GPS location services on your mobile device, then launch the EspBluFi app. Find the device named `blufi_test` and click it to enter. Next, click `Connect` to establish a connection. At this point, the ESP device should log something similar to `+BLU-FICONN`, and the app will display the below information, indicating that the BluFi connection has been successfully established.

```
Connected <mac>

Discover service and characteristics success

Set notification enable complete

Set mtu complete, mtu=...
```

4.2 BluFi Wi-Fi Provisioning

After the connection is established, click **Configure** button in the EspBluFi app. The page will navigate to **Configure** page. Enter the Wi-Fi SSID and password, then click **OK** button to start provisioning. After a short wait, the app will display the below information.

```
Post configure params complete

Receive device status response:
OpMode: Station
Station connect Wi-Fi now
Station connect Wi-Fi bssid: <mac>
Station connect Wi-Fi ssid: <ssid>
```

4.3 Successful Provisioning

Please wait for a moment, and the ESP device will output the following logs. At this point, the ESP device's network configuration is successfully completed.

```
WIFI CONNECTED
WIFI GOT IP
```

5. (Optional) Send custom user data via BluFi on the ESP32 device.

Command:

```
AT+BLUFISEND=4
```

Response:

```
>
```

Input 4 bytes of data, for example, if the input data is `test`, then AT will output the following information.

```
OK
```

At this point, the app will display the below information:

```
Receive custom data:
test
```

6. Disable BluFi on the ESP32 device.

Command:

```
AT+BLUFI=0
```

Response:

```
+BLUFIDISCONN

OK
```

At this point, the app will display the below information:

```
Disconnect <mac>, status=19
```

4.13.2 SmartConfig Provisioning

AT SmartConfig provisioning offers four types: ESP-TOUCH, AirKiss, ESP-TOUCH+AirKiss, and ESP-TOUCH v2. Below are examples of ESP-TOUCH v2 provisioning.

ESP-TOUCH v2 Provisioning Example

1. Install the EspTouch app on your mobile device.
 - Android: [Download EspTouch for Android \(Source Code for Android\)](#)
 - iOS: [Download EspTouch for iOS \(Source Code for iOS\)](#)

2. Set the ESP device to Station mode.

Command:

```
AT+CWMODE=1
```

Response:

```
OK
```

3. Enable SmartConfig on the ESP32 device.

Command:

```
AT+CWSTARTSMART=4,0,"1234567890123456"
```

Response:

```
OK
```

4. Perform Wi-Fi provisioning on your mobile device.

Please follow the steps below: enable Wi-Fi on your mobile device and connect to the target network (e.g., SSID: test, password: 1234567890). Then, launch the EspTouch application and click EspTouch V2 button. On the redirected page, enter the password for the connected Wi-Fi, the number of devices for provisioning, and the AES key, as shown below:

```
Wi-Fi Password: 1234567890
Device Count for Provisioning: 1
AES Key: 1234567890123456
```

5. Confirm the successful provisioning on the ESP device.

At this point, the ESP device should output the below information:

```
smartconfig type:ESPTOUCH_V2
Smart get wifi info
ssid:test
password:1234567890
WIFI CONNECTED
WIFI GOT IP
smartconfig connected wifi
```

At this point, the ESP device's provisioning has been successfully completed.

6. Disable SmartConfig on the ESP32 device.

Command:

```
AT+CWSTOPSMART
```

Response:

```
OK
```

4.13.3 SoftAP Provisioning

AT SoftAP provisioning refers to WEB provisioning. For more details, please refer to [Web Server AT Examples](#).

4.13.4 WPS Provisioning

1. Prepare a router that supports WPS provisioning (e.g., Wi-Fi SSID: test).
2. Set the ESP device to Station mode.

Command:

```
AT+CWMODE=1
```

Response:

```
OK
```

3. Enable WPS provisioning on the ESP32 device.

Command:

```
AT+WPS=1
```

Response:

```
OK
```

Enable the WPS provisioning according to the router's user manual, and wait for a moment. The ESP device should output the below information, indicating that the provisioning has been successfully completed.

```
WIFI CONNECTED  
WIFI GOT IP
```

4. Query the Wi-Fi information of the connected ESP32 device.

Command:

```
AT+CWJAP?
```

Response:

```
+CWJAP:"test",.....  
OK
```

5. Disable WPS provisioning on the ESP32 device.

Command:

```
AT+WPS=0
```

Response:

```
OK
```


Chapter 5

ESP-AT Versions

This document mainly introduces ESP-AT versions, how to select a version, versioning scheme, support periods, viewing versions, and subscribing to AT releases.

5.1 Releases

The full history of ESP-AT releases can be found on the GitHub repository [Release page](#). There you can find the AT firmwares, release notes, links to each version of the documentation, and instructions for obtaining each version.

5.2 Which Version Should I Start With?

Please read [AT software solution selection](#).

5.3 Versioning Scheme

5.3.1 ESP-AT Firmware Release Management

ESP-AT release is specific to a chip and usually refers to the release of one or more AT firmwares for that chip. The AT command set supported by the released AT firmware is usually backward compatible. This means that you can update the new version of the AT firmware to devices with older versions.

The firmware released by ESP-AT follows a similar version management method as [Semantic Versioning](#). You can understand the differences of each version from the literal meaning. For example, `v3.3.0.0`, `v` represents version, followed by the version number in the following format:

```
<major>.<minor>.<patch>.<custom>
```

Where:

- `<major>` is the major version. For example, `v4.0.0.0` represents a major update, usually including the introduction of new chip support, new features, and bug fixes.
- `<minor>` is the minor version. For example, `v3.3.0.0` represents an important update, usually including new features, ESP-IDF version upgrade, and bug fixes.
- `<patch>` is the patch version, also known as the bugfix version. For example, `v2.4.2.0` represents only bug fixes without adding any new features.
- `<custom>` is the custom version. It is usually used for downstream distributors or customized project versions.

5.3.2 ESP-AT Branch Release Management

When ESP-AT releases AT firmware, if there are major or minor version updates, a new [release branch](#) will be created.

The release branches of ESP-AT follow a similar version management method as [Semantic Versioning](#). You can understand the differences of each version from the literal meaning. For example, `release/v3.3.0.0`, `release` represents the release branch, `v` represents the version of the release branch, followed by the version number in the following format:

```
<major>.<minor>.0.0
```

Where:

- `<major>` is the major version. For example, `release/v4.0.0.0` represents a major update, usually including the introduction of new chip support, new features, and bug fixes.
- `<minor>` is the minor version. For example, `release/v3.3.0.0` represents an important update, usually including new features, ESP-IDF version upgrade, and bug fixes.

In general, the AT firmware to be released will undergo multiple rounds of testing on the release branch until there are no major issues. Then, the AT firmware will be released, and the release branch will be synchronized to GitHub.

- New feature development usually takes place on the master branch and is not merged into the release branch.
- Bug fixes usually take place on the master branch and, if the issue is severe, will be merged into the release branch.

5.4 Support Periods

Each ESP-IDF major and minor release version has an associated support period. After this period, the release is End of Life and no longer supported. Since ESP-AT is a project based on ESP-IDF, the support period of ESP-AT is limited by the support period of ESP-IDF. The current information of each version of ESP-AT is as follows:

AT Version (Release Date)	Supported Chips	IDF Version	IDF EOL Date	AT New Version Plan
v5.0.1.0 (2026.3.9)	• ESP32-C5 • ESP32-C61	~v5.5.3 (82e525ef)	2028.1.21	2027.8.21 - 2027.10.21
v5.0.0.0 (2025.11.24)	• ESP32-C5 • ESP32-C61	~v5.5.2 (cc569cbd)	2028.1.21	New version v5.0.1.0 released
v4.1.1.0 (2025.7.31)	• ESP32 • ESP32-C2 • ESP32-C3 • ESP32-C6 • ESP32-S2	~v5.4.2 (8ad0d3d8)	2027.7.5	2027.2.5 - 2027.4.5
v4.1.0.0 (2025.7.18)	• ESP32 • ESP32-C2 • ESP32-C3 • ESP32-C6	~v5.4.2 (8ad0d3d8)	2027.7.5	New version v4.1.1.0 released
v4.0.0.0 (2023.12.29)	ESP32-C6	v5.1.2	2025.12.30	New version v4.1.1.0 released
v3.4.0.0 (2024.6.7)	• ESP32 • ESP32-S2	v5.0.6	2025.5.29	New version v4.1.1.0 released
v3.3.0.0 (2024.5.9)	• ESP32-C2 • ESP32-C3	v5.0.6	2025.5.29	New version v4.1.1.0 released

Support periods for each released version of ESP-AT are as follows:

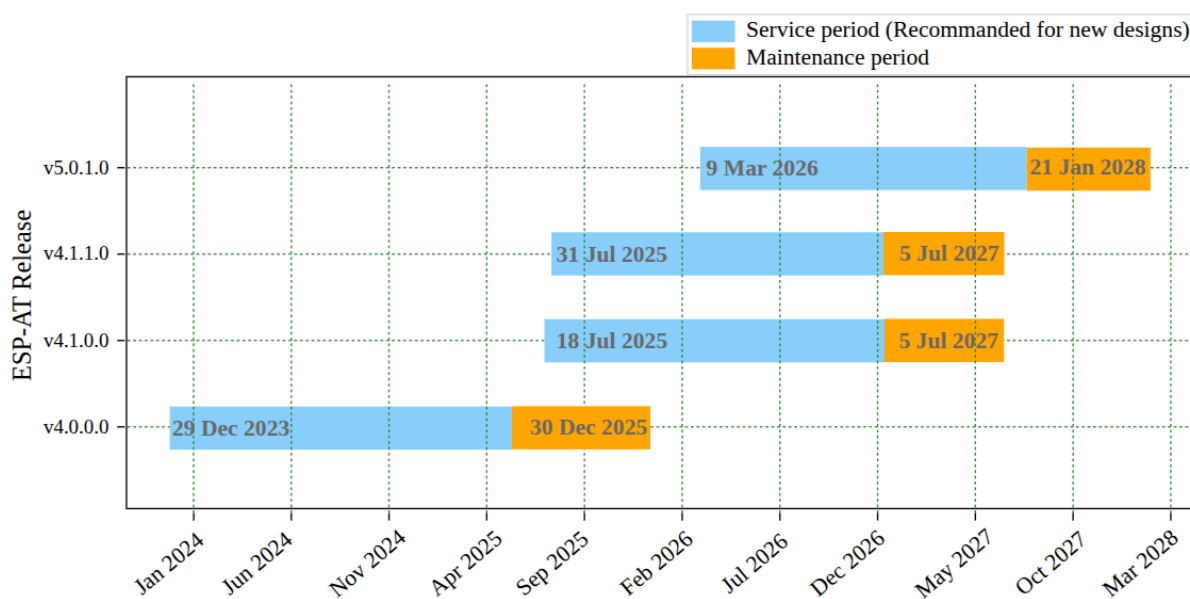


Fig. 1: ESP-AT Version Support Periods

- Service Period of Support Period

Usually, it starts from the release of the AT firmware for the chip and lasts until the planned release of the next AT version for that chip. The release time of the next AT version is usually a few months before the end of the corresponding [ESP-IDF support period](#) (the AT release notes provide information about the ESP-IDF version corresponding to the chip).

- **Maintenance Period of Support Period**

Usually, it starts after the service period ends and lasts until the end of the corresponding [ESP-IDF support period](#) for the chip (the AT release notes provide information about the ESP-IDF version corresponding to the chip). For example, if the support period of ESP-IDF v5.0 is until May 29, 2025, then the maintenance period for ESP-AT v3.0 ~ v3.3 will also be until May 29, 2025.

In general:

- Once a new version of AT is released, the service period of the support period for the old version ends and enters the maintenance period of the support period.
For example, when AT releases version v3.3.0.0 (for ESP32-C2 and ESP32-C3 chips), the service period of the support period for version v3.2.0.0 of ESP32-C3 ends and enters the maintenance period of the support period; the service period of the support period for version v3.1.0.0 of ESP32-C2 ends and enters the maintenance period of the support period.
- If you have a GitHub account, please [Subscribe to AT releases](#). GitHub will notify you when a new version is released. When a bugfix version of the AT firmware you are using is released, please plan to upgrade to that bugfix version.
- Please make sure to plan the upgrade to a new version before the version you are using stops receiving updates and support.
- Being within the support period means that the ESP-AT team will continue to perform important bug fixes, security fixes, etc. on the release branch on GitHub, and periodically release new bugfix versions as needed.

5.5 Check the Current AT Firmware Version

Please send the [AT+GMR](#) command to check the AT firmware version information. Refer to the parameter description under the [AT+GMR](#) command for more information.

5.6 Subscribe to AT Releases

- *Step 1: Log in to Your GitHub Account*
- *Step 2: Choose Customized Notifications*
- *Step 3: Apply for Release Notifications*

5.6.1 Step 1: Log in to Your GitHub Account

Before you start, please [sign in your GitHub account](#), as you need login permission to subscribe to AT releases.

5.6.2 Step 2: Choose Customized Notifications

Visit the [ESP-AT repository](#), click on `Watch` in the upper right corner of the page, and then click on `Custom`.

5.6.3 Step 3: Apply for Release Notifications

Check `Releases` and click on `Apply`.

This completes the process of subscribing to AT releases. You will receive notifications from GitHub when a new AT version is released.

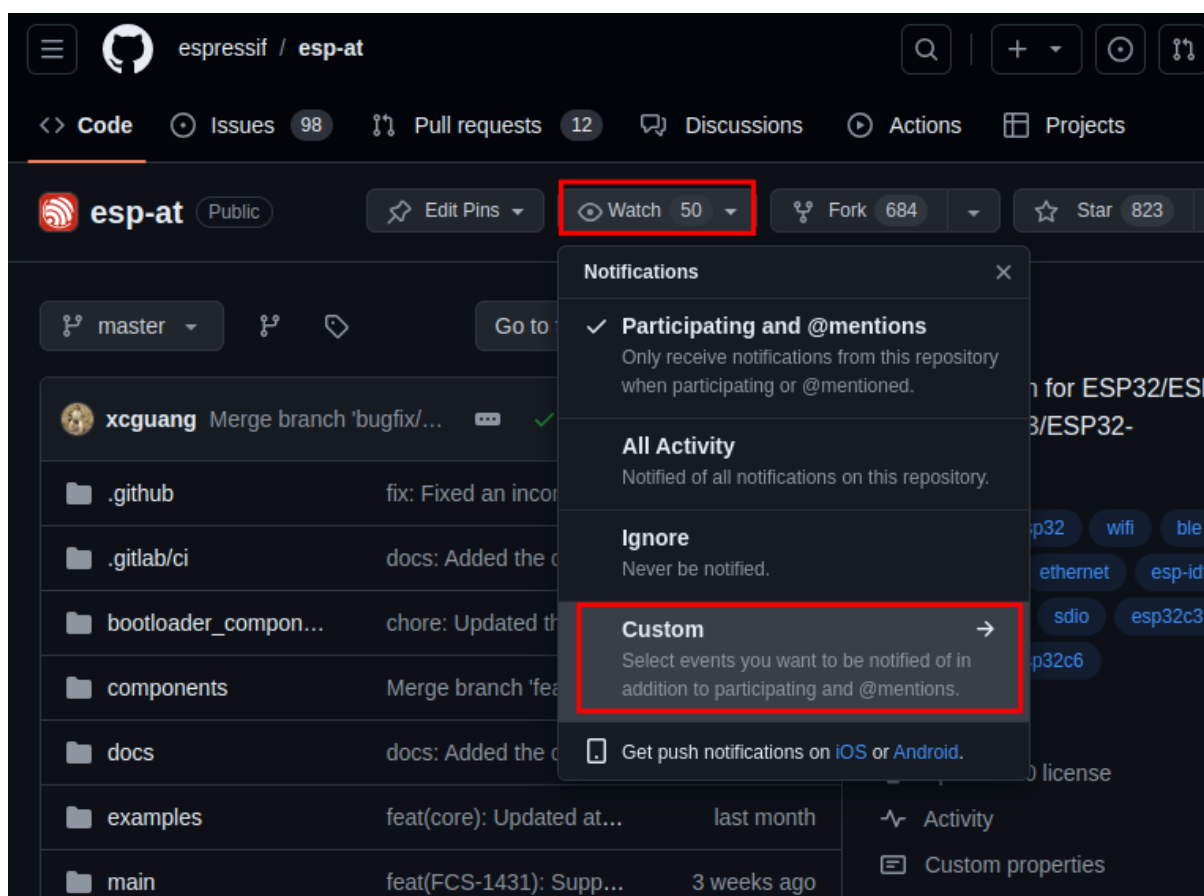


Fig. 2: Customized Notifications (click to enlarge)

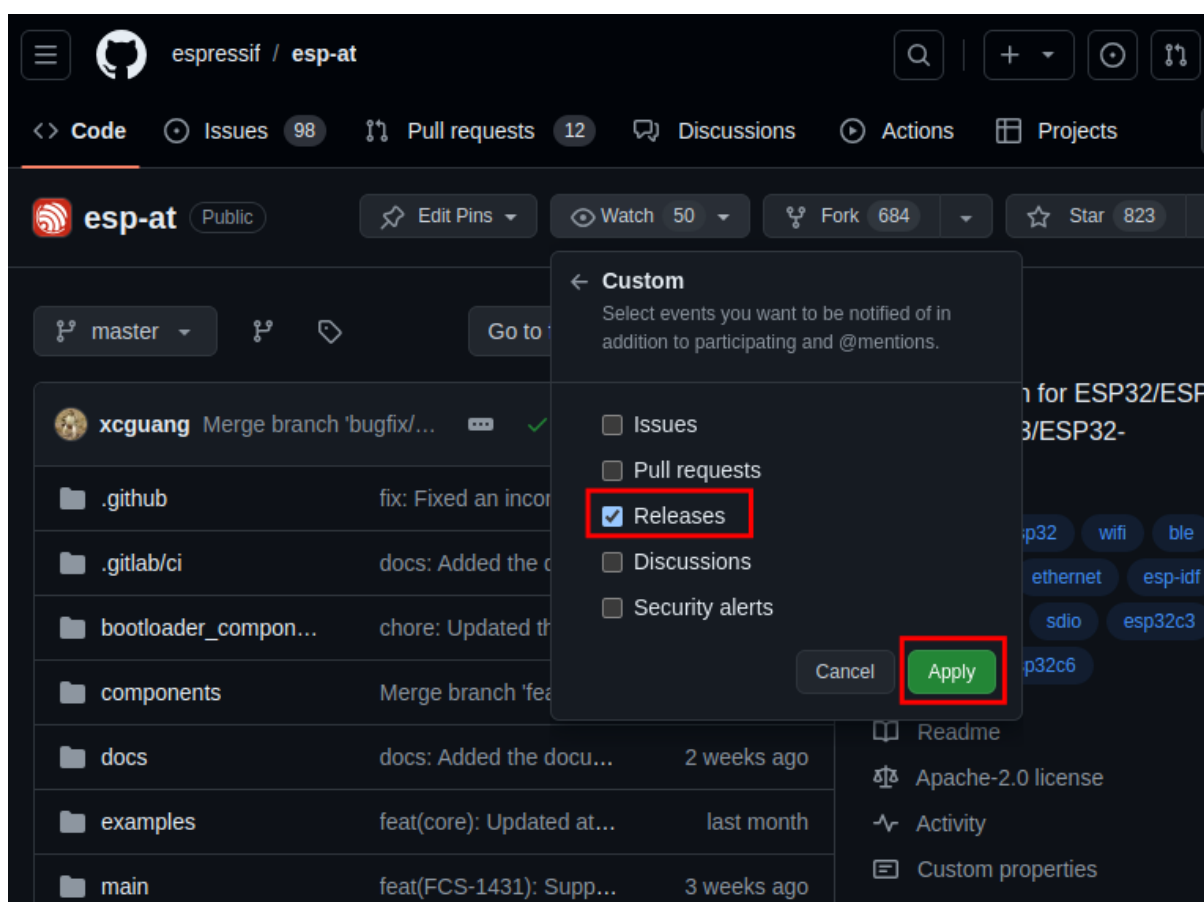


Fig. 3: Apply for Release Notifications (click to enlarge)

Chapter 6

How to Compile and Develop Your Own AT Project

6.1 Compile ESP-AT Project Locally

This document details how to build your own ESP-AT project locally and flash the generated firmware into your ESP32. It comes in handy when the [official released firmware](#) cannot meet your needs, for example, to customize the *AT port pins*, *Bluetooth LE services*, and *partitions*, and so on.

If you have difficulties in compiling ESP-AT project locally, or you only need to modify a small amount of code, we recommend that you use [Compile ESP-AT Project on the GitHub Webpage](#).

6.1.1 Detailed Steps

Please follow the detailed steps below to set up your environment and build the project. **It is recommended that you develop the esp-at project on Linux system.**

- [Step 1. Get Started with ESP-IDF](#)
- [Step 2. Get ESP-AT](#)
- [Step 3. Install Environment](#)
- [Step 4. Connect Your Device](#)
- [Step 5. Configure](#)
- [Step 6. Build the Project](#)
- [Step 7. Flash onto the Device](#)
- [build.py Advanced Usage](#)

6.1.2 Step 1. Get Started with ESP-IDF

Get started with ESP-IDF before compiling an ESP-AT project, because ESP-AT is developed based on ESP-IDF.

Please follow [ESP-IDF v5.4 Get Started](#) guide, configure, build, flash onto the ESP32 device of the `hello_world` example.

Note: This step is not a must, but if you are a beginner, you are strongly recommended to complete it in order to familiarize yourself with ESP-IDF and ensure smooth proceeding with the following steps.

After you complete the ESP-IDF get started in the step above, now you can proceed to compile an ESP-AT project according to the following steps.

6.1.3 Step 2. Get ESP-AT

To compile an ESP-AT project, you need the software libraries provided by Espressif in the ESP-AT repository.

To get ESP-AT, navigate to your installation directory and clone the repository with `git clone`, following instructions below specific to your operating system.

- Linux or macOS

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

- Windows

For ESP32 series of modules, it is recommended that you run [ESP-IDF 5.4 CMD](#) as an administrator first.

```
cd %userprofile%\esp
git clone --recursive https://github.com/espressif/esp-at.git
```

If you are in China or regions with restricted GitHub access, ESP-AT may be cloned faster from the following mirror: `git clone https://jihulab.com/esp-mirror/espressif/esp-at.git`.

ESP-AT will be downloaded into `~/esp/esp-at` on Linux or macOS, or `%userprofile%\esp\esp-at` on Windows.

Note: This guide uses the directory `~/esp` on Linux or macOS, or `%userprofile%\esp` on Windows as an installation folder for ESP-AT. You can use any directory, but you will need to adjust paths for the commands respectively. Keep in mind that ESP-AT does not support spaces in paths.

6.1.4 Step 3. Install Environment

Run the project tool `install` to install the environment. This tool will automatically install Python packages, ESP-IDF repository, and the compiler and tools used by ESP-IDF.

- Linux or macOS

```
./build.py install
```

- Windows

```
python build.py install
```

Select the following configuration options for your ESP32 if it is your first time.

- Select the `Platform` name for your ESP32. For example, select `PLATFORM_ESP32` for ESP32 series of products. `Platform` name is defined in [factory_param_data.csv](#).
- Select the `Module` name for your ESP32. For example, select `WROOM-32` for the ESP32-WROOM-32 module. `Module` name is defined in [factory_param_data.csv](#).
- Before selecting to enable or disable silence mode, please read the [documentation](#) to understand silence mode. Generally, it should be disabled.
- The above three option items will not appear if the file `build/module_info.json` exists. So please delete it if you want to reconfigure the module information.

For example, set `Platform` name to `ESP32`, `Module` name to `WROOM-32`, and enable silence mode as follows:

```
$ ./build.py install
Ready to install ESP-IDF prerequisites..

... (more lines of install ESP-IDF prerequisites)

Ready to install ESP-AT prerequisites..

... (more lines of install ESP-IDF prerequisites)

Platform name:
1. PLATFORM_ESP32
...
choose(range[1,7]):1

Module name:
1. WROOM-32 (Firmware description: ...)
...
choose(range[1,7]):1

Enable silence mode to remove some logs and reduce the firmware size?
0. No
1. Yes
choose(range[0,1]):1
Platform name:ESP32 Module name:WROOM-32 Silence:1
Cloning into 'esp-idf'...

... (more lines of clone esp-idf)

Ready to set up ESP-IDF tools..

... (more lines of set up ESP-IDF tools)

All done! You can now run:

./build.py build
```

6.1.5 Step 4. Connect Your Device

Connect your ESP32 device to the PC with a USB cable to download firmware and print log. See [Hardware Connection](#) for more information. Note that you do not need to set up the “AT command/response” connection if you do not send AT commands and receive AT responses during the compiling process. You can change default port pins referring to [How to Set AT Port Pins](#).

6.1.6 Step 5. Configure

Run the project configuration utility `menuconfig` to configure.

- Linux or macOS

```
./build.py menuconfig
```

- Windows

```
python build.py menuconfig
```

If the previous steps have been done correctly, the following menu pops up:

You are using this menu to set up project-specific configuration, e.g. changing AT port pins, enabling Classic Bluetooth function, etc. If you made no changes, it will run with the default configuration.

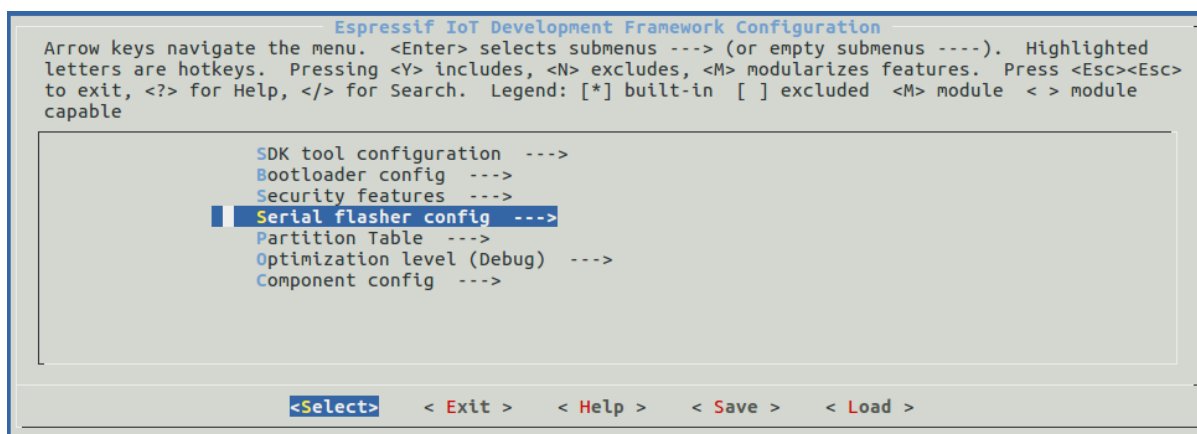


Fig. 1: Project configuration - Home window

6.1.7 Step 6. Build the Project

Build the project by running:

- Linux or macOS

```
./build.py build
```

- Windows

```
python build.py build
```

If Bluetooth feature is enabled, the firmware size will be much larger. Please make sure it does not exceed the OTA partition size.

After compiled, the combined factory bin will be created in `build/factory`. See [ESP-AT Firmware Differences](#) for more information.

6.1.8 Step 7. Flash onto the Device

Flash the firmware that you just compiled onto your ESP32 by running:

- Linux or macOS

```
./build.py -p (PORT) flash
```

- Windows

```
python build.py -p (PORT) flash
```

Note that you need to replace `(PORT)` with your ESP32's serial port name. Or you can follow the printed instructions to download the bin files into flash. Note that you also need to replace the `(PORT)`.

If the ESP-AT bin fails to boot and prints "ota data partition invalid", you should run `python build.py erase_flash` to erase the entire flash, and then reflash the AT firmware.

6.1.9 build.py Advanced Usage

The script `build.py` is based on `idf.py`, which means that all `idf.py <cmd>` features should be included in `build.py <cmd>`. You can run the following command for more details.

- Linux or macOS


```
./build.py --help
```

- Windows

```
python build.py --help
```

6.2 Compile ESP-AT Project on the GitHub Webpage

This document provides a detailed guide on how to compile the ESP-AT project through a web page. When the default *officially released firmware* does not satisfy your requirements, for example, you want to enable *WebSocket functionality* and disable *mDNS functionality*, you will need to compile the ESP-AT project. Usually we recommend you to *Compile ESP-AT Project Locally*. However, if you have difficulties building the project locally, please refer to this document to compile the ESP-AT project through the webpage.

Attention: The AT firmware compiled from webpages needs to be tested and verified for functionality based on your own product.

Please save the firmware and download link, for possible issue debugging in the future.

6.2.1 Detailed Steps

Please follow the detailed steps below to complete the Fork, environment configuration, code modification, and compilation of the ESP-AT project.

- *Step 1. Log in to your GitHub account*
- *Step 2. Fork the ESP-AT project*
- *Step 3. Enable GitHub Actions*
- *Step 4. Configure the secrets required to compile the ESP-AT project*
- *Step 5. Use the github.dev editor to modify and submit the code*
- *Step 6. Compile the AT firmware using GitHub Actions*

6.2.2 Step 1. Log in to your GitHub account

Before starting, please [log in to your GitHub account](#) as compile and firmware download require login credentials.

6.2.3 Step 2. Fork the ESP-AT project

Visit the [ESP-AT repository](#) and click `Fork` in the upper right corner of the page.

Click `Create fork` to copy the repository to your GitHub account. The default Fork only copies the master branch. If you need to modify the code based on a specific branch, uncheck `Copy the master branch only`.

6.2.4 Step 3. Enable GitHub Actions

Under the repository, click `Actions` to enter the GitHub Actions page.

Click `I understand my workflows, go ahead and enable them` to enable GitHub Actions.

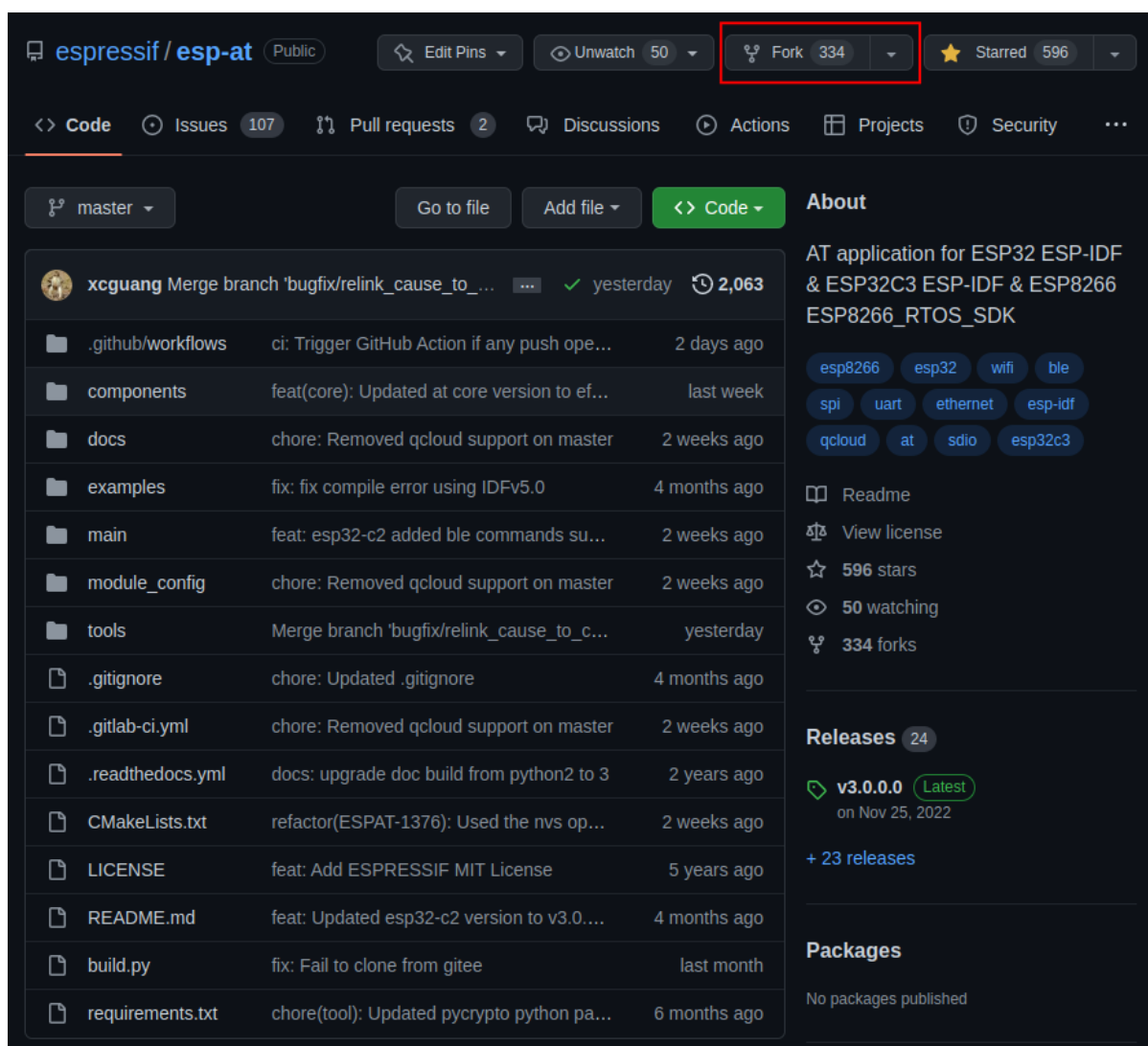


Fig. 2: Click Fork

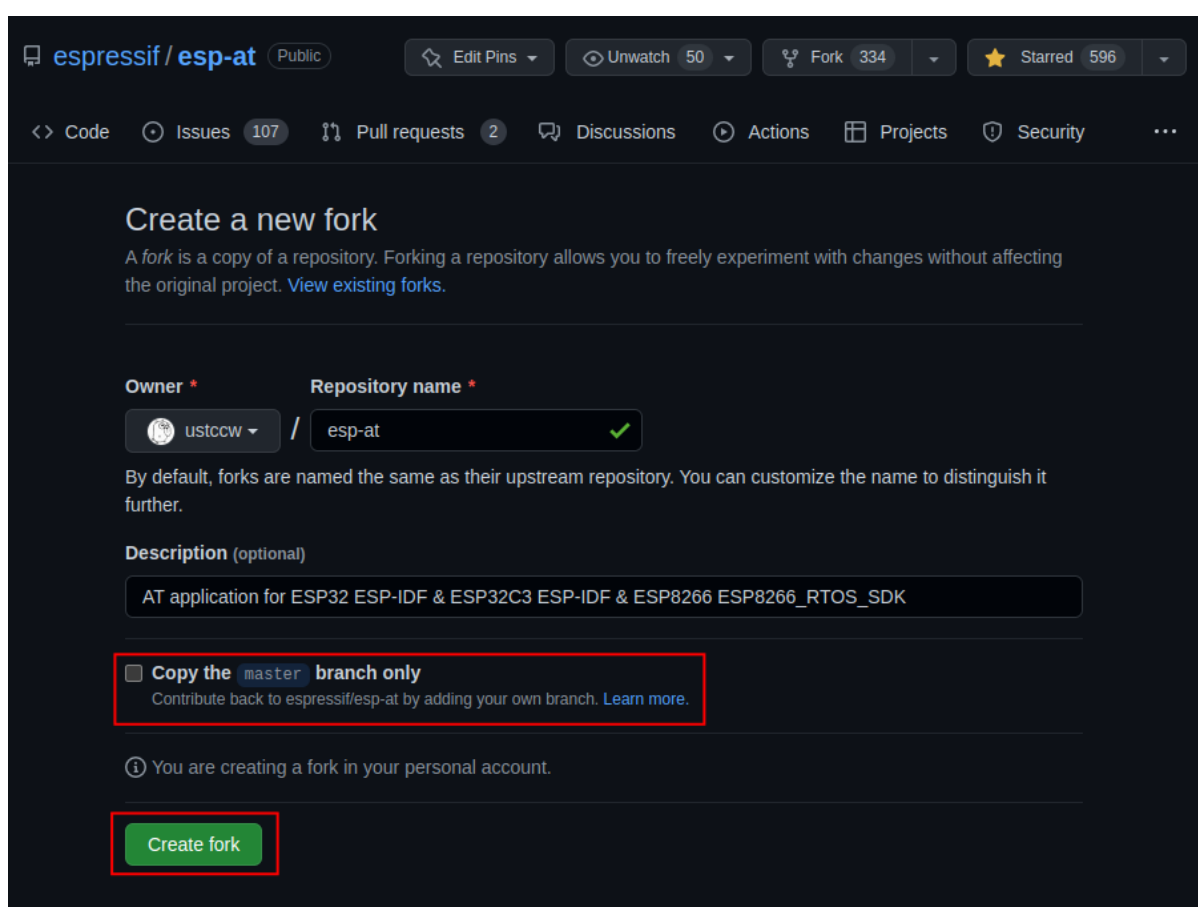


Fig. 3: Create Fork

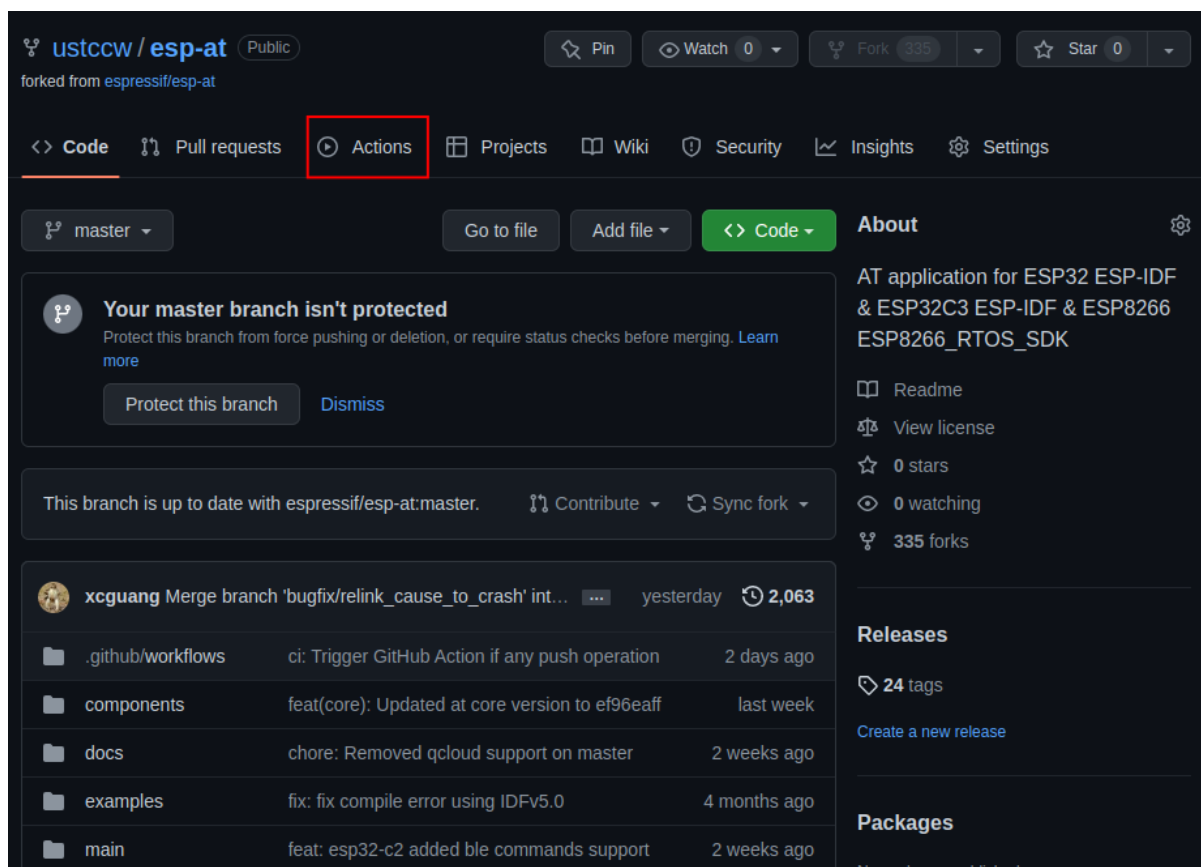


Fig. 4: Click Actions

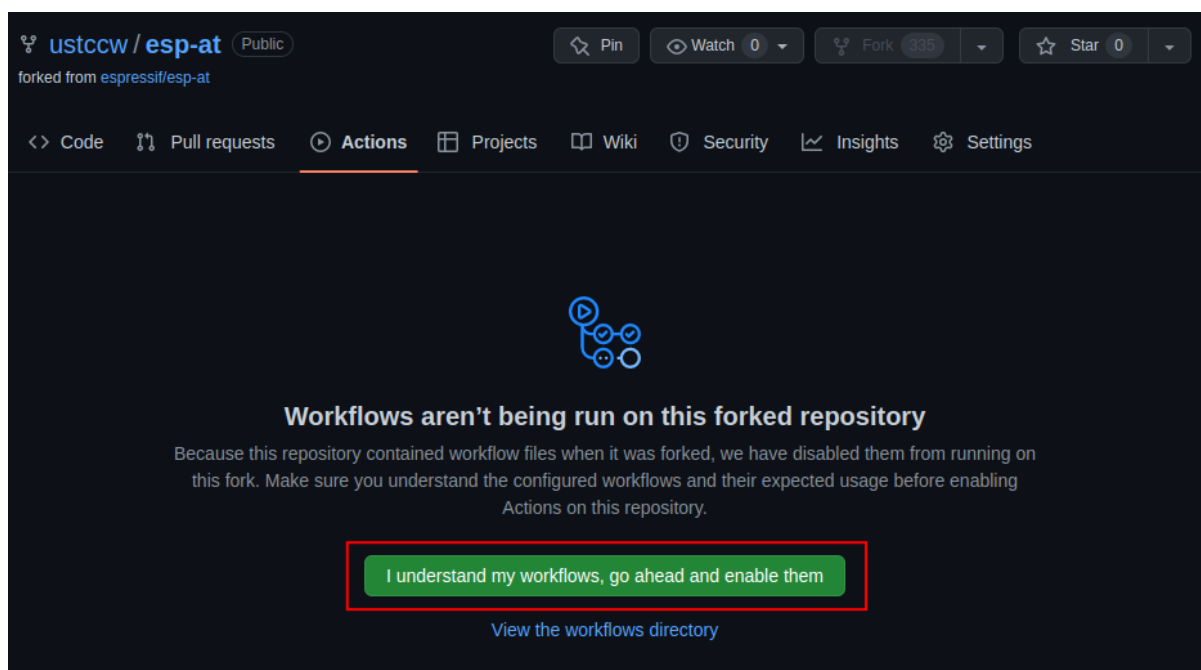


Fig. 5: Enable GitHub Actions

GitHub Actions is now enabled.

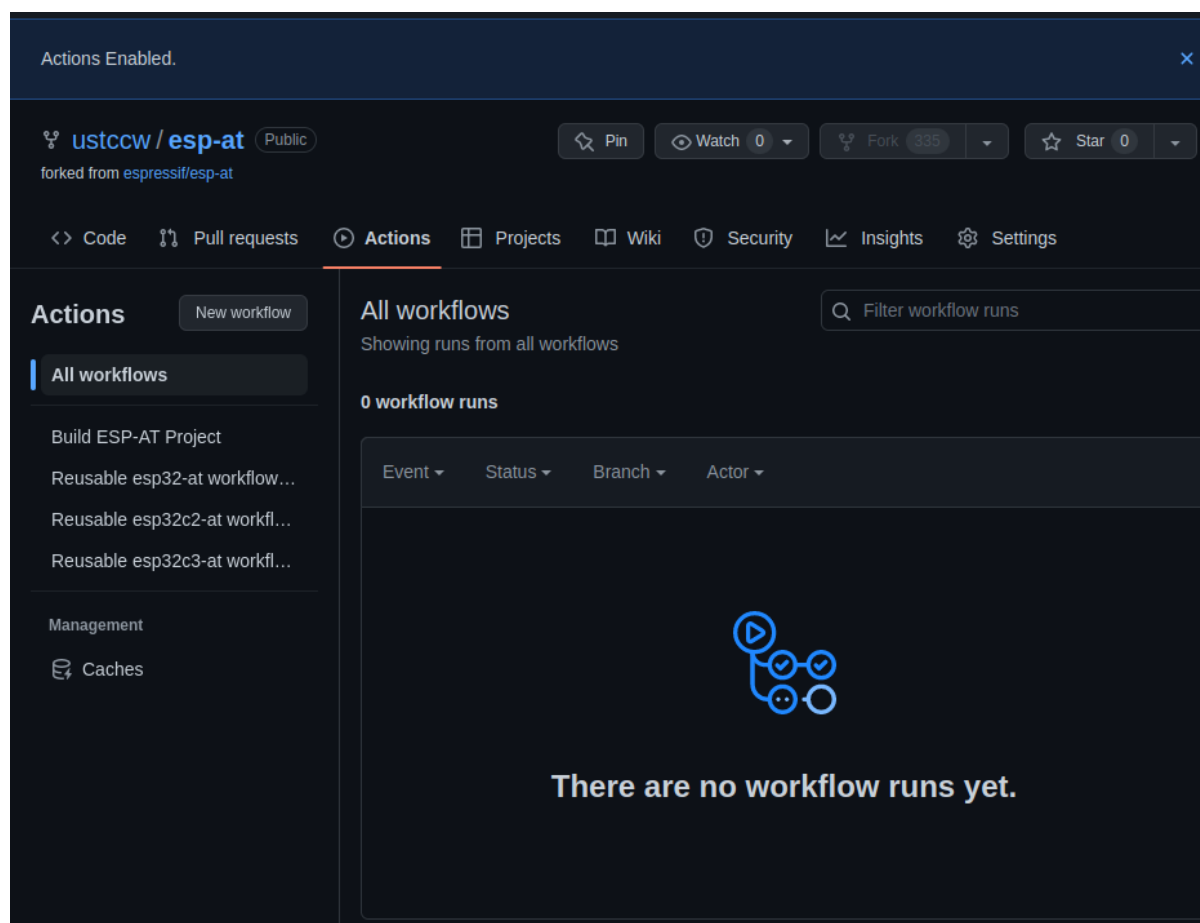


Fig. 6: GitHub Actions Enabled Successfully

6.2.5 Step 4. Configure the secrets required to compile the ESP-AT project

If you have an account and an OTA token of [the Espressif OTA server](#), and you need to upgrade the AT firmware by using the `AT+CIUPDATE` command, you need to complete this step. Otherwise, it is recommended to disable `CONFIG_AT_OTA_SUPPORT` (see [Step 5. Use the github.dev editor to modify and submit the code](#) for details) or skip this step.

Under the repository, click `Setting` to enter the settings page.

Click on `Settings > Secrets and variables > Actions` to access the Action configuration page.

Click `New repository secret` to enter the secret creation page.

Enter `Name and Secrets`, and click `Add secret` to create a new secret. This secret is the OTA token.

All the possible secret names that need to be configured are as follows:

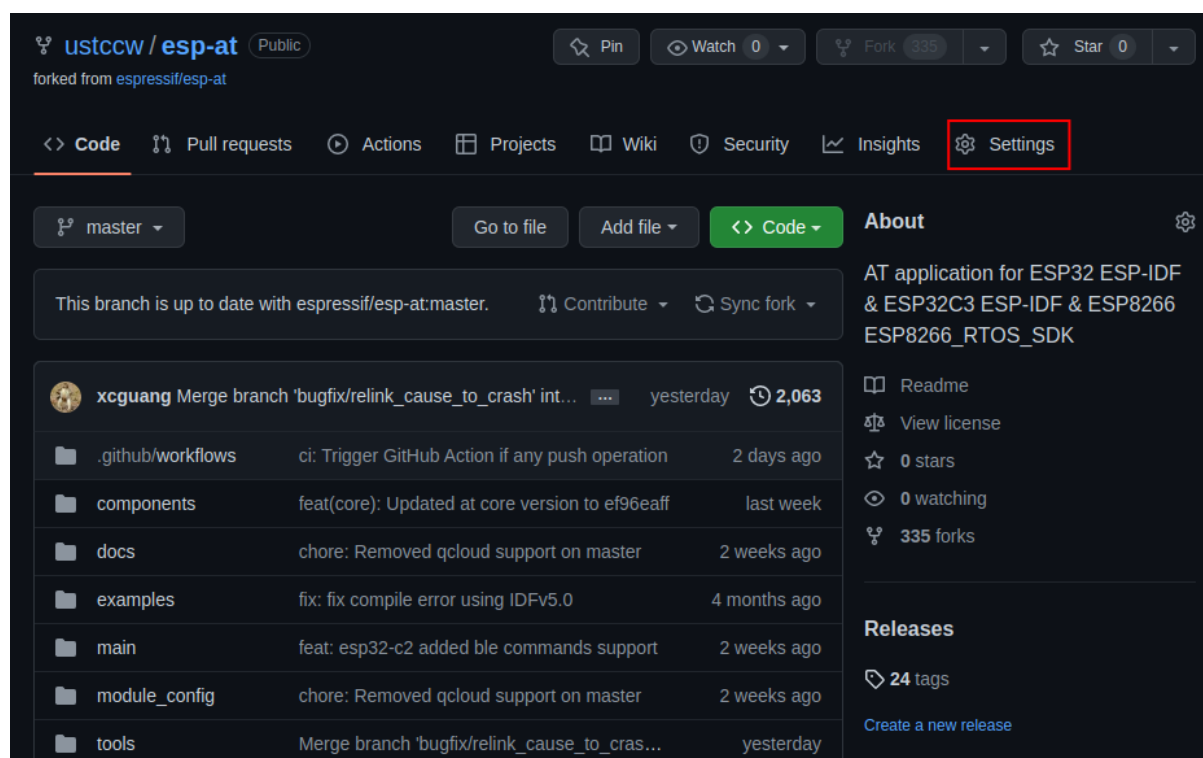


Fig. 7: Click on Settings

```

AT_OTA_TOKEN_WROOM32
AT_OTA_TOKEN_WROVER32
AT_OTA_TOKEN_ESP32_MINI_1
AT_OTA_TOKEN_ESP32_PICO_D4
AT_OTA_TOKEN_ESP32_SOLO_1
AT_OTA_TOKEN_ESP32C3_MINI
ESP32C2_2MB_TOKEN
ESP32C2_4MB_TOKEN
ESP32C5_4MB_TOKEN
ESP32C6_4MB_TOKEN
ESP32C61_4MB_TOKEN

```

If you need to support AT firmware upgrades for more modules, please repeat the above steps and add different keys again. After creating all the secrets, you will have the following page.

6.2.6 Step 5. Use the github.dev editor to modify and submit the code

For how to use the github.dev editor, please refer to the official [github.dev document](#). The example process is shown below.

5.1 Open the github.dev editor

Go back to the repository homepage and select the branch name that you want to modify.

Press the `.` key on the keyboard to open the branch code using the github.dev editor.

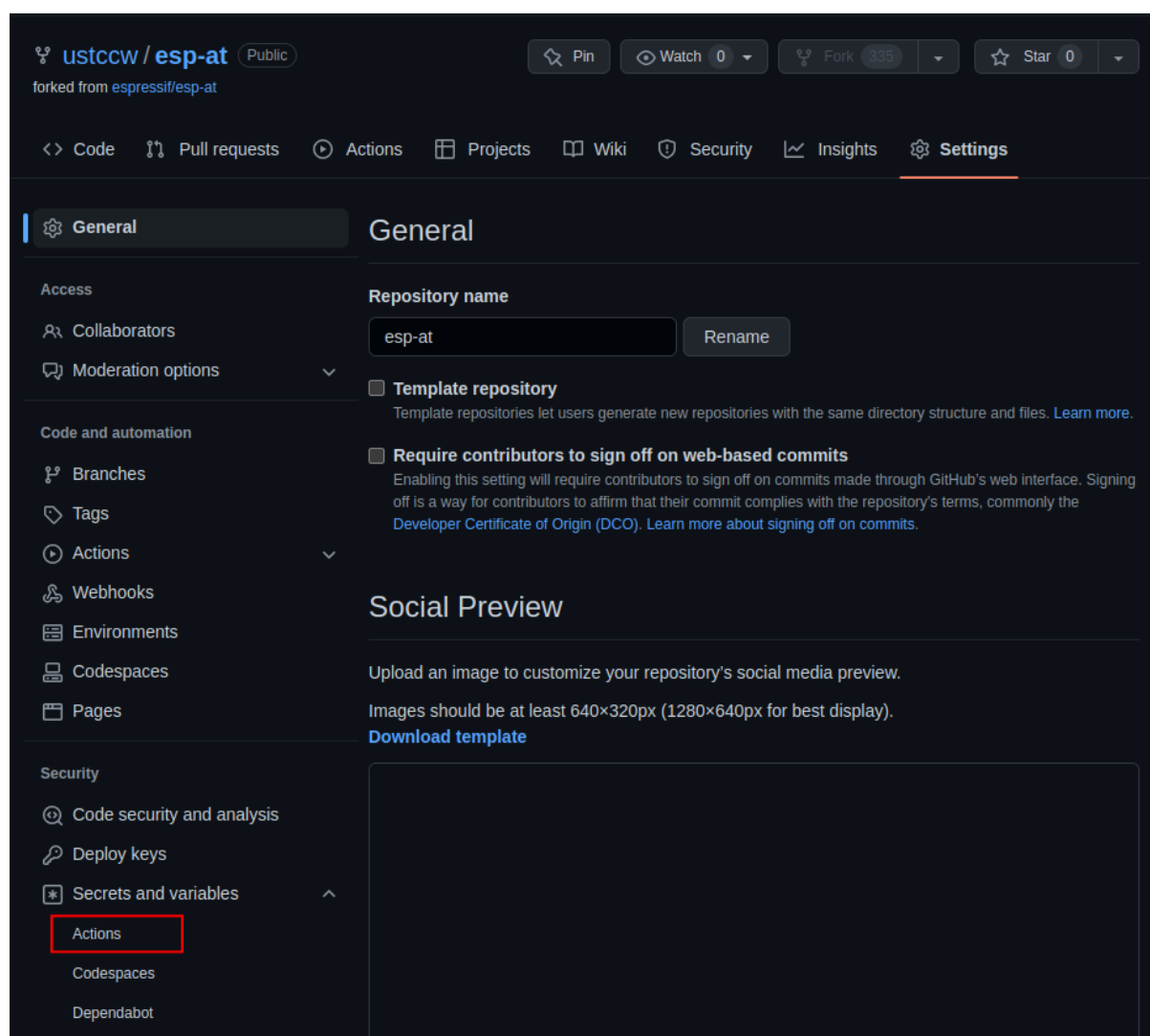


Fig. 8: Enter Actions Configuration Page

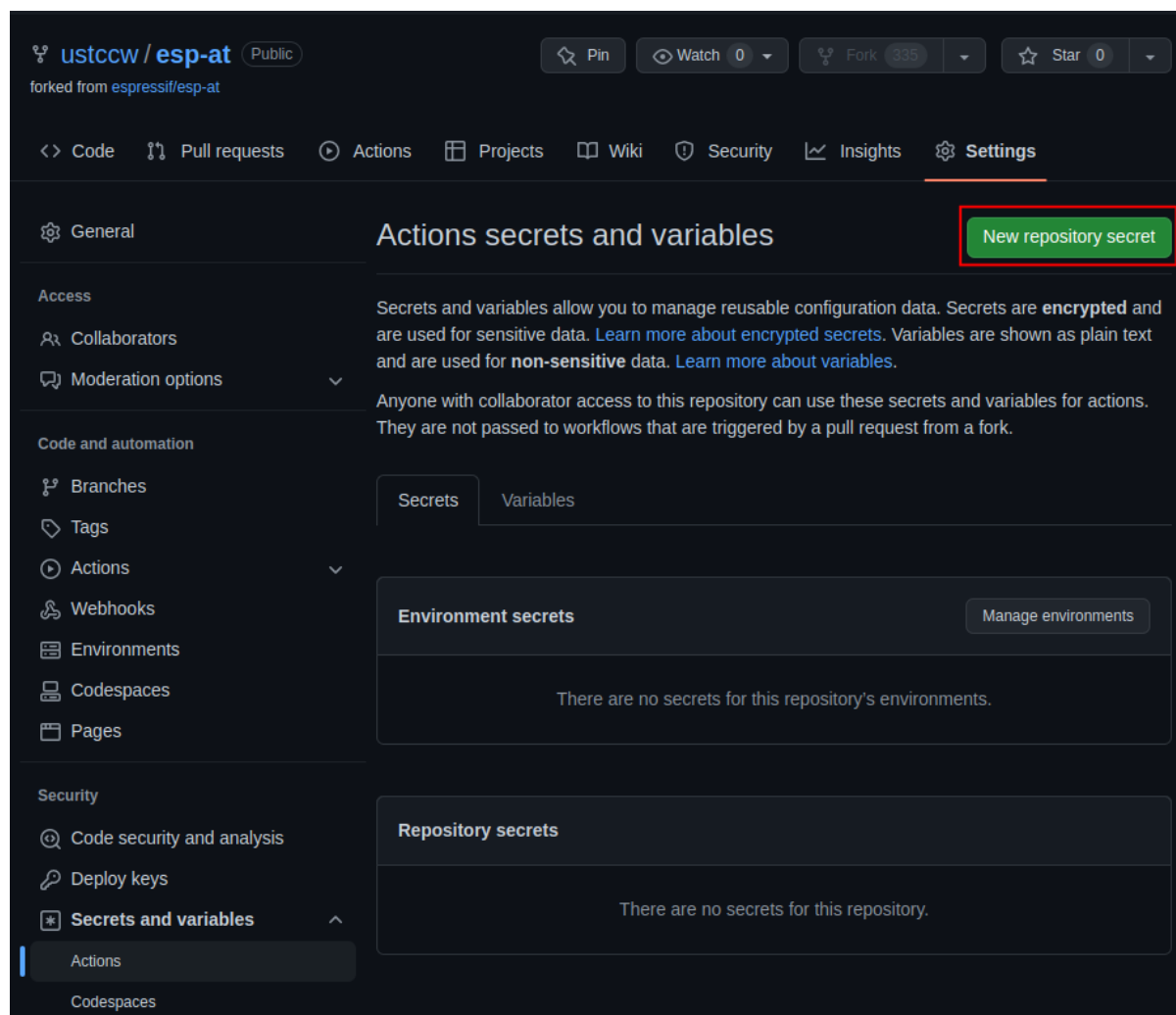


Fig. 9: Create Secret Page

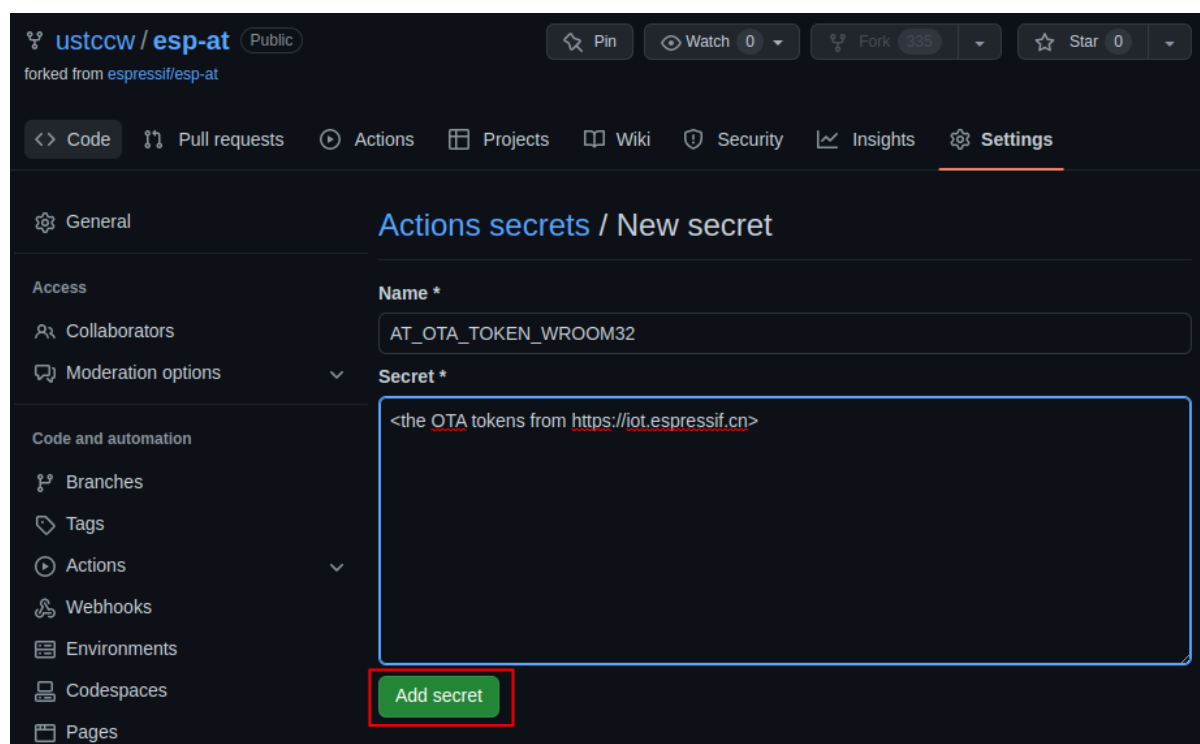


Fig. 10: Create Secret



Fig. 11: Create Secrets Completed

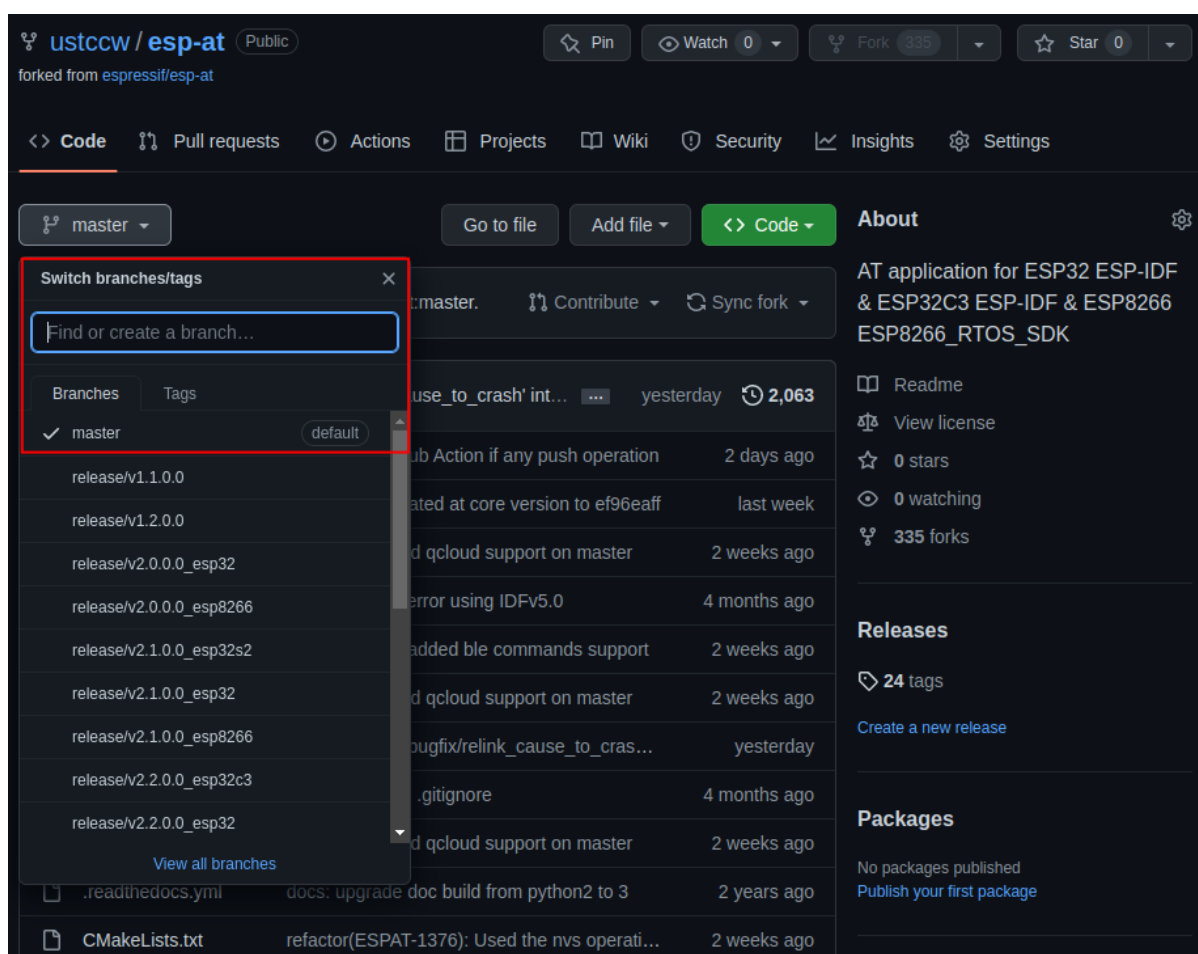


Fig. 12: Select Branch

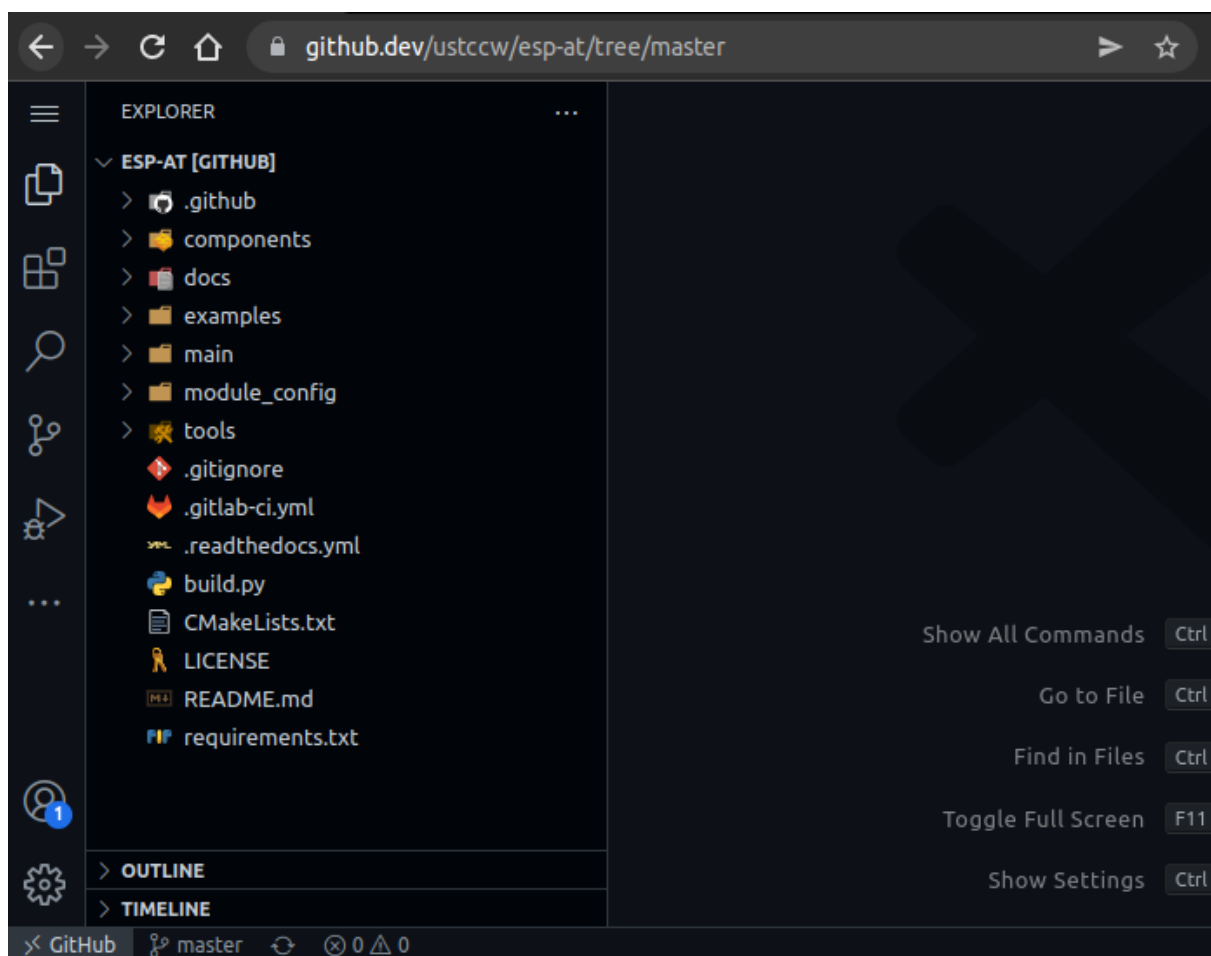


Fig. 13: Open Branch Code

5.2 Create a new branch

At the bottom of the editor, click the branch name in the status bar. In the dropdown, enter the name for a new branch and click **Create new branch**.

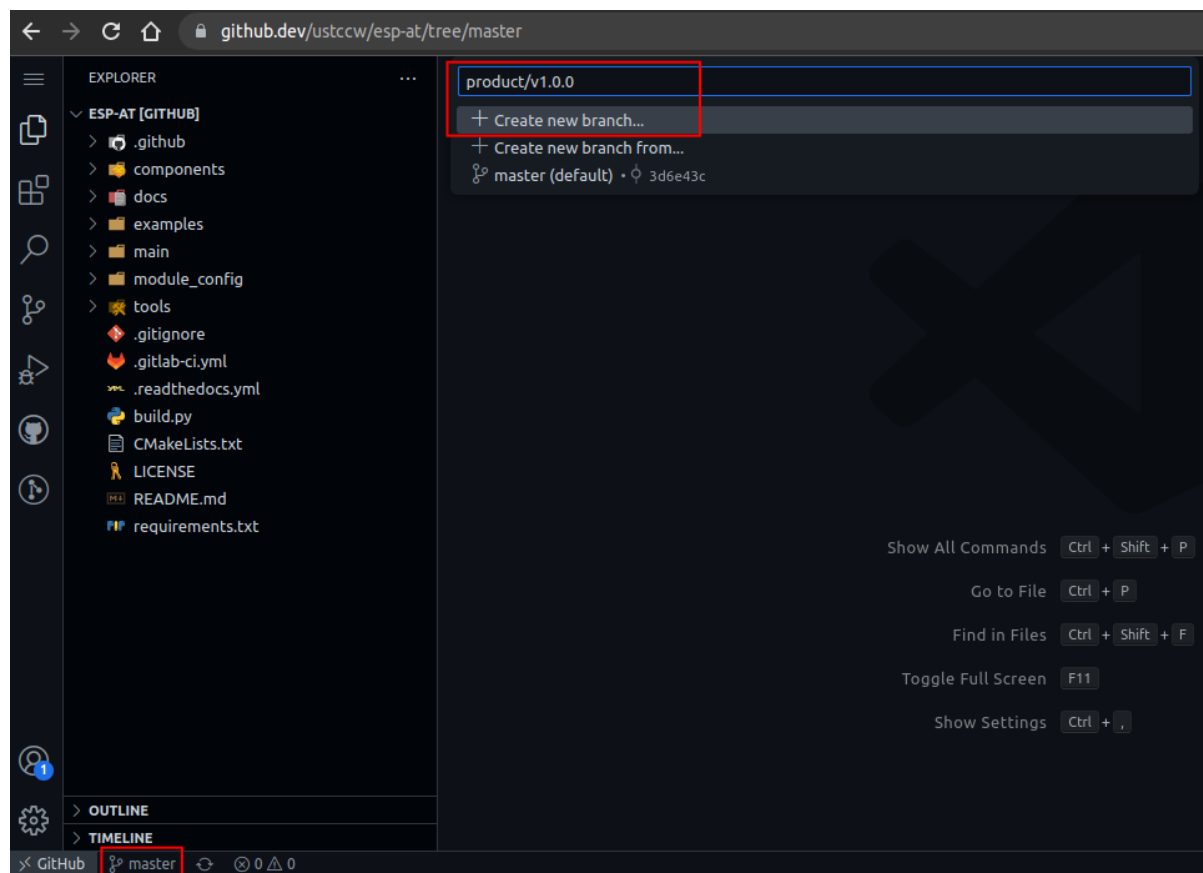


Fig. 14: Create New Branch (Click to Enlarge)

Click **Switch to Branch** to create the branch.

The branch has been created.

5.3 Commit changes

Modify the code in the github.dev editor. For example, to enable *WebSocket functionality* and disable *mDNS functionality*, open the configuration file `esp-at/module_config/<your_module_name>/sdkconfig.defaults` and add the following lines:

```
CONFIG_AT_WS_COMMAND_SUPPORT=y
CONFIG_AT_MDNS_COMMAND_SUPPORT=n
```

In the activity bar, click the **Source Control** view. Click the plus sign **+** next to the file that has been modified to stage the changes.

Enter a commit message to describe the changes you have made. Click **Commit & Push** to submit the changes.

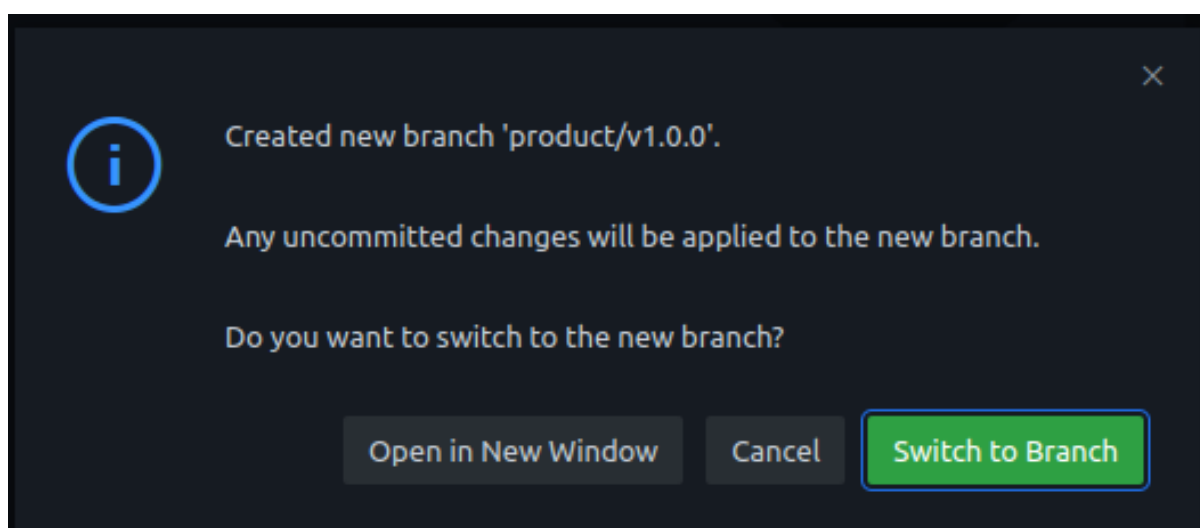


Fig. 15: Confirm Branch Creation

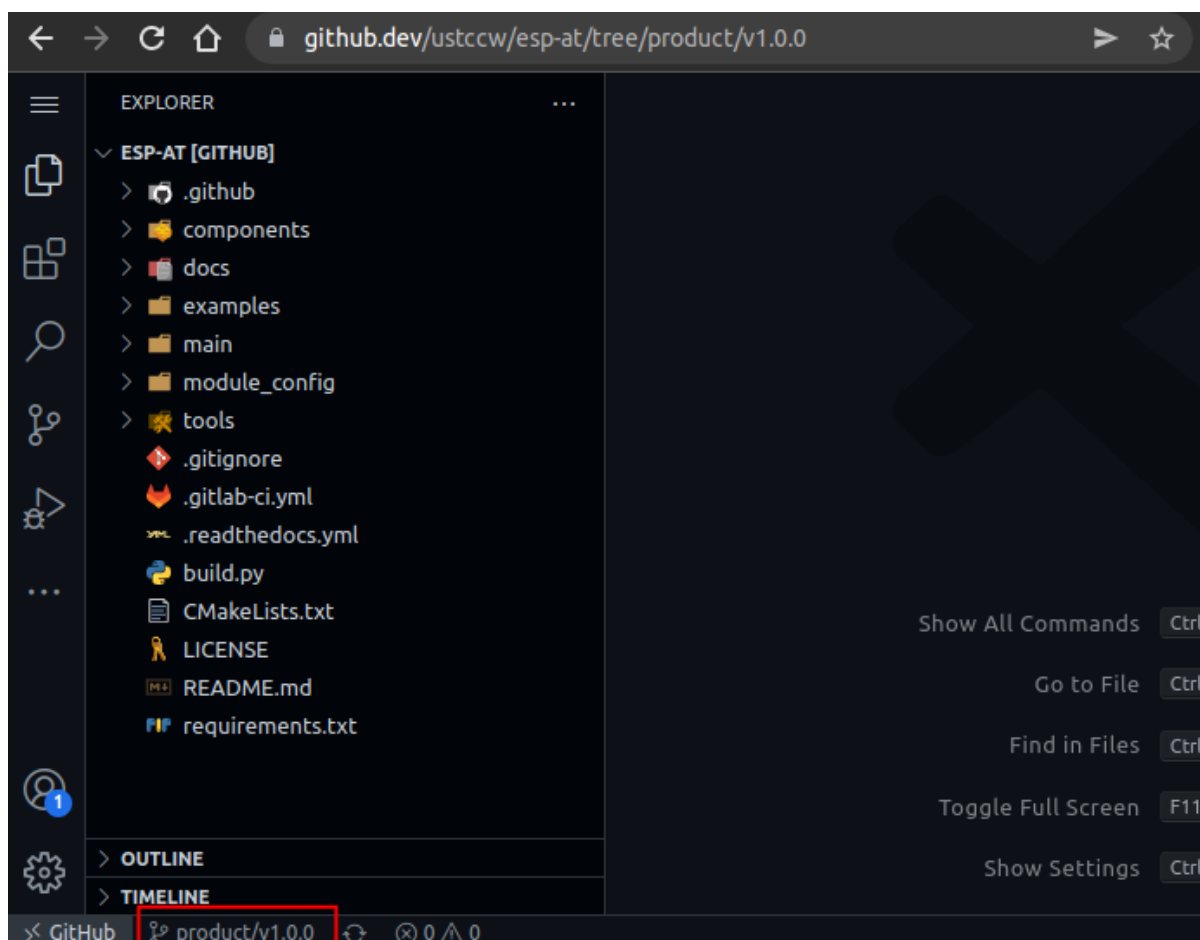


Fig. 16: Branch Created

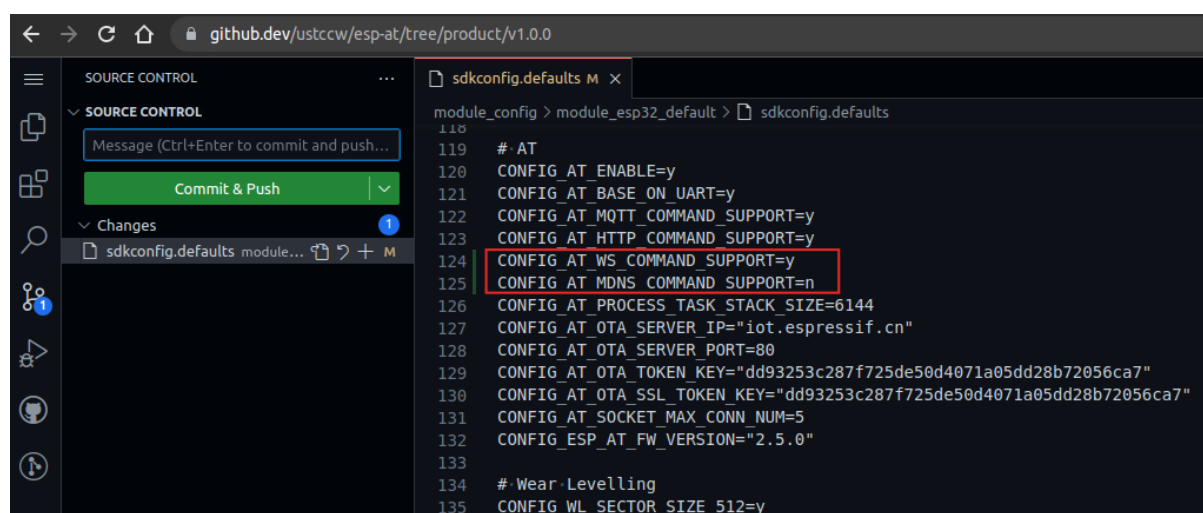


Fig. 17: Add WebSocket Command Support and Disable mDNS Functionality (Click to Enlarge)

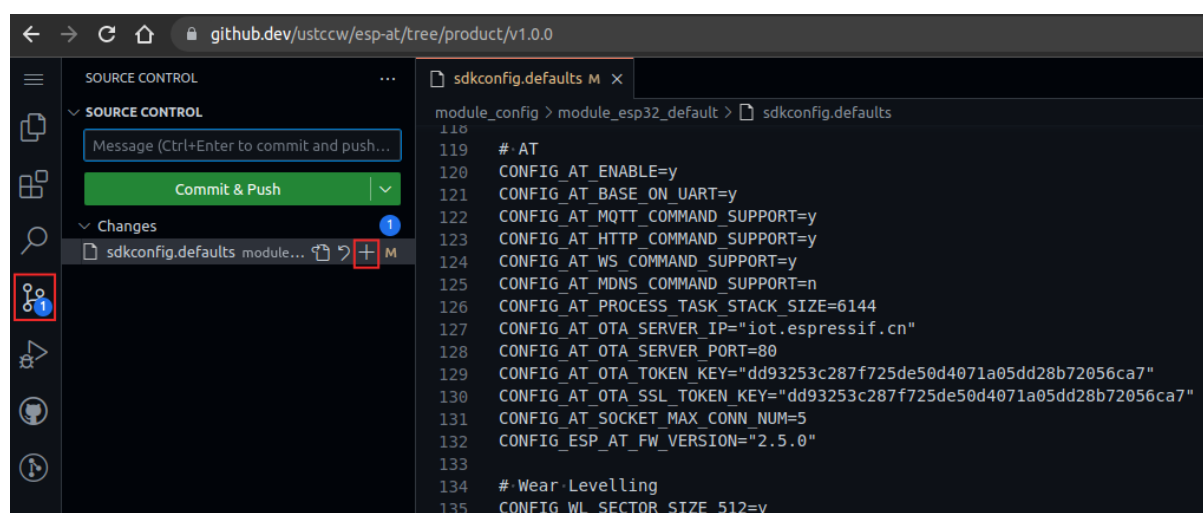


Fig. 18: Stage Changes (Click to Enlarge)

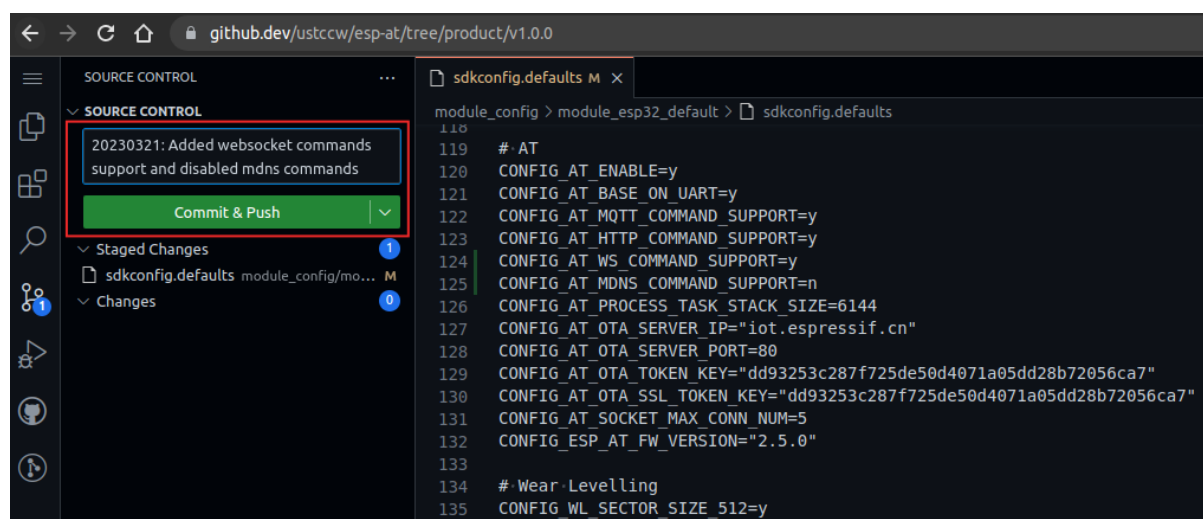


Fig. 19: Submit Changes (Click to Enlarge)

Note: If you want to enable or disable the AT firmware's *silence mode*, please refer to the [How to enable or disable silence mode](#) document.

6.2.7 Step 6. Compile the AT firmware using GitHub Actions

After you complete the above steps, GitHub Actions will automatically trigger the compilation of your ESP-AT firmware. After compilation, please refer to [How to Download the Latest Temporary Version of AT Firmware from GitHub](#) to download the AT firmware you need. Note that the firmware is generated based on your own esp-at project, not the <https://github.com/espressif/esp-at> project. AT firmware on GitHub will automatically expire after 90 days, so please save the downloaded AT firmware.

6.3 How to Set AT Port Pins

This document introduces how to modify *AT port* pins in the firmware for ESP32 series of products. By default, ESP-AT uses two UART interfaces as AT ports: one is to output logs (named as log port below) and the other to send AT commands and receive responses (named as command port below).

If you already have the ESP32 AT firmware and only need to modify the command port pins, you can directly modify the firmware using the [at.py Tool](#) to generate a new firmware for your module. Otherwise, if you need to modify the AT port pins of your ESP32 device, you need to complete the following steps:

- [clone the ESP-AT project](#);
- modify the pins either in the menuconfig utility or the factory_param_data.csv table;
- [compile the project](#) to generate the new bin in build/customized_partitions/factory_param.bin;
- [flash the new bin to your device](#).

This document focuses on modifying the pins. Click the links above for details of other steps.

Note: To use other interfaces as the AT command port, please refer to [AT through SDIO](#) , [AT through SPI](#) , or [AT through socket](#) for more details.

6.3.1 ESP32 Series

The log port and command port pins of ESP32 AT firmware can be user-defined to other pins. Refer to [ESP32 Technical Reference Manual](#) for the pins you can use.

Modify Log Port Pins

By default, the ESP32 AT firmware provided by Espressif uses the following UART0 pins to output log:

- TX: GPIO1
- RX: GPIO3

When compiling your ESP-AT project, you can modify them to other pins with the menuconfig utility:

- `./build.py menuconfig>Component config>ESP System Settings>Channel for console output>Custom UART`
- `./build.py menuconfig>Component config>ESP System Settings>UART TX on GPIO#`

- `./build.py menuconfig>Component config>ESP System Settings>UART RX on GPIO#`

Modify Command Port Pins

By default, UART1 is used to send AT commands and receive AT responses, and its pins are defined in Column `uart_port`, `uart_tx_pin`, `uart_rx_pin`, `uart_cts_pin`, and `uart_rts_pin` of the [factory_param_data.csv](#).

You can change them directly in your `factory_param_data.csv` table:

- Open your local `factory_param_data.csv` file.
- Locate the row of your module.
- Set `uart_port` as needed. (If you want to use the AT log port as the AT command port as well, you need to modify this line, and ensure that the `uart_tx_pin` and `uart_rx_pin` below have the same pins as the AT log port)
- Set `uart_tx_pin` and `uart_rx_pin` as needed. (Make sure that the pins you are going to modify are not being used by other functions, including the AT log port)
- Set `uart_cts_pin` and `uart_rts_pin` to be -1 if you do not use the hardware flow control function.
- Save the table.

6.4 How to Add User-defined AT Commands

This document provides guidance on how to add custom AT commands. It uses the example of [at_custom_cmd](#) to demonstrate each step with the sample code.

To define a basic, functional command, at least the following six steps are required:

- [Step 1: Define AT Commands](#)
- [Step 2: Register AT Command Functions](#)
- [Step 3: Add Component Dependencies](#)
- [Step 4: Add Link Options](#)
- [Step 5: Set Component Environment Variables](#)
- [Step 6: Compile the AT Firmware](#)

After completing the above steps, please *execute the AT+TEST command to get result*.

To customize relatively complex commands, please refer to the sample codes below:

- [Define Return Values](#)
- [Access Command Parameters](#)
- [Omit Command Parameters](#)
- [Block Command Execution](#)
- [Access Input Data from AT Command Port](#)

The source code for the AT command set is not open source and is presented in the form of [library files](#), which is also the basis for parsing custom AT commands.

6.4.1 Customize AT Commands

Step 1: Define AT Commands

You can define AT commands in the files `at_custom_cmd.c` and `at_custom_cmd.h`, or create new source files and header files in the directories [examples/at_custom_cmd/custom](#) and [examples/at_custom_cmd/include](#) to define AT commands.

Before customizing AT commands, please first determine the name and type of the AT command.

Command naming rules:

- Start a command with + character.
- Alphabetic characters (A~Z, a~z), numeric characters (0~9), and some other characters (!, %, -, ., /, :, _) are supported. See [AT Command Types](#) for more information.

Command types:

Each AT command can have up to four types: Test Command, Query Command, Set Command, and Execute Command. See [AT Command Types](#) for more information.

Then, define desired type of command. Assuming that AT+TEST supports all the four types. Below is the code to define the name and types of the AT command, as well as sample code to define each type.

- First, call `esp_at_cmd_struct` to define the name and type(s) that your AT command supports. The sample code below defined the name +TEST (omitting AT) and all the four types.

```
static const esp_at_cmd_struct at_custom_cmd[] = {
    {"+TEST", at_test_cmd_test, at_query_cmd_test, at_setup_cmd_test,
    ↪at_exe_cmd_test},
    /**
     * @brief You can define your own AT commands here.
     */
};
```

Note: If you do not want to define a particular type, set it to NULL.

- Test Command:

```
static uint8_t at_test_cmd_test(uint8_t *cmd_name)
{
    uint8_t buffer[64] = {0};
    snprintf((char *)buffer, 64, "test command: <AT%s=?> is executed\r\
    ↪n", cmd_name);
    esp_at_port_write_data(buffer, strlen((char *)buffer));

    return ESP_AT_RESULT_CODE_OK;
}
```

- Query Command:

```
static uint8_t at_query_cmd_test(uint8_t *cmd_name)
{
    uint8_t buffer[64] = {0};
    snprintf((char *)buffer, 64, "query command: <AT%s?> is executed\r\
    ↪n", cmd_name);
    esp_at_port_write_data(buffer, strlen((char *)buffer));

    return ESP_AT_RESULT_CODE_OK;
}
```

- Set Command:

```
static uint8_t at_setup_cmd_test(uint8_t para_num)
{
    uint8_t index = 0;

    // get first parameter, and parse it into a digit
    int32_t digit = 0;
    if (esp_at_get_para_as_digit(index++, &digit) != ESP_AT_PARA_PARSE_
    ↪RESULT_OK) {
        return ESP_AT_RESULT_CODE_ERROR;
    }
}
```

(continues on next page)

(continued from previous page)

```

// get second parameter, and parse it into a string
uint8_t *str = NULL;
if (esp_at_get_para_as_str(index++, &str) != ESP_AT_PARA_PARSE_
↪RESULT_OK) {
    return ESP_AT_RESULT_CODE_ERROR;
}

// allocate a buffer and construct the data, then send the data to_
↪mcu via interface (uart/spi/sdio/socket)
uint8_t *buffer = (uint8_t *)malloc(512);
if (!buffer) {
    return ESP_AT_RESULT_CODE_ERROR;
}
int len = snprintf((char *)buffer, 512, "setup command: <AT%s=%d,\"
↪%s\"> is executed\r\n",
    esp_at_get_current_cmd_name(), digit, str);
esp_at_port_write_data(buffer, len);

// remember to free the buffer
free(buffer);

return ESP_AT_RESULT_CODE_OK;
}

```

- Execute Command:

```

static uint8_t at_exe_cmd_test(uint8_t *cmd_name)
{
    uint8_t buffer[64] = {0};
    snprintf((char *)buffer, 64, "execute command: <AT%s> is executed\
↪r\n", cmd_name);
    esp_at_port_write_data(buffer, strlen((char *)buffer));

    return ESP_AT_RESULT_CODE_OK;
}

```

Step 2: Register AT Command Functions

- Please define the `esp_at_custom_cmd_register` function and call the API `esp_at_custom_cmd_array_regist()` to register AT commands.

Sample code:

```

bool esp_at_custom_cmd_register(void)
{
    return esp_at_custom_cmd_array_regist(at_custom_cmd, sizeof(at_custom_cmd)
↪/ sizeof(esp_at_cmd_struct));
}

```

- Then, call the API `ESP_AT_CMD_SET_INIT_FN` to initialize your implemented registration AT command function `esp_at_custom_cmd_register`.

Sample code:

```
ESP_AT_CMD_SET_INIT_FN(esp_at_custom_cmd_register, 1);
```

Note: To customize AT commands in the examples/at_custom_cmd/custom and examples/at_custom_cmd/include directories, please avoid naming the registered AT command function `esp_at_custom_cmd_register`, as this function is already defined and initialized in the `at_custom_cmd` example. Instead, name it something like `esp_at_custom_cmd_register_foo`, and use `ESP_AT_CMD_SET_INIT_FN` to initialize it.

Step 3: Add Component Dependencies

If you use components other than `at`, `freertos`, `nvs_flash` during *Step 1: Define AT Commands*, please add these component dependencies in the `examples/at_custom_cmd/CMakeLists.txt` file. Otherwise, you can skip this step. For example, if you additionally use the `lwip` component, the sample code is as follows:

```
set(require_components at freertos nvs_flash lwip)
```

Step 4: Add Link Options

Please link the name of your custom *registered AT command function* as a link option to `${COMPONENT_LIB}` in the `examples/at_custom_cmd/CMakeLists.txt` file to ensure that the program can find this function at runtime. The sample code is as follows:

```
target_link_libraries(${COMPONENT_LIB} INTERFACE "-u esp_at_custom_cmd_register")
```

Note: If the name of the custom *registered AT command function* is `esp_at_custom_cmd_register_foo`, the sample code is as follows:

```
target_link_libraries(${COMPONENT_LIB} INTERFACE "-u esp_at_custom_cmd_register_foo
↪")
```

Step 5: Set Component Environment Variables

This section introduces two methods for setting the `at_custom_cmd` component environment variables to ensure that the ESP-AT project can locate this component correctly during compilation. Choose the method that best suits your needs. If you customize AT commands or modify code in the original components under the `esp-at/components` directory, you do not need to perform this step. However, it is not recommended to customize AT commands in the original components under the `esp-at/components` directory, and this document does not explain this.

Method 1: Set the `AT_CUSTOM_COMPONENTS` environment variable directly in the command line (Suitable for *local compilation*).

- Linux or macOS

```
export AT_CUSTOM_COMPONENTS=(path_of_at_custom_cmd)
```

- Windows

```
set AT_CUSTOM_COMPONENTS=(path_of_at_custom_cmd)
```

Note:

- Please replace `(path_of_at_custom_cmd)` with the actual absolute path of the `at_custom_cmd` directory.
 - You can specify multiple components. For example:

```
export AT_CUSTOM_COMPONENTS="~/prefix/my_path1 ~/prefix/my_path2"
```
-

Method 2: Add the code to set the `AT_CUSTOM_COMPONENTS` environment variable in the `esp-at/build.py` file's `setup_env_variables()` function. (Suitable for *local compilation* and *web compilation*). The sample code is as follows:

```
# set AT_CUSTOM_COMPONENTS
at_custom_cmd_path=os.path.join(os.getcwd(), 'examples/at_custom_cmd')
os.environ['AT_CUSTOM_COMPONENTS']=at_custom_cmd_path
```

Step 6: Compile the AT Firmware

After completing these steps, choose either [web compilation](#) or [local compilation](#) of the AT firmware according to your needs, and [flash](#) it to your device.

Execute the AT+TEST Command to Get Result

If you have followed the steps correctly, below is the execution result of the AT+TEST command you defined.

Test Command:

```
AT+TEST=?
```

Response:

```
AT+TEST=?
test command: <AT+TEST=?> is executed

OK
```

Query Command:

```
AT+TEST?
```

Response:

```
AT+TEST?
query command: <AT+TEST?> is executed

OK
```

Set Command:

```
AT+TEST=1,"espressif"
```

Response:

```
AT+TEST=1,"espressif"
setup command: <AT+TEST=1,"espressif"> is executed

OK
```

Execute Command:

```
AT+TEST
```

Response:

```
AT+TEST
execute command: <AT+TEST> is executed

OK
```

6.4.2 Customize Complex AT Commands

The sample codes below are used to customize more complex commands, from which you can choose based on personal needs.

Define Return Values

ESP-AT has defined return values in [esp_at_result_code_string_index](#). See [AT Messages](#) for more return values.

In addition to output return values through the return mode, you can also use API [esp_at_response_result\(\)](#) to output the execution result of the command. [ESP_AT_RESULT_CODE_SEND_OK](#) and [ESP_AT_RESULT_CODE_SEND_FAIL](#) can be used with the API in code.

For example, when you send data to the server or MCU with the Execute Command of AT+TEST, you can use [esp_at_response_result\(\)](#) to output the sending result, and the return mode to output the command execution result. Below is the sample code:

```
uint8_t at_exe_cmd_test(uint8_t *cmd_name)
{
    uint8_t buffer[64] = {0};

    snprintf((char *)buffer, 64, "this cmd is execute cmd: %s\r\n", cmd_name);

    esp_at_port_write_data(buffer, strlen((char *)buffer));

    // user-defined operation of sending data to server or MCU
    send_data_to_server();

    // output SEND OK
    esp_at_response_result(ESP_AT_RESULT_CODE_SEND_OK);

    return ESP_AT_RESULT_CODE_OK;
}
```

How it works out:

```
AT+TEST
this cmd is execute cmd: +TEST

SEND OK

OK
```

Access Command Parameters

ESP-AT provides two APIs to access command parameters:

- [esp_at_get_para_as_digit\(\)](#) obtains digital parameters.
- [esp_at_get_para_as_str\(\)](#) obtains string parameters.

See [Set Command](#) for an example.

Omit Command Parameters

This section describes how to provide optional command parameters:

- [Omit the First or Middle Parameter](#)
- [Omit the Last Parameter](#)

Omit the First or Middle Parameter Let's say you want to make <param_2> and <param_3> of AT+TEST optional. <param_2> is a digital parameter, and <param_3> a string parameter.

```
AT+TEST=<param_1>[, <param_2>][, <param_3>], <param_4>
```

Below is the sample code to achieve it:

```
uint8_t at_setup_cmd_test(uint8_t para_num)
{
    int32_t para_int_1 = 0;
    int32_t para_int_2 = 0;
    uint8_t *para_str_3 = NULL;
    uint8_t *para_str_4 = NULL;
    uint8_t num_index = 0;
    uint8_t buffer[64] = {0};
    esp_at_para_parse_result_type parse_result = ESP_AT_PARA_PARSE_RESULT_OK;

    snprintf((char *)buffer, 64, "this cmd is setup cmd and cmd num is: %u\r\n",
    ↪para_num);
    esp_at_port_write_data(buffer, strlen((char *)buffer));

    parse_result = esp_at_get_para_as_digit(num_index++, &para_int_1);
    if (parse_result != ESP_AT_PARA_PARSE_RESULT_OK) {
        return ESP_AT_RESULT_CODE_ERROR;
    } else {
        memset(buffer, 0, 64);
        snprintf((char *)buffer, 64, "first parameter is: %d\r\n", para_int_1);
        esp_at_port_write_data(buffer, strlen((char *)buffer));
    }

    parse_result = esp_at_get_para_as_digit(num_index++, &para_int_2);
    if (parse_result != ESP_AT_PARA_PARSE_RESULT_OMITTED) {
        if (parse_result != ESP_AT_PARA_PARSE_RESULT_OK) {
            return ESP_AT_RESULT_CODE_ERROR;
        } else {
            // sample code
            // user needs to customize the operation
            memset(buffer, 0, 64);
            snprintf((char *)buffer, 64, "second parameter is: %d\r\n", para_int_
    ↪2);
            esp_at_port_write_data(buffer, strlen((char *)buffer));
        }
    } else {
        // sample code
        // the second parameter is omitted
        // user needs to customize the operation
        memset(buffer, 0, 64);
        snprintf((char *)buffer, 64, "second parameter is omitted\r\n");
        esp_at_port_write_data(buffer, strlen((char *)buffer));
    }

    parse_result = esp_at_get_para_as_str(num_index++, &para_str_3);
    if (parse_result != ESP_AT_PARA_PARSE_RESULT_OMITTED) {
        if (parse_result != ESP_AT_PARA_PARSE_RESULT_OK) {
            return ESP_AT_RESULT_CODE_ERROR;
        } else {
            // sample code
            // user needs to customize the operation
            memset(buffer, 0, 64);
            snprintf((char *)buffer, 64, "third parameter is: %s\r\n", para_str_3);
            esp_at_port_write_data(buffer, strlen((char *)buffer));
        }
    }
}
```

(continues on next page)

(continued from previous page)

```

    } else {
        // sample code
        // the third parameter is omitted
        // user needs to customize the operation
        memset(buffer, 0, 64);
        snprintf((char *)buffer, 64, "third parameter is omitted\r\n");
        esp_at_port_write_data(buffer, strlen((char *)buffer));
    }

    parse_result = esp_at_get_para_as_str(num_index++, &para_str_4);
    if (parse_result != ESP_AT_PARA_PARSE_RESULT_OK) {
        return ESP_AT_RESULT_CODE_ERROR;
    } else {
        memset(buffer, 0, 64);
        snprintf((char *)buffer, 64, "fourth parameter is: %s\r\n", para_str_4);
        esp_at_port_write_data(buffer, strlen((char *)buffer));
    }

    return ESP_AT_RESULT_CODE_OK;
}

```

Note: If the string parameter input is "", it is not omitted.

Omit the Last Parameter Let's say you want to make the string parameter <param_3> of AT+TEST optional, which is also the last parameter.

```
AT+TEST=<param_1>,<param_2>[,<param_3>]
```

There are two cases of omission:

- AT+TEST=<param_1>,<param_2>
- AT+TEST=<param_1>,<param_2>,

Below is the sample code to achieve it:

```

uint8_t at_setup_cmd_test(uint8_t para_num)
{
    int32_t para_int_1 = 0;
    uint8_t *para_str_2 = NULL;
    uint8_t *para_str_3 = NULL;
    uint8_t num_index = 0;
    uint8_t buffer[64] = {0};
    esp_at_para_parse_result_type parse_result = ESP_AT_PARA_PARSE_RESULT_OK;

    snprintf((char *)buffer, 64, "this cmd is setup cmd and cmd num is: %u\r\n",
    ↪ para_num);
    esp_at_port_write_data(buffer, strlen((char *)buffer));

    parse_result = esp_at_get_para_as_digit(num_index++, &para_int_1);
    if (parse_result != ESP_AT_PARA_PARSE_RESULT_OK) {
        return ESP_AT_RESULT_CODE_ERROR;
    } else {
        memset(buffer, 0, 64);
        snprintf((char *)buffer, 64, "first parameter is: %d\r\n", para_int_1);
        esp_at_port_write_data(buffer, strlen((char *)buffer));
    }

    parse_result = esp_at_get_para_as_str(num_index++, &para_str_2);
    if (parse_result != ESP_AT_PARA_PARSE_RESULT_OK) {

```

(continues on next page)

(continued from previous page)

```

    return ESP_AT_RESULT_CODE_ERROR;
} else {
    memset(buffer, 0, 64);
    snprintf((char *)buffer, 64, "second parameter is: %s\r\n", para_str_2);
    esp_at_port_write_data(buffer, strlen((char *)buffer));
}

if (num_index == para_num) {
    memset(buffer, 0, 64);
    snprintf((char *)buffer, 64, "third parameter is omitted\r\n");
    esp_at_port_write_data(buffer, strlen((char *)buffer));
} else {
    parse_result = esp_at_get_para_as_str(num_index++, &para_str_3);
    if (parse_result != ESP_AT_PARA_PARSE_RESULT_OMITTED) {
        if (parse_result != ESP_AT_PARA_PARSE_RESULT_OK) {
            return ESP_AT_RESULT_CODE_ERROR;
        } else {
            // sample code
            // user needs to customize the operation
            memset(buffer, 0, 64);
            snprintf((char *)buffer, 64, "third parameter is: %s\r\n", para_
↪str_3);
            esp_at_port_write_data(buffer, strlen((char *)buffer));
        }
    } else {
        // sample code
        // the third parameter is omitted
        // user needs to customize the operation
        memset(buffer, 0, 64);
        snprintf((char *)buffer, 64, "third parameter is omitted\r\n");
        esp_at_port_write_data(buffer, strlen((char *)buffer));
    }
}

return ESP_AT_RESULT_CODE_OK;
}

```

Note: If the string parameter input is "", it is not omitted.

Block Command Execution

Sometimes you want to block the execution of one command to wait for another execution result, and the system may return different values according to the result.

Generally, this kind of command needs to synchronize the results of other tasks.

semaphore is recommended to handle synchronization.

The sample code is as follows:

```

xSemaphoreHandle at_operation_sema = NULL;

uint8_t at_exe_cmd_test(uint8_t *cmd_name)
{
    uint8_t buffer[64] = {0};

    snprintf((char *)buffer, 64, "this cmd is execute cmd: %s\r\n", cmd_name);

    esp_at_port_write_data(buffer, strlen((char *)buffer));
}

```

(continues on next page)

(continued from previous page)

```

// sample code
// users do not have to create semaphores here
at_operation_sema = xSemaphoreCreateBinary();
assert(at_operation_sema != NULL);

// block command execution
// wait for another execution result
// other tasks can call xSemaphoreGive to release the semaphore
xSemaphoreTake(at_operation_sema, portMAX_DELAY);

return ESP_AT_RESULT_CODE_OK;
}

```

Access Input Data from AT Command Port

ESP-AT supports accessing input data from AT Command port. It provides two APIs for this purpose.

- `esp_at_port_enter_specific()` sets the callback function which will be called by AT port after receiving the input data.
- `esp_at_port_exit_specific()` deletes the callback function set by `esp_at_port_enter_specific`.

Approaches to access the data vary depending on whether the data length has been specified or not.

Input Data of Specified Length Assuming that you have specified the data length in `<param_1>` as follows:

```
AT+TEST=<param_1>
```

Below is the sample to access the input data of `<param_1>` length from AT Command Port:

```

static xSemaphoreHandle at_sync_sema = NULL;

void wait_data_callback(void)
{
    xSemaphoreGive(at_sync_sema);
}

uint8_t at_setup_cmd_test(uint8_t para_num)
{
    int32_t specified_len = 0;
    int32_t received_len = 0;
    int32_t remain_len = 0;
    uint8_t *buf = NULL;
    uint8_t buffer[64] = {0};

    if (esp_at_get_para_as_digit(0, &specified_len) != ESP_AT_PARA_PARSE_RESULT_
    OK) {
        return ESP_AT_RESULT_CODE_ERROR;
    }

    buf = (uint8_t *)malloc(specified_len);
    if (buf == NULL) {
        memset(buffer, 0, 64);
        snprintf((char *)buffer, 64, "malloc failed\r\n");
        esp_at_port_write_data(buffer, strlen((char *)buffer));
    }

    // sample code

```

(continues on next page)

(continued from previous page)

```

// users do not have to create semaphores here
if (!at_sync_sema) {
    at_sync_sema = xSemaphoreCreateBinary();
    assert(at_sync_sema != NULL);
}

// output input prompt ">"
esp_at_port_write_data((uint8_t *)>", strlen(">"));

// set the callback function which will be called by AT port after receiving
the input data
esp_at_port_enter_specific(wait_data_callback);

// receive input data
while(xSemaphoreTake(at_sync_sema, portMAX_DELAY)) {
    received_len += esp_at_port_read_data(buf + received_len, specified_len -
received_len);

    if (specified_len == received_len) {
        esp_at_port_exit_specific();

        // get the length of the remaining input data
        remain_len = esp_at_port_get_data_length();
        if (remain_len > 0) {
            // sample code
            // if the remaining data length > 0, the actual input data length
is greater than the specified received data length
            // users can customize the operation to process the remaining data
            // here is just a simple print out of the remaining data
            esp_at_port_recv_data_notify(remain_len, portMAX_DELAY);
        }

        // sample code
        // output received data
        memset(buffer, 0, 64);
        snprintf((char *)buffer, 64, "\r\nreceived data is: ");
        esp_at_port_write_data(buffer, strlen((char *)buffer));

        esp_at_port_write_data(buf, specified_len);

        break;
    }
}

free(buf);

return ESP_AT_RESULT_CODE_OK;
}

```

So, if you set AT+TEST=5 and the input data is 1234567890, the ESP-AT output is as follows.

```

AT+TEST=5
>67890
received data is: 12345
OK

```

Input Data of Unspecified Length This scenario is similar to the Network *Passthrough Mode*. You do not specify the data length.

AT+TEST

Assuming that ESP-AT ends the execution of the command and returns the execution result, the sample code is as follows:

```
#define BUFFER_LEN (2048)
static xSemaphoreHandle at_sync_sema = NULL;

void wait_data_callback(void)
{
    xSemaphoreGive(at_sync_sema);
}

uint8_t at_exe_cmd_test(uint8_t *cmd_name)
{
    int32_t received_len = 0;
    int32_t remain_len = 0;
    uint8_t *buf = NULL;
    uint8_t buffer[64] = {0};

    buf = (uint8_t *)malloc(BUFFER_LEN);
    if (buf == NULL) {
        memset(buffer, 0, 64);
        snprintf((char *)buffer, 64, "malloc failed\r\n");
        esp_at_port_write_data(buffer, strlen((char *)buffer));
    }

    // sample code
    // users do not have to create semaphores here
    if (!at_sync_sema) {
        at_sync_sema = xSemaphoreCreateBinary();
        assert(at_sync_sema != NULL);
    }

    // output input prompt ">"
    esp_at_port_write_data((uint8_t *)>, strlen(">"));

    // set the callback function which will be called by AT port after receiving
    // the input data
    esp_at_port_enter_specific(wait_data_callback);

    // receive input data
    while(xSemaphoreTake(at_sync_sema, portMAX_DELAY)) {
        memset(buf, 0, BUFFER_LEN);

        received_len = esp_at_port_read_data(buf, BUFFER_LEN);
        // check whether to exit the mode
        // the exit condition is the "+++" string received
        if ((received_len == 3) && (strcmp((const char *)buf, "+++", 3) == 0)) {
            esp_at_port_exit_specific();

            // sample code
            // if the remaining data length > 0, it means that there is still data
            // left in the buffer to be processed
            // users can customize the operation to process the remaining data
            // here is just a simple print out of the remaining data
            remain_len = esp_at_port_get_data_length();
            if (remain_len > 0) {
                esp_at_port_rcv_data_notify(remain_len, portMAX_DELAY);
            }
        }
    }
}
```

(continues on next page)

(continued from previous page)

```

        break;
    } else if (received_len > 0) {
        // sample code
        // users can customize the operation to process the received data
        // here is just a simple print received data
        memset(buffer, 0, 64);
        snprintf((char *)buffer, 64, "\r\nreceived data is: ");
        esp_at_port_write_data(buffer, strlen((char *)buffer));

        esp_at_port_write_data(buf, strlen((char *)buf));
    }
}

free(buf);

return ESP_AT_RESULT_CODE_OK;
}

```

So, if the first input data is 1234567890, and the second input data is +++, the ESP-AT output is as follows:

```

AT+TEST
>
received data is: 1234567890
OK

```

6.5 How to Improve ESP-AT Throughput Performance

By default, UART is used for communication between ESP-AT and the host MCU, so the maximum throughput speed will not exceed its default configuration, that is, 115200 bps. If users want ESP-AT to achieve high throughput, it is necessary to understand this document and choose the appropriate configuration method. Taking ESP32 as an example, this document introduces how to improve the throughput performance of ESP-AT.

Note: This document describes general methods to improve TCP/UDP/SSL throughput performance when ESP-AT is in *Passthrough Mode*.

Users could choose one of the following methods to improve throughput performance:

- *(Simple) Quick Configuration*
- *(Recommended) Understand Data Stream and Make the Precise Configuration*

6.5.1 (Simple) Quick Configuration

1. Configure system, LWIP, Wi-Fi parameters

Copy the following code snippet and append to tail of [module_config/module_esp32_default/sdkconfig.defaults](#) file. For other ESP32 series devices, please modify the sdkconfig.defaults file in the corresponding folder.

```

# System
CONFIG_ESP_SYSTEM_EVENT_TASK_STACK_SIZE=4096
CONFIG_FREERTOS_UNICORE=n
CONFIG_FREERTOS_HZ=1000
CONFIG_ESP_DEFAULT_CPU_FREQ_240=y
CONFIG_ESP_DEFAULT_CPU_FREQ_MHZ=240

```

(continues on next page)

(continued from previous page)

```

CONFIG_ESPTOOLPY_FLASHMODE_QIO=y
CONFIG_ESPTOOLPY_FLASHFREQ_80M=y

# LWIP
CONFIG_LWIP_TCP_SND_BUF_DEFAULT=65534
CONFIG_LWIP_TCP_WND_DEFAULT=65534
CONFIG_LWIP_TCP_RECVMBOX_SIZE=12
CONFIG_LWIP_UDP_RECVMBOX_SIZE=12
CONFIG_LWIP_TCPIP_RECVMBOX_SIZE=64

# Wi-Fi
CONFIG_ESP32_WIFI_STATIC_RX_BUFFER_NUM=16
CONFIG_ESP32_WIFI_DYNAMIC_RX_BUFFER_NUM=64
CONFIG_ESP32_WIFI_DYNAMIC_TX_BUFFER_NUM=64
CONFIG_ESP32_WIFI_TX_BA_WIN=32
CONFIG_ESP32_WIFI_RX_BA_WIN=32
CONFIG_ESP32_WIFI_AMPDU_TX_ENABLED=y
CONFIG_ESP32_WIFI_AMPDU_RX_ENABLED=y

```

2. Enlarge UART buffer size

Copy the following code snippet and replace the `uart_driver_install()` line of `at_uart_task.c` file.

```

uart_driver_install(g_at_cmd_port, 1024 * 16, 1024 * 16, 100, &s_at_uart_
→queue, 0);

```

3. Delete, Build, Flash, Run

```

rm -rf build sdkconfig
./build.py build
./build.py flash monitor

```

4. Increase UART baud rate before entering passthrough mode

A typical ESP-AT commands sequence is as follows:

```

AT+CWMODE=1
AT+CWJAP="ssid","password"
AT+UART_CUR=3000000,8,1,0,3
AT+CIPSTART="TCP","192.168.105.13",3344
AT+CIPMODE=1
AT+CIPSEND
// data transmission

```

This simple and quick configuration method can improve the throughput to a certain extent, but sometimes it might not meet the expectations of users. In addition, some configurations may not hit the bottleneck of throughput. Higher configurations may sacrifice memory resources or power consumption. Therefore, users could also familiarize themselves with the following recommended method and make the precise configuration.

6.5.2 (Recommended) Understand Data Stream and Make the Precise Configuration

The factors that generally affect ESP-AT throughput are illustrated in the following figure:

As shown by the arrows in the figure:

- The Data stream sent by ESP-AT is (TX): S1 -> S2 -> S3 -> S4 -> S5 -> S6 -> S7 -> S8
- The Data stream received by ESP-AT is (RX): R8 -> R7 -> R6 -> R5 -> R4 -> R3 -> R2 -> R1

The data stream of throughput is similar to water flow. In order to improve throughput, it is necessary to consider optimizing between nodes with low data flow rate rather than making additional configuration between nodes with expected data flow rate, so as to avoid unnecessary waste of resources. In actual products, usually, users only need to improve the throughput of one data stream. So here, users need to configure it according to the following instructions.

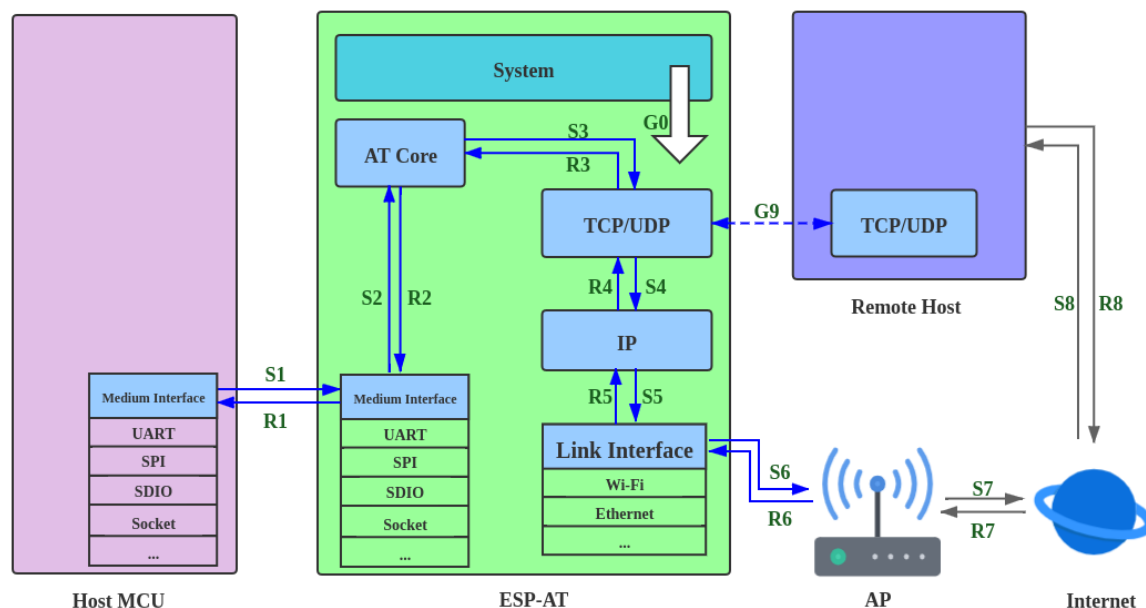


Fig. 20: Data Stream in Throughput

Note: The following configurations are based on sufficient available memory. Users can query the available memory through the AT command: `AT+SYSRAM`.

1. G0 throughput optimization

G0 is a part of the system that can be optimized. The recommended configuration is as follows:

```
CONFIG_ESP_SYSTEM_EVENT_TASK_STACK_SIZE=4096
CONFIG_FREERTOS_UNICORE=n
CONFIG_FREERTOS_HZ=1000
CONFIG_ESP_DEFAULT_CPU_FREQ_240=y
CONFIG_ESP_DEFAULT_CPU_FREQ_MHZ=240
CONFIG_ESPTOOLPY_FLASHMODE_QIO=y
CONFIG_ESPTOOLPY_FLASHFREQ_80M=y
```

2. S1, R1 throughput optimization

Generally, S1 and R1 are the key to the throughput of ESP-AT. Because UART is used for communication between ESP-AT and the host MCU by default, and the baud rate is 115200. On the hardware, the baud rate upper limit is 5 Mbps. Therefore, if the throughput is expected to be less than 5 Mbps, the user can use the default UART as the communication medium with the host MCU, and the following optimization methods can be carried out.

2.1 Enlarge UART buffer size

Copy the following code snippet and replace the `uart_driver_install()` line of `at_uart_task.c` file.

- Improve UART TX throughput

```
uart_driver_install(g_at_cmd_port, 1024 * 16, 8192, 100, &s_at_uart_
↪queue, 0);
```

- Improve UART RX throughput

```
uart_driver_install(g_at_cmd_port, 2048, 1024 * 16, 100, &s_at_uart_
↪queue, 0);
```

- Improve UART TX and RX throughput

```
uart_driver_install(g_at_cmd_port, 1024 * 16, 1024 * 16, 100, &s_at_  
↳uart_queue, 0);
```

2.2 Increase UART baud rate before entering passthrough mode

A typical ESP-AT commands sequence is as follows:

```
AT+CWMODE=1  
AT+CWJAP="ssid", "password"  
AT+UART_CUR=3000000, 8, 1, 0, 3  
AT+CIPSTART="TCP", "192.168.105.13", 3344  
AT+CIPMODE=1  
AT+CIPSEND  
// data transmission
```

Note: The user needs to ensure that the UART of the host MCU can support such a high rate, and the UART connection between the host MCU and ESP-AT is as short as possible.

Note: If the user expects the throughput rate to be greater than or close to 5 Mbps, then SPI, SDIO, Socket or other methods can be considered. Please refer to:

- SDIO: [SDIO AT Guide](#)
 - Socket: [Socket AT Guide](#)
-

3. S2, R2, R3, S3 throughput optimization

Generally, S2, R2, R3, S3 are not the bottleneck of ESP-AT throughput. Because AT core transfers data between UART buffer and the transport layer of communication protocol, where has minimal and non-time-consuming application logic, there is no need to optimize them.

4. S4, R4, S5, R5, S6, R6 throughput optimization

If UART is used for communication between ESP-AT and host MCU, S4, R4, S5, R5, S6, R6 need not be optimized. If other transmission media are used, S4, R4, S5, R5, S6, R6 should be a factor affecting throughput.

S4, R4, S5, R5, S6, R6 is the data stream between the transport layer, network layer and data link layer of the communication protocol. Users need to read [How to improve Wi-Fi performance](#) in ESP-IDF to understand the principle and make reasonable configuration. These configurations can be configured in `./build.py menuconfig`.

- Improve throughput of S4 -> S5 -> S6: [TX direction](#)
- Improve throughput of R6 -> R5 -> R4: [RX direction](#)

5. S6, R6 throughput optimization

S6 and R6 are the data link layers of the communication protocol. ESP32 can use Wi-Fi or ethernet as the transmission medium. In addition to the optimization methods described above, Wi-Fi throughput optimization may also need users' attention:

- Improve RF Power
The default RF power is usually not the bottleneck of throughput. Users could query and set RF power through AT command: [AT+RFPPOWER](#).
- Set 802.11 b/g/n protocol
The default Wi-Fi mode is 802.11 b/g/n protocol. Users could query and set 802.11 b/g/n protocol through AT command: [AT+CWSTAPROTO](#). The configuration is bidirectional. Therefore, it is recommended that the Wi-Fi mode of AP is configured as 802.11 b/g/n protocol and the bandwidth mode of AP is configured as HT20/HT40 (20/40 MHz) mode.

6. S7, R7, S8, R8 throughput optimization

Generally, S7, R7, S8, R8 are not the scope of ESP-AT throughput optimization because this is related to the actual network bandwidth, network routing, physical distance, etc.

6.6 How to Update mfg_nvs Partition

6.6.1 mfg_nvs Partition

mfg_nvs (*manufacturing nvs*) partition is defined in `esp-at/module_config/{module_name}/at_customize.csv` file. This partition stores the following configurations of factory firmware:

- Factory parameter configurations (Wi-Fi configurations, UART configurations, module configurations): Please refer to [Factory Parameter Configuration](#).
- PKI configurations (various certificate and key configurations): Please refer to [Introduction to PKI Configuration](#).
- GATTS configuration (Bluetooth LE services): Please refer to [Bluetooth LE Service Source File](#).

When you need to modify the factory parameter configurations, PKI configurations, or GATTS configurations, you can recompile the ESP-AT project to generate a new mfg_nvs.bin file; Alternatively, through [at.py Tool](#) modify the firmware to generate a new firmware for your module. This article introduces the former.

6.6.2 Generate mfg_nvs.bin

After the configuration modifications are completed, [re-build the project](#). During the compilation stage, the `mfg_nvs.py` script is automatically used to generate the `build/customized_partitions/mfg_nvs.csv` file (which contains the namespace, key, type, and value information for all configurations). Then, the `nvs_partition_gen.py` script in `esp-idf` is used to generate the `build/customized_partitions/mfg_nvs.bin` file (which represents the structure of the NVS library ([NVS flash API reference](#))). Then, you can download the mfg_nvs.bin file to the device.

Download mfg_nvs.bin

You can download the mfg_nvs.bin in one of the following ways.

- Download the recompiled ESP-AT firmware. See [Step 7. Flash onto the Device](#) for more information.
- Download mfg_nvs.bin only. This way only updates the mfg_nvs area in the device.
 - Windows
Please download the Windows [Flash Download Tool](#). You can refer to the [Flash Download Tool User Guide](#) for instructions. Download the `build/customized_partitions/mfg_nvs.bin` file to ESP32. You can use the `AT+SYSFLASH?` command to query the download address of mfg_nvs.bin.
 - Linux or macOS
Please use `esptool.py`.
You can execute the following command in the ESP-AT root directory to download the mfg_nvs.bin file.

```
esptool.py --chip auto --port PORTNAME --baud 921600 --before default_  
↪reset --after hard_reset write_flash -z --flash_mode dio --flash_freq_  
↪40m --flash_size 4MB ADDRESS mfg_nvs.bin
```

Replace PORTNAME with your serial port name. Replace ADDRESS with the address for downloading the mfg_nvs.bin file. You can use the `AT+SYSFLASH?` command to query the download address.

- MCU sends the `AT+SYSFLASH` command to update the mfg_nvs partition of ESP32.


```
# Erase the mfg_nvs partition
AT+SYSFLASH=0, "mfg_nvs", 0, MFG_NVS_SIZE

# Write the mfg_nvs.bin file
AT+SYSFLASH=1, "mfg_nvs", 0, MFG_NVS_SIZE
```

Replace `MFG_NVS_SIZE` with the size of the downloaded `mfg_nvs.bin` file. Different modules have different partition sizes. You can use the [AT+SYSFLASH?](#) command to query the partition size.

6.7 How to Update Factory Parameters

This document describes how to update the default factory parameter configuration for ESP-AT by recompiling the firmware. The factory parameter configuration includes some Wi-Fi configurations, UART configurations, and module configurations.

Note: If you want to update factory parameters at runtime using AT commands without recompiling the firmware, please refer to [AT+SYSMFG command examples](#).

- [Factory Parameter Configuration](#)

6.7.1 Factory Parameter Configuration

The default factory parameters are configured in the source file [customized_partitions/raw_data/factory_param/factory_param_data.csv](#), as shown below:

Function		Current Configuration	Related AT Commands
Version Information		version (version number of the factory parameter structure, currently version 4, used to identify parameter format and compatibility)	AT+SYSMFG
Wi-Fi Configuration		- max_tx_power (Wi-Fi maximum transmission power for ESP32, see ESP32 Transmit Power for the setting range) - country_code (country code) - start_channel (starting Wi-Fi channel) - channel_num (total number of Wi-Fi channels)	All AT commands requiring Wi-Fi functionality
UART Configuration		- uart_port (UART port used for sending AT commands and receiving AT responses) - uart_baudrate (UART baud rate) - uart_tx_pin (UART TX pin) - uart_rx_pin (UART RX pin) - uart_cts_pin (UART CTS pin) - uart_rts_pin (UART RTS pin)	All AT commands requiring UART functionality
System Mode	Store	sys_store (default system store mode) 0: Do not automatically store configuration changes to flash 1: Automatically store configuration changes to flash (default)	AT+SYSSTORE
Module Name		module_name	AT+GMR

Please modify the factory parameters configurations according to your own needs and generate `mfg_nvs.bin` file.

Generate `mfg_nvs.bin`

Please refer to [Generate `mfg\_nvs.bin`](#) document to generate the `mfg_nvs.bin` file with the factory parameters configurations.

Download `mfg_nvs.bin`

Please refer to [Download `mfg\_nvs.bin`](#) document.

6.8 How to Update PKI Configuration

This document describes how to update the default [PKI](#) configuration provided by ESP-AT by recompiling the firmware. The PKI configuration includes certificates and keys for TLS clients, TLS servers, MQTT clients, Web-Socket clients, HTTP clients, and WPA2 Enterprise clients.

Note: If you want to update certificates at runtime using AT commands without recompiling the firmware, please refer to [AT+SYSMFG command examples](#).

- [Introduction to PKI Configuration](#)

6.8.1 Introduction to PKI Configuration

The source file for the current default PKI configuration is located in the [customized_partitions/raw_data](#) directory, as shown below:

Function	Current Configuration	Related AT Commands
TLS client	Set 0 client configuration • client_ca_00.crt • client_cert_00.crt • client_key_00.key Set 1 client configuration • client_ca_01.crt • client_cert_01.crt • client_key_01.key	• AT+CIPSTART • AT+CIPSSLCONF
TLS server	• server_ca.crt • server_cert.crt • server.key	AT+CIPSERVER
MQTT client	• mqtt_ca.crt • mqtt_client.crt • mqtt_client.key	AT+MQTTUSERCFG
WebSocket client	Set 0 client configuration • wss_ca_00.crt • wss_client_00.crt • wss_client_00.key Set 1 client configuration • wss_ca_01.crt • wss_client_01.crt • wss_client_01.key	AT+WSCFG
HTTP client	Set 0 client configuration • https_ca_00.crt • https_client_00.crt • https_client_00.key Set 1 client configuration • https_ca_01.crt • https_client_01.crt • https_client_01.key	AT+HTTPCFG
WPA2 Enterprise client	• wpa2_ca.pem • wpa2_client.crt • wpa2_client.key	AT+CWJEAP

Please modify the PKI configurations according to your own needs and generate `mfg_nvs.bin` file.

Generate `mfg_nvs.bin`

Please refer to [Generate `mfg\_nvs.bin`](#) document to generate the `mfg_nvs.bin` file with the PKI configurations.

Download `mfg_nvs.bin`

Please refer to [Download `mfg\_nvs.bin`](#) document.

6.9 How to Customize Bluetooth® LE Services

This document describes how to customize Bluetooth LE services on your ESP32 with the Bluetooth LE service source file provided by ESP-AT.

- [Bluetooth LE Service Source File](#)
- [Customize Bluetooth LE Services during Compilation](#)
 - [Modify the Bluetooth LE Service Source File](#)
 - [Generate `mfg\_nvs.bin`](#)
 - [Download `mfg\_nvs.bin`](#)

The Bluetooth LE services are defined as a multivariate array of GATT structures, and the array contains at least one primary service whose attribute type is defined as 0x2800. Each service always consists of a service definition and several characteristics. Each characteristic always consists of a value and optional descriptors. Please refer to Part Generic Attribute Profile (GATT) of [Bluetooth Core Specification](#) for more information.

6.9.1 Bluetooth LE Service Source File

The ESP-AT project creates Bluetooth LE services based on its Bluetooth LE service source file. It is located in `customized_partitions/raw_data/ble_data/gatts_data.csv`. The table below shows the default source file.

index	uuid_len	uuid	perm	val_max_len	val_cur_len	value
0	16	0x2800	0x01	2	2	A002
1	16	0x2803	0x01	1	1	2
2	16	0xC300	0x01	1	1	30
3	16	0x2901	0x11	1	1	30
...	...	...	...	...	...	...

Below are descriptions of the table above.

- `perm` field describes the permission. Its definition in the ESP-AT project is as follows:

```
/* relate to BTA_GATT_PERM_XXX in bta/bta_gatt_api.h */
/**
 * @brief Attribute permissions
 */
#define ESP_GATT_PERM_READ (1 << 0) /* bit 0 - 0x0001 */
↪ /* relate to BTA_GATT_PERM_READ in bta/bta_gatt_api.h */
#define ESP_GATT_PERM_READ_ENCRYPTED (1 << 1) /* bit 1 - 0x0002 */
↪ /* relate to BTA_GATT_PERM_READ_ENCRYPTED in bta/bta_gatt_api.h */
#define ESP_GATT_PERM_READ_ENC_MITM (1 << 2) /* bit 2 - 0x0004 */
```

(continues on next page)

(continued from previous page)

```

→ /* relate to BTA_GATT_PERM_READ_ENC_MITM in bta/bta_gatt_api.h */
#define ESP_GATT_PERM_WRITE (1 << 4) /* bit 4 - 0x0010 */
→ /* relate to BTA_GATT_PERM_WRITE in bta/bta_gatt_api.h */
#define ESP_GATT_PERM_WRITE_ENCRYPTED (1 << 5) /* bit 5 - 0x0020 */
→ /* relate to BTA_GATT_PERM_WRITE_ENCRYPTED in bta/bta_gatt_api.h */
#define ESP_GATT_PERM_WRITE_ENC_MITM (1 << 6) /* bit 6 - 0x0040 */
→ /* relate to BTA_GATT_PERM_WRITE_ENC_MITM in bta/bta_gatt_api.h */
#define ESP_GATT_PERM_WRITE_SIGNED (1 << 7) /* bit 7 - 0x0080 */
→ /* relate to BTA_GATT_PERM_WRITE_SIGNED in bta/bta_gatt_api.h */
#define ESP_GATT_PERM_WRITE_SIGNED_MITM (1 << 8) /* bit 8 - 0x0100 */
→ /* relate to BTA_GATT_PERM_WRITE_SIGNED_MITM in bta/bta_gatt_api.h */
#define ESP_GATT_PERM_READ_AUTHORIZATION (1 << 9) /* bit 9 - 0x0200 */
#define ESP_GATT_PERM_WRITE_AUTHORIZATION (1 << 10) /* bit 10 - 0x0400 */

```

- The first line of table is the service definition with a UUID of 0xA002.
- The second line is the declaration of a characteristic. UUID 0x2803 means the characteristic declaration. The value 2 sets the permission. The length of permission is 8 bits, and each bit represents permission for an operation. 1 indicates that the operation is supported, and 0 indicates not supported.

Bit	Permission
0	BROADCAST
1	READ
2	WRITE WITHOUT RESPONSE
3	WRITE
4	NOTIFY
5	INDICATE
6	AUTHENTICATION SIGNED WRITES
7	EXTENDED PROPERTIES

- The third line defines a characteristic of the service. UUID of this line is the characteristic's UUID, and value is the characteristic's value.
- The fourth line defines a descriptor of the characteristic (optional).

For more information about UUID, please refer to [Bluetooth Special Interest Group \(SIG\) Assigned Numbers](#).

If you use the default source file on your ESP32 without any modification and establish a Bluetooth LE connection, you will get the following result after querying the server service on the client side.

Unknown Service

UUID: 0000a002-0000-1000-8000-00805f9b34fb

PRIMARY SERVICE

Unknown Characteristic

UUID:

0000c300-0000-1000-8000-00805f9b34fb

Properties: READ

Value: (0x) 30, "0"

Descriptors:

Characteristic User Description

UUID: 0x2901

Value: 0



6.9.2 Customize Bluetooth LE Services during Compilation

If you want to customize the Bluetooth LE services, follow the steps below.

- [Modify the Bluetooth LE Service Source File](#)
- [Generate mfg_nvs.bin](#)
- [Download mfg_nvs.bin](#)

Modify the Bluetooth LE Service Source File

You can define more than one service. For example, if you want to define three services (Server_A, Server_B and Server_C), these three services need to be arranged in order. Since the definition of each service is similar, here we define one service as an example, and then you can define others one by one accordingly.

1. Add the service definition.

In this example, we define a primary service with a value of 0xFF01.

index	uuid_len	uuid	perm	val_max_len	val_cur_len	value
31	16	0x2800	0x01	2	2	FF01

2. Add the characteristic declaration and characteristic value.

In this example, we define a readable and writable characteristic with UUID 0xC300, and set its value to 0x30.

index	uuid_len	uuid	perm	val_max_len	val_cur_len	value
32	16	0x2803	0x11	1	1	0A
33	16	0xC300	0x11	1	1	30

3. Add the characteristic descriptor (optional).

In this example, we add client characteristic configuration. Its value 0x0000 represents notifications and indications are disabled.

index	uuid_len	uuid	perm	val_max_len	val_cur_len	value
34	16	0x2902	0x11	2	2	0000

After the above steps, the customized Bluetooth LE service has been defined as follows.

index	uuid_len	uuid	perm	val_max_len	val_cur_len	value
31	16	0x2800	0x01	2	2	FF01
32	16	0x2803	0x11	1	1	0A
33	16	0xC300	0x11	1	1	30
34	16	0x2902	0x11	2	2	0000

Please modify the GATTs configurations according to your own needs and generate `mfg_nvs.bin` file.

Generate mfg_nvs.bin

Please refer to [Generate mfg_nvs.bin](#) document to generate the `mfg_nvs.bin` file with the Low Energy Bluetooth services.

Download mfg_nvs.bin

Please refer to [Download mfg_nvs.bin](#) document.

After the download is complete, re-establish the Bluetooth LE connection. Query the server service on the client side as follows:



6.10 How to Customize Partitions

This document describes how to customize the partitions in your ESP32 by modifying the `at_customize.csv` table provided by ESP-AT. There are two partition tables: the primary partition and the secondary partition table.

The primary partition table `partitions_at.csv` is for system usage, based on which the `partitions_at.bin` file is generated. If the primary partition table goes wrong, the system will fail to startup. Therefore, it is not recommended to modify the `partitions_at.csv`.

ESP-AT provides a secondary partition table `at_customize.csv` that you can customize to store self-defined blocks of data. It is based on the primary partition table.

To modify the partition in your ESP32, please follow the first three steps. The fourth section illustrates the three steps with an example.

- [Modify `at\_customize.csv`](#)
- [Generate `at\_customize.bin`](#)
- [Flash `at\_customize.bin` into ESP32 Device](#)
- [Example](#)

6.10.1 Modify `at_customize.csv`

Find the `at_customize.csv` for your module with reference to the following table.

Table 1: at_customize.csv paths

Platform	Module	Path
ESP32	• WROOM-32 • PICO-D4 • SOLO-1 • MINI-1	module_config/module_esp32_default/at_customize.csv
ESP32	WROVER-32	module_config/module_wrover-32/at_customize.csv
ESP32	ESP32-D2WD	module_config/module_esp32-d2wd/at_customize.csv

Then, follow the rules below when modifying at_customize.csv.

- Do not change the Name and Type of the user partitions that have already been defined in it, while SubType, Offset, and Size can be changed.
- If you need to add a new user partition, please check if it has already been defined in the ESP-IDF (`esp_partition.h`) first.
 - If yes, you should keep the Type value the same as that of ESP-IDF.
 - If no, please set the Type to 0x40.
- A user partition's Name should not be longer than 16 bytes.
- The default size of the entire at_customize partition is defined in the partitions_at.csv table. Please do not exceed the range when adding new user partitions.

6.10.2 Generate at_customize.bin

After having modified the at_customize.csv, you can either recompile the ESP-AT project to generate the at_customize.bin file, or use the python script `gen_esp32part.py`.

If you use the script, execute the following command under the root directory of ESP-AT project and replace INPUT and OUTPUT:

```
python esp-idf/components/partition_table/gen_esp32part.py <INPUT> [OUTPUT]
```

- Replace INPUT with the path to at_customize.csv or the binary file to parse.
- Replace OUTPUT with the path to output converted binary or CSV file. Stdout will be used if omitted.

6.10.3 Flash at_customize.bin into ESP32 Device

Download the at_customize.bin into flash. Please refer to [Flash AT Firmware into Your Device](#) for how to flash bin files into ESP32 device and the following table for the download address for your module.

Table 2: Download Address of at_customize.bin in Modules

Platform	Module	Address	Size
ESP32	• WROOM-32 • WROVER-32 • PICO-D4 • SOLO-1 • MINI-1 • ESP32-D2WD	0x20000	0xE0000

There are cases where at_customize.bin must be downloaded to flash in order to use certain AT commands:

- [AT+SYSFLASH: Query or Set User Partitions in Flash](#)
- [AT+FS: Filesystem Operations](#)
- SSL server relevant commands
- BLE server relevant commands

6.10.4 Example

The section demonstrates how to add a 4 KB partition named `test` into the ESP32-WROOM-32 module.

Firstly, find the `at_customize.csv` table for ESP32-WROOM-32 and set the Name, Type, Subtype, Offset, and Size of the new partition:

```
# Name, Type, SubType, Offset, Size
... ..
test, 0x40, 15, 0x3D000, 4K
fs_storage, data, 0xff, 0x70000, 576K
```

Secondly, recompile the ESP-AT project, or execute the python script in the ESP-AT root directory to generate `at_customize.bin`.

```
python esp-idf/components/partition_table/gen_esp32part.py -q ./module_config/
↪ module_esp32_default/at_customize.csv at_customize.bin
```

Then, the `at_customize.bin` will be generated in the ESP-AT root directory.

Thirdly, download the `at_customize.bin` to flash.

Execute the following command under the root directory of ESP-AT project and replace PORT and BAUD.

```
python esp-idf/components/esptool_py/esptool/esptool.py -p PORT -b BAUD --before_
↪ default_reset --after hard_reset --chip auto write_flash --flash_mode dio --
↪ flash_size detect --flash_freq 40m 0x20000 ./at_customize.bin
```

- Replace PORT with your port name.
- Replace BAUD with the baud rate.

6.11 How to Enable ESP-AT Ethernet

6.11.1 Overview

This document is intended to help you enable ESP-AT Ethernet. After that, a simple test will show you how to confirm whether the enablement is successful.

Note: For more technical details about Ethernet (including basic concepts, MAC/PHY configuration, clock configuration, driver installation, and more), please refer to the [ESP-IDF Ethernet Programming Guide](#).

6.11.2 Step 1. Configure and Flash

1. `./build.py menuconfig`>Component config>AT>AT ethernet support to enable the Ethernet interface.
2. `./build.py menuconfig`>Component config>AT>Ethernet PHY to choose the PHY model. For more details see [PHY Configuration](#).
3. Recompile the `esp-at` project (see [Compile ESP-AT Project Locally](#)), download AT bin into flash.

PHY Configuration

Use `./build.py menuconfig` to set the PHY model. These configuration items will vary depending on the hardware configuration you are using.

The default configuration is correct for Espressif's Ethernet board with TP101 PHY. ESP32 AT supports up to four Ethernet PHY: LAN8720, IP101, DP83848 and RTL8201. TLK110 PHY is no longer supported because TI stopped production. If you want to use other PHY, follow the [ESP32-Ethernet-Kit V1.2 Getting Started Guide](#) to design.

6.11.3 Step 2. Connect the Board and Test

Now connect the board to the router via a ethernet cable, the board will automatically send a DHCP request and try to obtain an IP address. If the device obtains the IP address successfully, you can use the PC connected to the router to ping the board.

6.12 How to Add Support for a Module

The ESP-AT project supports multiple modules, and provides configuration for them in the [factory_param_data.csv](#) table and the files in the [module_config](#) folder. If you want to add support for an ESP32 module in your ESP-AT project, you need to modify those configuration files. The “ESP32 module” here means:

- Modules that the ESP-AT project has not supported yet, including those of supported platform and not supported platform. However, adding support for the latter requires extra huge work, thus not recommended and not explained in this document.
- Modules that the ESP-AT project supports, but you want to modify the default configuration.

This document will explain how to add new module support for a ESP32 chip/module that ESP-AT already supports in the ESP-AT project. The following example will enable the default [FileSystem AT commands](#) and add support for a new module.

- *Step 1: Configure the Factory Parameters for the New Module*
 - *Step 2: Configure OTA for the Newly Added Module*
 - *Step 3: Add New Module Configuration*

6.12.1 Step 1: Configure the Factory Parameters for the New Module

Open your local [factory_param_data.csv](#) , insert a new row at the end, and set the relevant parameters as needed. In this example, we set `platform` to `PLATFORM_ESP32` and `module_name` to `ESP32-USER-DEFINED`. The values for other parameters are shown in the table below (for parameter meanings, please refer to [Factory Parameter Configuration](#)).

- `platform`: `PLATFORM_ESP32`
- `module_name`: `ESP32-USER-DEFINED`
- `description`:
- `version`: 4
- `max_tx_power`: 78
- `uart_port`: 1
- `start_channel`: 1
- `channel_num`: 13
- `country_code`: CN
- `uart_baudrate`: -1
- `uart_tx_pin`: -1

- `uart_rx_pin`: -1
- `uart_cts_pin`: -1
- `uart_rts_pin`: -1

6.12.2 Step 2: Configure OTA for the Newly Added Module

Add customized module information in the `s_module_info` structure in `at/src/at_default_config.c`.

The `s_module_info` structure provides OTA upgrade verification token:

```
typedef struct {
    char* module_name;
    char* ota_token;
    char* ota_ssl_token;
} at_module_info_t;
```

If you do not want to use OTA features, member 2 `ota_token` and member 3 `ota_ssl_token` should be set to `NULL`. Member 1 `module_name` must correspond to the field `module_name` in the `factory_param_data.csv` file.

The modified `s_module_info` structure is as follows:

```
static const at_module_info_t s_module_info[] = {
#ifdef CONFIG_IDF_TARGET_ESP32
    ...
#endif

#ifdef CONFIG_IDF_TARGET_ESP32C3
    ...
#endif

#ifdef CONFIG_IDF_TARGET_ESP32C2
    ...
#endif

#ifdef CONFIG_IDF_TARGET_ESP32C5
    ...
#endif

#ifdef CONFIG_IDF_TARGET_ESP32C6
    ...
#endif

#ifdef CONFIG_IDF_TARGET_ESP32
    {"MY_MODULE", CONFIG_ESP_AT_OTA_TOKEN_MY_MODULE, CONFIG_ESP_AT_OTA_
    ↪SSL_TOKEN_MY_MODULE }, // MY_MODULE
#endif
};
```

Macro `CONFIG_ESP_AT_OTA_TOKEN_MY_MODULE` and macro `CONFIG_ESP_AT_OTA_SSL_TOKEN_MY_MODULE` are defined in the header file `at/private_include/at_ota_token.h`.

```
#if defined(CONFIG_IDF_TARGET_ESP32)
    ...
#define CONFIG_ESP_AT_OTA_TOKEN_MY_MODULE CONFIG_ESP_AT_OTA_TOKEN_DEFAULT

    ...
#define CONFIG_ESP_AT_OTA_SSL_TOKEN_MY_MODULE CONFIG_ESP_AT_OTA_SSL_TOKEN_
    ↪DEFAULT
```

6.12.3 Step 3: Add New Module Configuration

The ESP-AT project supports multiple platforms, each of which supports multiple module configurations and provides configuration files for each module configuration: [factory_param_data.csv](#) and [module_config](#). The table below lists the names of the platforms (i.e., chip series) supported by the ESP-AT project, the names of the module configurations, and the locations of the corresponding configuration files for each module configuration.

Platform	Module Configuration Name	Corresponding Default Configuration File
ESP32	WROOM-32	module_config/module_esp32_default/sdkconfig.defaults module_config/module_esp32_default/sdkconfig_silence.defaults
ESP32	PICO-D4	module_config/module_esp32_default/sdkconfig.defaults module_config/module_esp32_default/sdkconfig_silence.defaults
ESP32	SOLO-1	module_config/module_esp32_default/sdkconfig.defaults module_config/module_esp32_default/sdkconfig_silence.defaults
ESP32	MINI-1	module_config/module_esp32_default/sdkconfig.defaults module_config/module_esp32_default/sdkconfig_silence.defaults
ESP32	WROVER-32	module_config/module_wrover-32/sdkconfig.defaults module_config/module_wrover-32/sdkconfig_silence.defaults
ESP32	ESP32-D2WD	module_config/module_esp32-d2wd/sdkconfig.defaults module_config/module_esp32-d2wd/sdkconfig_silence.defaults
ESP32	ESP32-SDIO	module_config/module_esp32-sdio/sdkconfig.defaults module_config/module_esp32-sdio/sdkconfig_silence.defaults
ESP32-C2	ESP32C2-2MB	module_config/module_esp32c2-2mb/sdkconfig.defaults module_config/module_esp32c2-2mb/sdkconfig_silence.defaults
ESP32-C2	ESP32C2-2MB-BLE	module_config/module_esp32c2-2mb-ble/sdkconfig.defaults module_config/module_esp32c2-2mb-ble/sdkconfig_silence.defaults
ESP32-C2	ESP32C2-4MB	module_config/module_esp32c2_default/sdkconfig.defaults module_config/module_esp32c2_default/sdkconfig_silence.defaults
ESP32-C3	MINI-1	module_config/module_esp32c3_default/sdkconfig.defaults module_config/module_esp32c3_default/sdkconfig_silence.defaults
ESP32-C3	ESP32C3-SPI	module_config/module_esp32c3-spi/sdkconfig.defaults module_config/module_esp32c3-spi/sdkconfig_silence.defaults
ESP32-C3	ESP32C3_RAINM	module_config/module_esp32c3_rainmaker/sdkconfig.defaults module_config/module_esp32c3_rainmaker/sdkconfig_silence.defaults
ESP32-C5	ESP32C5-4MB	module_config/module_esp32c5_default/sdkconfig.defaults module_config/module_esp32c5_default/sdkconfig_silence.defaults
ESP32-C6	ESP32C6-4MB	module_config/module_esp32c6_default/sdkconfig.defaults module_config/module_esp32c6_default/sdkconfig_silence.defaults
ESP32-C61	ESP32C61-4MB	module_config/module_esp32c61_default/sdkconfig.defaults module_config/module_esp32c61_default/sdkconfig_silence.defaults

Note:

- When the `silence` mode in *python build.py install* is 0, the default sdkconfig corresponding to the module is `sdkconfig.defaults`.
- When the `silence` mode in *python build.py install* is 1, the default sdkconfig corresponding to the module is `sdkconfig_silence.defaults`.

Firstly, enter `module_config` folder, and create a new folder to store all the configuration files for your module. Note that the folder name should be in lower case. Then, add the configuration files in the new folder: `IDF_VERSION`, `patch`, `at_customize.csv`, `partitions_at.csv`, `sdkconfig.defaults`, and `sdkconfig_silence.defaults`.

In this example, we copy the `module_esp32_default` folder as well as the files within it and rename it as `module_esp32-user-defined`. The copied `IDF_VERSION`, `patch`, `at_customize.csv`, and `partitions_at.csv` do not need any modification in our case. We only need to modify the `sdkconfig.defaults` and `sdkconfig_silence.defaults`:

- Modify the two files to use the partition table in the `module_esp32-user-defined` folder as follows:

```
CONFIG_PARTITION_TABLE_CUSTOM_FILENAME="module_config/module_esp32-user-
→defined/partitions_at.csv"
CONFIG_PARTITION_TABLE_FILENAME="module_config/module_esp32-user-defined/
→partitions_at.csv"
CONFIG_AT_CUSTOMIZED_PARTITION_TABLE_FILE="module_config/module_esp32-user-
→defined/at_customize.csv"
```

- Add support for *FileSystem AT commands* configuration

```
CONFIG_AT_FS_COMMAND_SUPPORT=y
```

After completing the above steps, you can recompile the ESP-AT project to generate the module firmware. In this example, when we compile the AT firmware locally during *Step 3: Install Environment*, we can select `PLATFORM_ESP32` and `ESP32-USER-DEFINED` to generate the AT firmware.

6.13 ESP32 SDIO AT Guide

6.13.1 Introduction

ESP32 SDIO AT uses SDIO rather than UART as the communication interface between ESP32 (SDIO slave) and the host MCU.

SDIO can use either 1-bit or 4-bit data transfer mode.

- For 1-bit mode at least 4 pins are required: `CMD`, `CLK`, `DAT0`, `DAT1`; where `DAT1` runs as the interrupt pin.
- For 4-bit mode, two more pins (`DAT2` and `DAT3`) are needed.

The SDIO slave pins are as below:

- `CLK` is `GPIO14`
- `CMD` is `GPIO15`
- `DAT0` is `GPIO2`
- `DAT1` is `GPIO4`
- `DAT2` is `GPIO12` (for 4-bit mode only)
- `DAT3` is `GPIO13` (for 4-bit mode only)

6.13.2 Implement SDIO AT

Before testing SDIO AT communication, please configure ESP32 hardware according to [SD Pull-up Requirements](#), otherwise, exceptions may occur during SDIO communication.

SDIO Slave

By default, the ESP-AT project uses UART as the communication interface. When ESP32 serves as the SDIO slave, please go to `./build.py menuconfig > Component config > AT > communicate method for AT command > AT via SDIO` to switch to the SDIO interface. Then, recompile the ESP-AT project, flash and run the new bin.

SDIO MCU

ESP-AT provides the [at_sdio_host](#) example for ESP32 or STM32 serving as the SDIO host. For ESP32 serving as the host, you can use the esp-idf of the AT project to compile and flash; for STM32, as the provided solution is based on STM32F103ZET, please use Keil5 to compile and flash as the application.

For other devices working as SDIO hosts, you can develop based on the above example.

6.13.3 SDIO Data Transmission

SDIO Host

1. Initialize the SDIO host
 - The SDIO host initialization is mainly the initialization of the SDIO protocol, including setting 1-bit or 4-bit transfer mode, SDIO frequency, and SD mode.
2. Negotiate SDIO communication
 - Negotiate SDIO parameters with the slave according to the SDIO specification. Please note that if the host and slave reboot at the same time, the host must wait until the slave startup is done to start the negotiation. Usually, the host needs to add a delay at boot time to achieve it.
3. Receive data
 - When the SDIO host detects an interrupt from DAT1 and determines that the slave sends new packets, the host will read those data by CMD53.
4. Send data
 - In SDIO AT demo, the data inputs from a serial port. When the SDIO host gets the data, it will transfer it to the slave through SDIO by CMD53.
 - Please note that if the sending time out, there may be something wrong with the slave, please re-initiate both the host and slave.
 - After [AT+RST](#) or [AT+RESTORE](#) is called, both the host and slave should be initialized again.

SDIO Slave

When the SDIO slave receives data from the SDIO host, it will inform the AT core and give the data to the AT core to handle. After the AT core finishes the work, the slave will send data to the host as feedback.

1. Initialize the SDIO slave
 - Call `sdio_slave_initialize` to initialize the SDIO slave driver.
 - Call `sdio_slave_recv_register_buf` to register receiving buffer. To make it faster to receive data, you can register multiple buffers.
 - Call `sdio_slave_recv_load_buf` to load the receiving buffer to receive data.
 - Call `sdio_slave_set_host_intena` to set host interrupts, of which the `SDIO_SLAVE_HOSTINT_SEND_NEW_PACKET` is used to notify that a new packet is sent from the host.
 - Call `sdio_slave_start` to start SDIO hardware transmission.
2. Send data

- Since the SDIO slave data transmission is via DMA, you need to copy the data from AT core to the memory which DMA can read firstly. Call `sdio_slave_transmit` to send the data.
3. Receive data
 - To speed up the data transmission, after receiving data by `sdio_slave_recv`, a circular linked list is used to transmit the received data to the AT core.

6.14 How to Implement OTA Upgrade

This document introduces how to implement OTA upgrade on ESP32 series of modules. Currently, ESP-AT provides the following three OTA commands targeting at different scenarios. You could choose one command according to your needs.

1. [*AT+USEROTA*](#)
2. [*AT+CIUPDATE*](#)
3. [*AT+WEBSERVER*](#)

The structure of this document is as follows:

- [*Comparison Among OTA Commands and Their Application Scenarios*](#)
- [*Perform OTA Upgrade with ESP-AT OTA Commands*](#)

6.14.1 Comparison Among OTA Commands and Their Application Scenarios

AT+USEROTA

This command implements OTA upgrade through a URL. You can upgrade to the firmware placed on the HTTP server. Currently, the command only supports the upgrade of app partitions. For more information on this command, please refer to [*AT+USEROTA*](#) for more details.

Since this command is a user-defined command, you can modify the implementation of this command by modifying the `at/src/at_user_cmd.c` source code.

The application scenarios of this command are as follows:

1. **You have your own HTTP server.**
2. **You need to specify the URL.**

Important:

- If the firmware you upgrade is not officially released by Espressif, you may no longer be able to use `AT+CIUPDATE` command to upgrade after the upgrade is complete, unless you create your own device according to [*OTA Upgrade with AT+CIUPDATE*](#).
-

AT+CIUPDATE

This command uses `iot.espressif.cn` as the default HTTP server. It can upgrade both the app partition and the user-defined partitions that are defined in the `at_customize.csv`. If you are using the version released by Espressif, it will only upgrade to the version released by Espressif. For more information on this command, please refer to [*AT+CIUPDATE*](#) for more details.

To upgrade the customized bin file with this command, please select one of the following ways.

1. **replace `iot.espressif.cn` with the your own HTTP server and implement the interactive process.** For how to implement your own `AT+CIUPDATE` command, please refer to [`at/src/at\_ota\_cmd.c`](#).

2. **create a devices on iot.espressif.cn and upload customized AT firmware on it.** (The premise is that the firmware running in the module already corresponds to the device you created on the Espressif server.) For more information, please refer to [OTA Upgrade with AT+CIUPDATE](#).

The application scenarios of this command are as follows:

1. **You only use the firmware released by Espressif, and only want to upgrade to the firmware released by Espressif.**
2. **You want to upgrade the customized bin file, but do not have an HTTP server.**
3. **You have your own HTTP server. In addition to the app partition, you also want to upgrade the user-defined partitions in `at_customize.csv`.**

AT+WEBSERVER

This command upgrades AT firmware with a browser or WeChat applet. Currently, this command only supports the upgrade of app partitions. Before starting the upgrade, please enable the web server command and copy the AT firmware to the computer or mobile phone in advance. For more information, you can refer to [AT+WEBSERVER](#) and [Web Server AT Example](#).

To implement your own HTML page, please refer to the example of [fs_image/index.html](#). To implement your own AT+WEBSERVER command, please refer to the example of [at/src/at_web_server_cmd.c](#).

The application scenarios of this command are as follows:

1. **You need a more convenient and faster OTA upgrade that does not rely on network status.**

Important:

- If the firmware you upgrade is not officially released by Espressif, you may no longer be able to use AT+CIUPDATE command to upgrade after the upgrade is complete, unless you create your own device according to [OTA Upgrade with AT+CIUPDATE](#).
-

6.14.2 Perform OTA Upgrade with ESP-AT OTA Commands

OTA Upgrade with AT+USEROTA

Please refer to [AT+USEROTA](#) for more details.

OTA Upgrade with AT+CIUPDATE

To upgrade the customized bin file with [AT+CIUPDATE](#) command, the first thing to do is to upload the bin file to the iot.espressif.cn and obtain the **token** value. The following steps describe how to create a device on iot.espressif.cn and upload the bin file to it.

1. Open the website <http://iot.espressif.cn> or <https://iot.espressif.cn>.
2. Click “Join” in the upper right corner of the webpage, and enter your name, email address, and password.

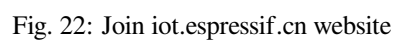
Note:

- The Join function is currently not open to new users. If you want to use it, please contact [Espressif](#).
-

3. Click on “Device” in the upper left corner of the webpage, and click on “Create” to create a device.
4. A key is generated when the device is successfully created, as the figure below shows.
5. Use the key to compile your own OTA bin file. The process of configuring the AT OTA token key is as follows:

Note:

- If using SSL OTA, the option “The SSL token for AT OTA” also needs to be configured.



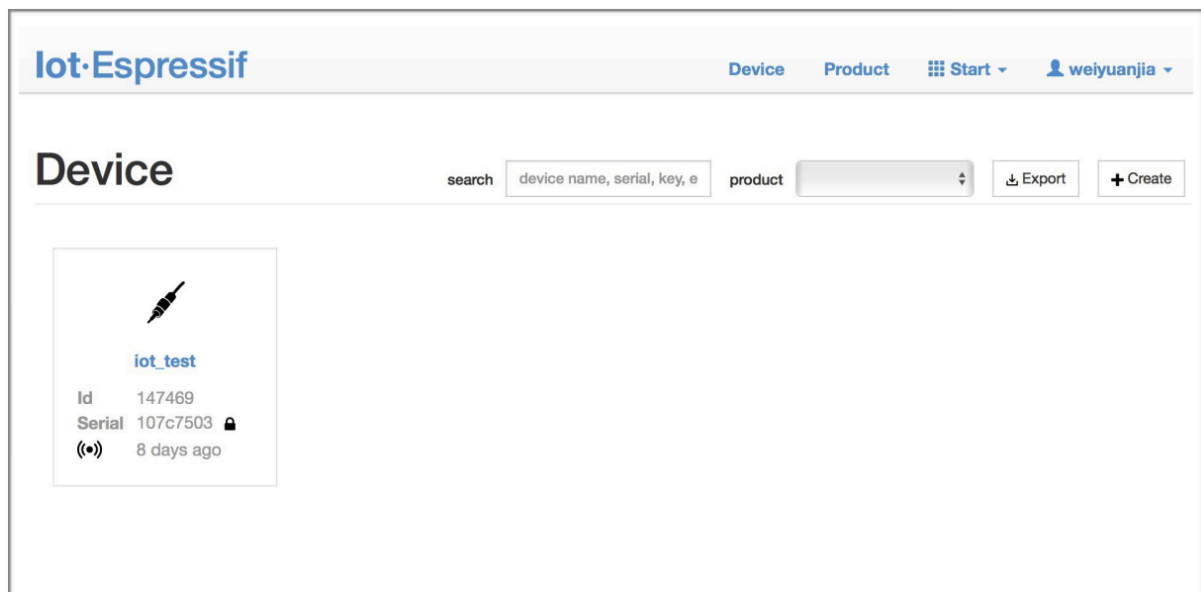


Fig. 23: Click on “Device”

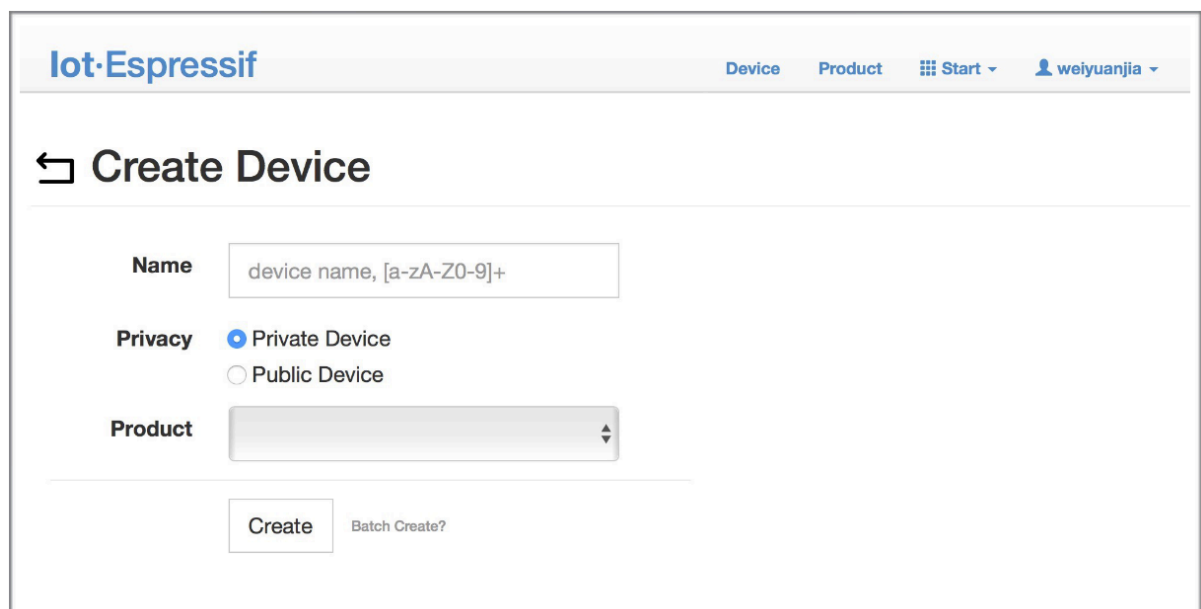


Fig. 24: Click on “Create”

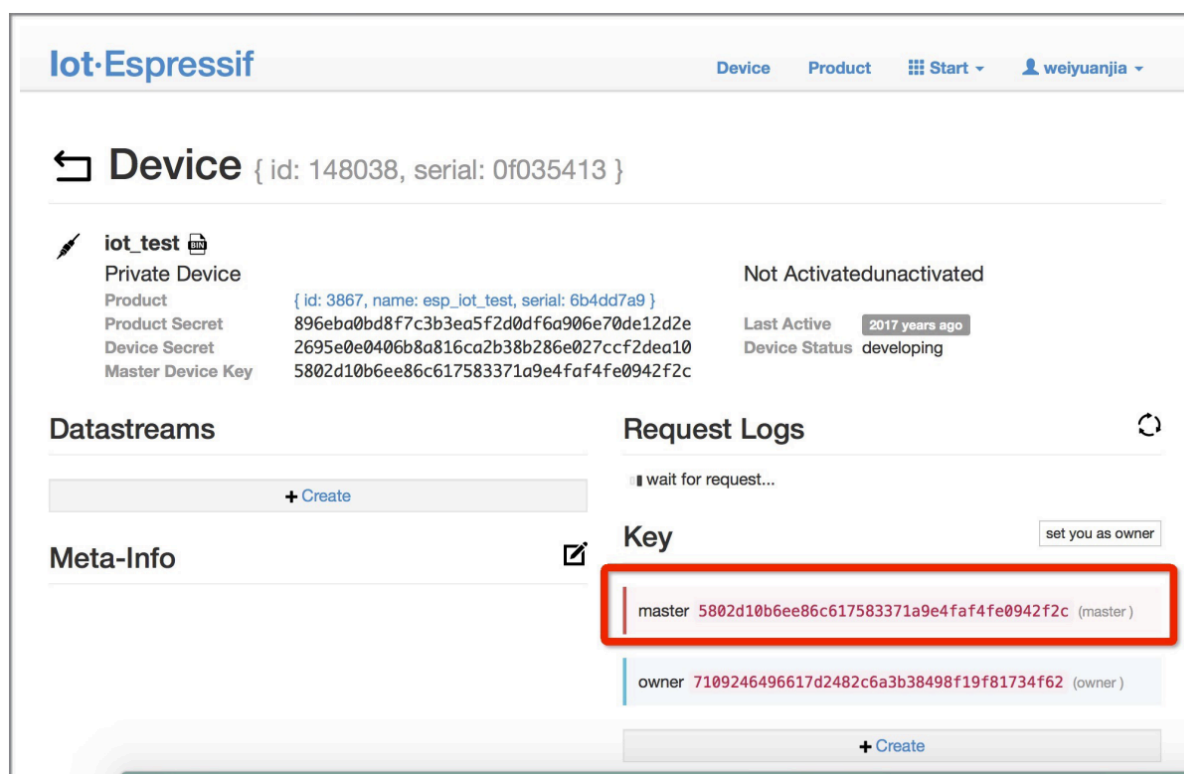


Fig. 25: A key has been generated

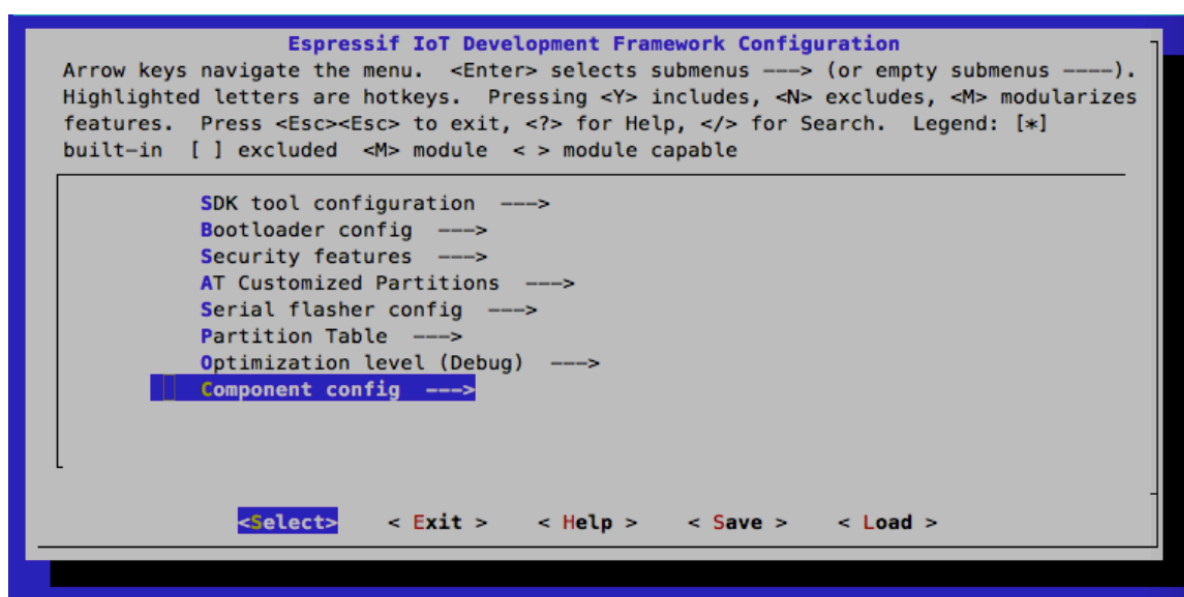


Fig. 26: Configuring the AT OTA token key - Step 1

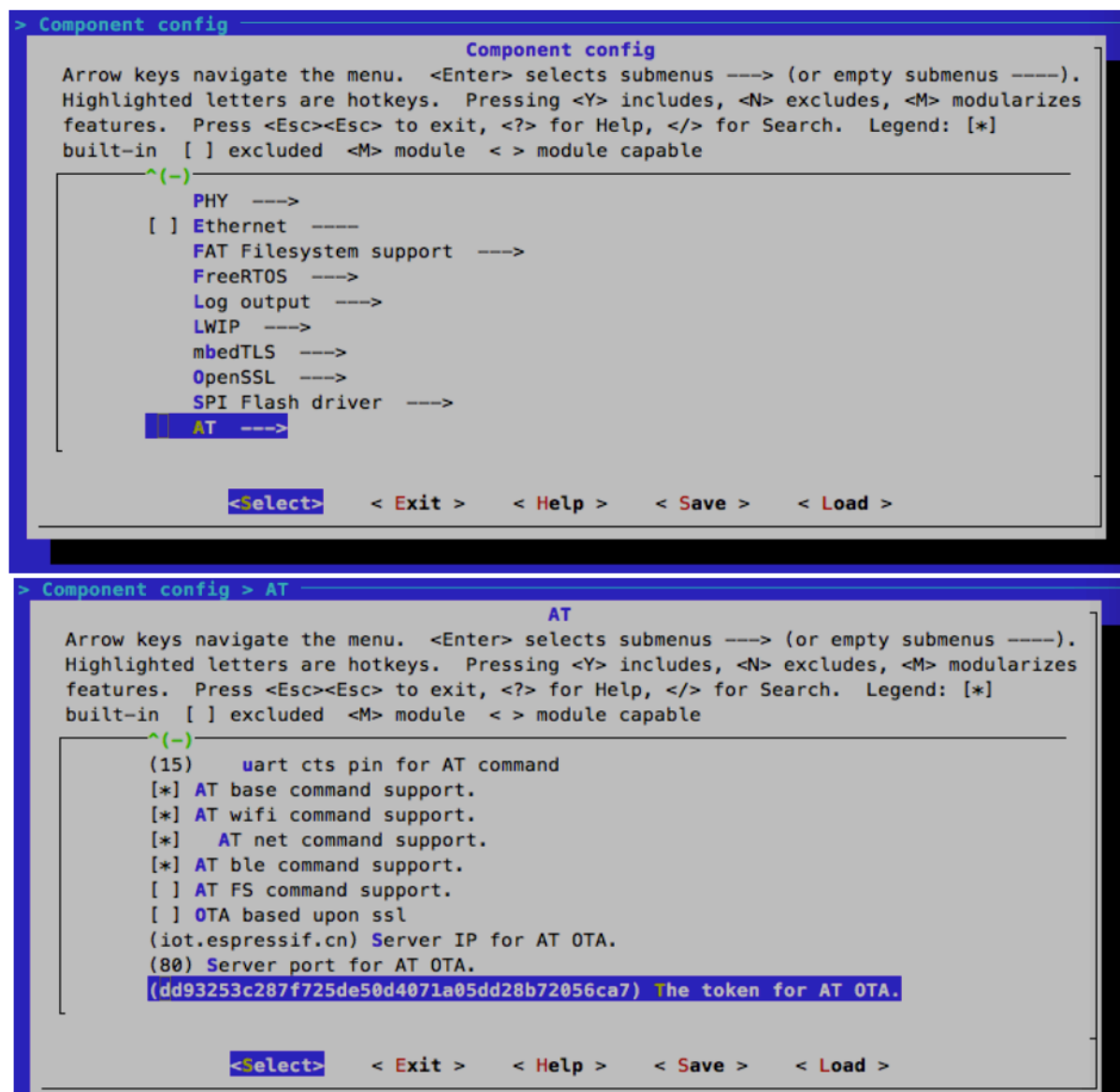


Fig. 27: Configuring the AT OTA token key - Step 2 and 3

6. Click on “Product” to enter the webpage, as shown below. Click on the device created. Enter version and corename under “ROM Deploy”. Rename the bin file in Step 5 as “ota.bin” and save the configuration.

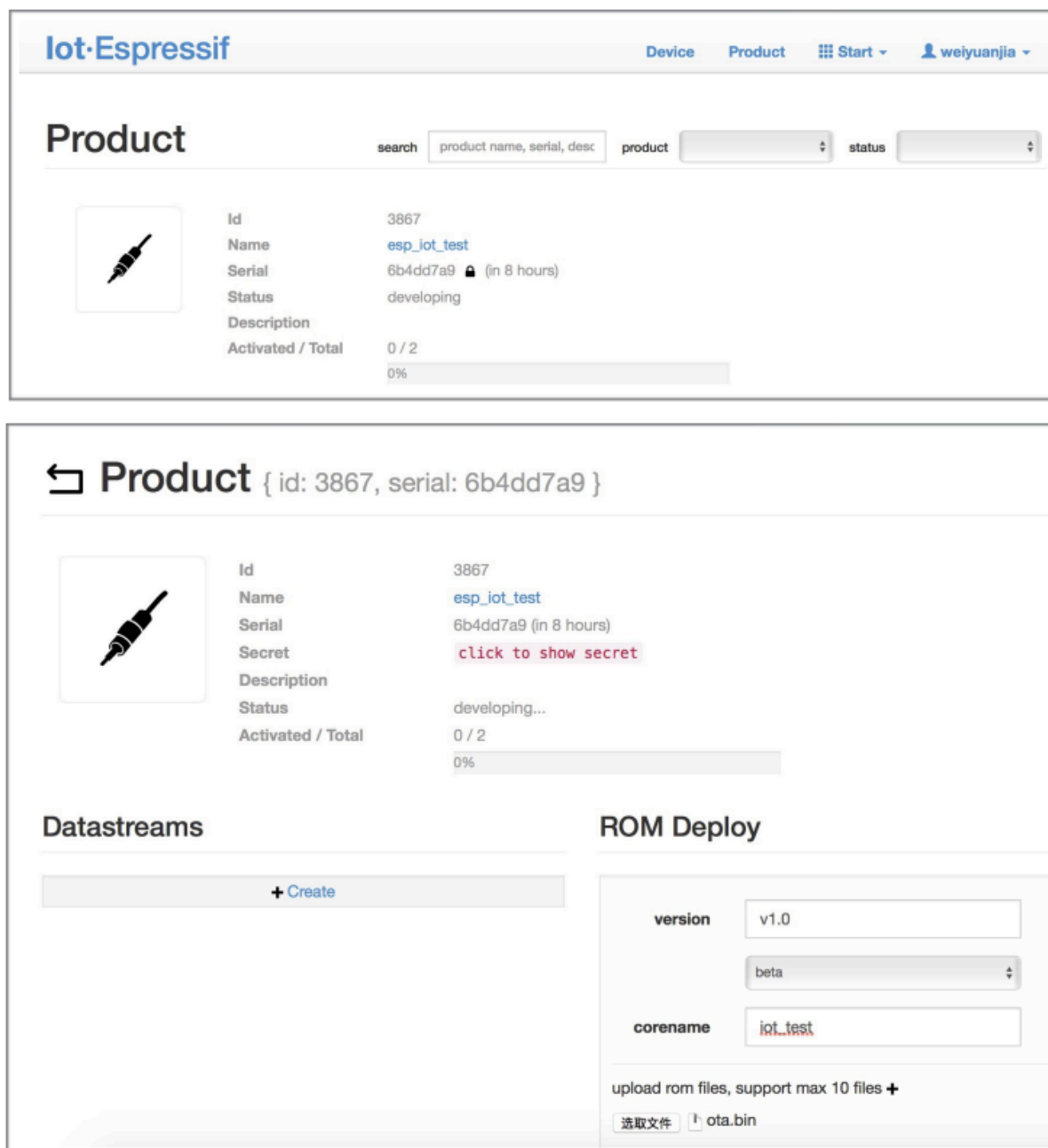


Fig. 28: Enter version and corename

Note:

- If you want to upgrade the user-defined partition defined in the `at_customize.csv`, just replace `ota.bin` with the bin of the user-defined partition.
- For the field `corename`, the purpose of this field is only to help you distinguish bin files.

7. Click on the `ota.bin` to save it as the current version.
 8. Run the command `AT+USEROTA` on the ESP32. If the network is connected, OTA upgrade will begin.

Important:

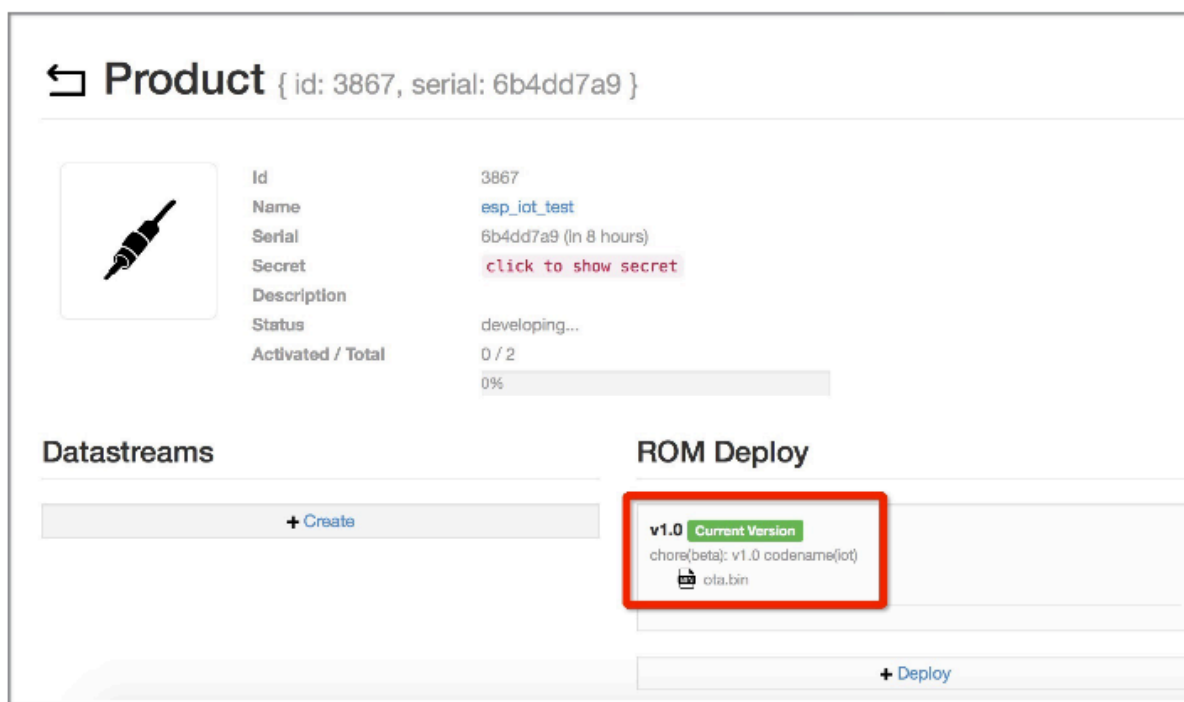


Fig. 29: Save the current version of ota.bin

- When setting the name of bin files uploaded to the `iot.espressif.cn`, please follow the rules below:
 - If you upgrade app partitions, set the bin file name to `ota.bin`.
 - If you upgrade a user-defined partition, set the bin file name as the Name field in the `at_customize.csv`. For example, if you upgrade `factory_Param` partition, please set it to `factory_param.bin`.
- ESP-AT stores the new firmware in a spare OTA partition. In this way, even if OTA fails due to unexpected reasons, the original ESP-AT firmware can still run. But for user-defined partitions, because ESP-AT has no backup measures, please upgrade it carefully.
- **If you intend to upgrade only customized bin file at the beginning, please set the OTA token to your own token value when the initial version is released.**

OTA Upgrade with AT+WEBSERVER

Please refer to [AT+WEBSERVER](#) and [Web Server AT Example](#) for more details.

6.15 How to Update the ESP-IDF Version

ESP-AT firmware is based on the Espressif IoT Development Framework (ESP-IDF), and each version of ESP-AT firmware corresponds to a specific ESP-IDF version. It is strongly recommended to use the default ESP-IDF version of the ESP-AT project, and **NOT** recommended to update it, because the inconsistency between the underlying ESP-IDF versions of `libesp32_at_core.a` and the ESP-AT project may cause incorrect operation of the firmware.

However, in some special cases, a minor updated ESP-IDF version may be also able to work with the ESP-AT project. In case you should update it, this document can be a reference.

The ESP-IDF version that a specific ESP-AT firmware corresponds to is recorded in the `IDF_VERSION` files, which are distributed in different module folders under the [module_config](#) folder. The file describes

the branch, commit ID, and repository URL of the ESP-IDF that a module firmware is based on. For example, the `IDF_VERSION` of the `WROOM-32` module of `PLATFORM_ESP32` platform is located at [module_config/module_esp32_default/IDF_VERSION](#).

If you want to update the ESP-IDF version for your ESP-AT firmware, please follow the steps:

- Find the `IDF_VERSION` file for your module.
- Update the branch, commit ID, repository as needed.
- Delete the original `esp-idf` under the `esp-at` root directory, so that the ESP-IDF of the version specified in `IDF_VERSION` will be cloned first in the next compilation.
- Recompile the ESP-AT project.

Note that when the ESP-AT version and the ESP-IDF version do not match, the following error will be reported when compiling:

Please wait **for** the update to complete, which will take some time

6.16 ESP-AT Firmware Differences

The documentation has been moved to [AT Released Firmware Differences](#).

6.17 How to Download the Latest Temporary Version of AT Firmware from GitHub

As ESP-AT enables CI (Continuous Integration) on GitHub, temporary versions of ESP-AT firmware is generated every time when the code is pushed to GitHub.

Attention: The latest temporary version of AT firmware you download is for testing purposes only. For mass production, please make sure to conduct comprehensive function verification and testing based on the actual application scenarios of your product. In addition, the current firmware does not support the official OTA function ([AT+CIUPDATE](#)).

Please save the firmware and download link, for possible issue debugging in the future.

The following steps guide you on how to download the latest temporary version of AT firmware from GitHub.

1. Sign in your GitHub account
Before you start, **please sign in your GitHub account**, as you need login permission to download firmware.
2. Open the website <https://github.com/espressif/esp-at>
3. Click “Actions” to enter the “Actions” page
4. Click the branch and choose the specified branch
Default branch is master. If you want to download the temporary firmware of the specified branch, click “Branch” to enter the workflow page of the specified branch.
5. Click the latest workflow and enter the latest workflow page
6. Scroll the page to the end, select the corresponding module on the `Artifacts` page
7. Click to download the temporary version of AT firmware for modules

Note: If you do not find the firmware of the corresponding module on the `Artifacts` page, you can refer to [ESP-AT Firmware Differences](#) to select a similar firmware to download.

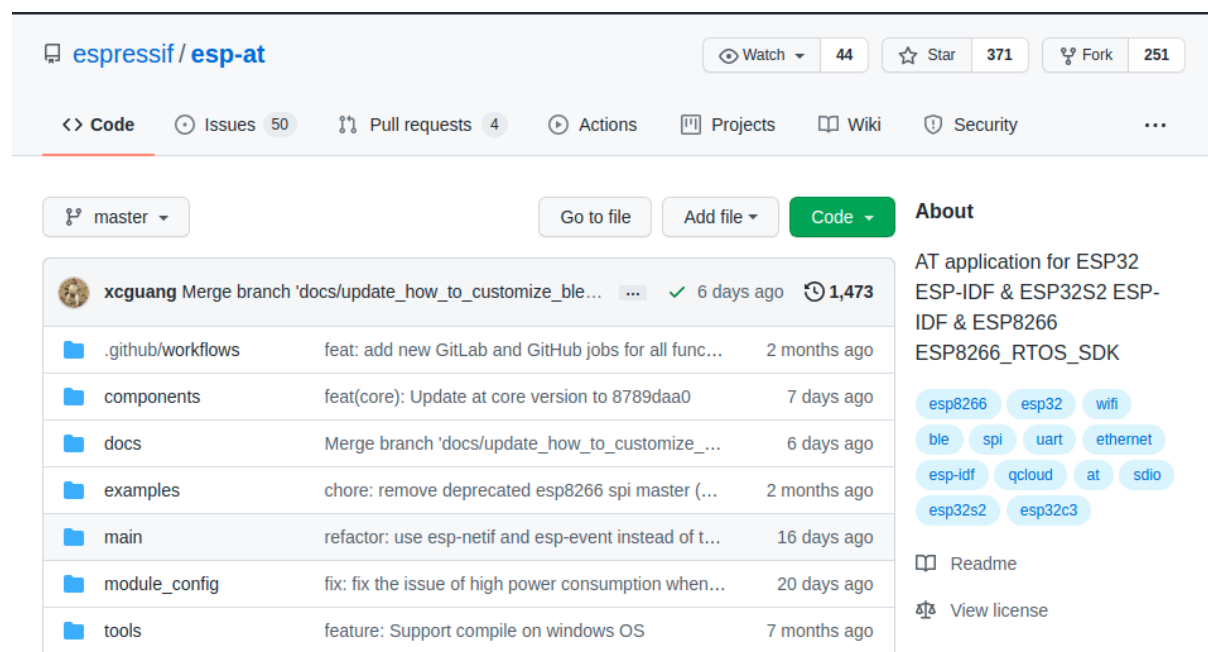


Fig. 30: ESP-AT GitHub Home Page

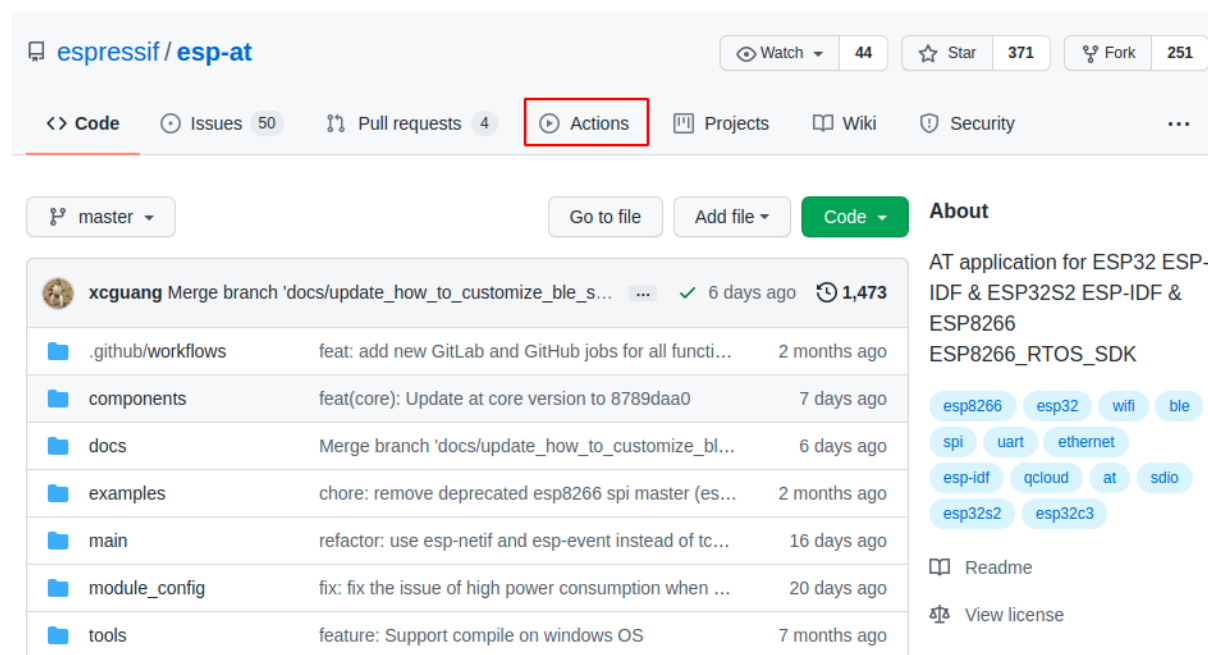


Fig. 31: Click Actions

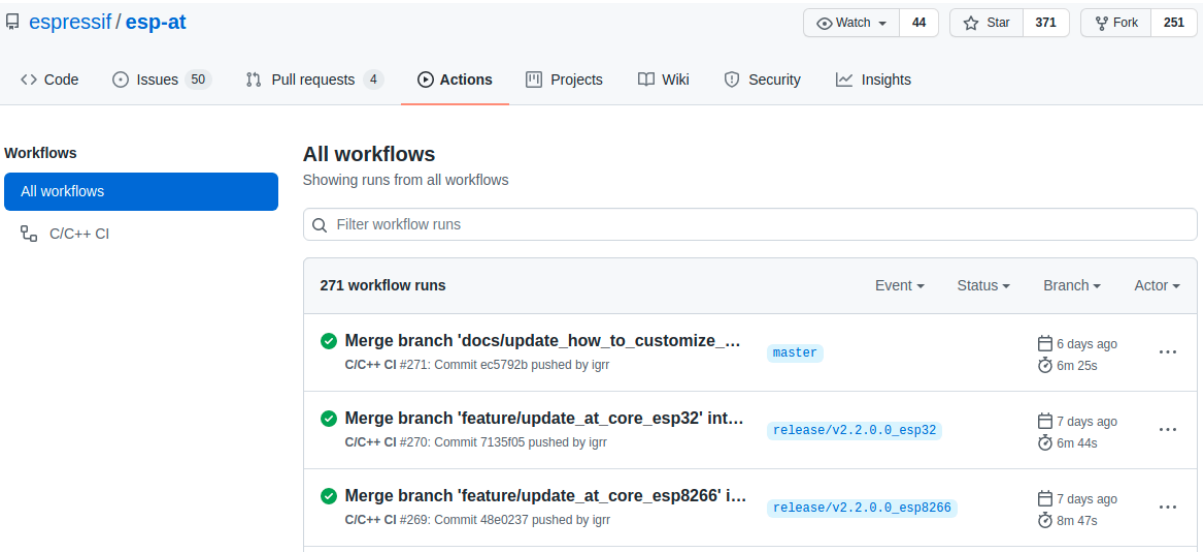


Fig. 32: Actions Page

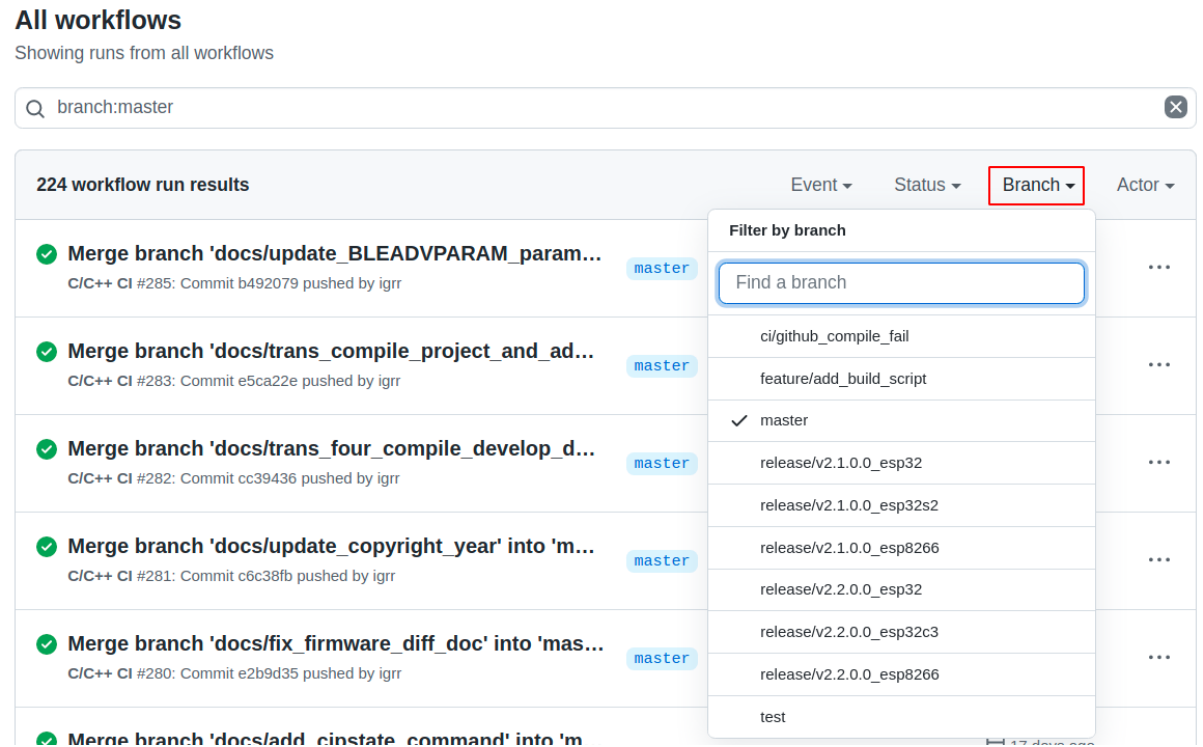


Fig. 33: Click the Branch

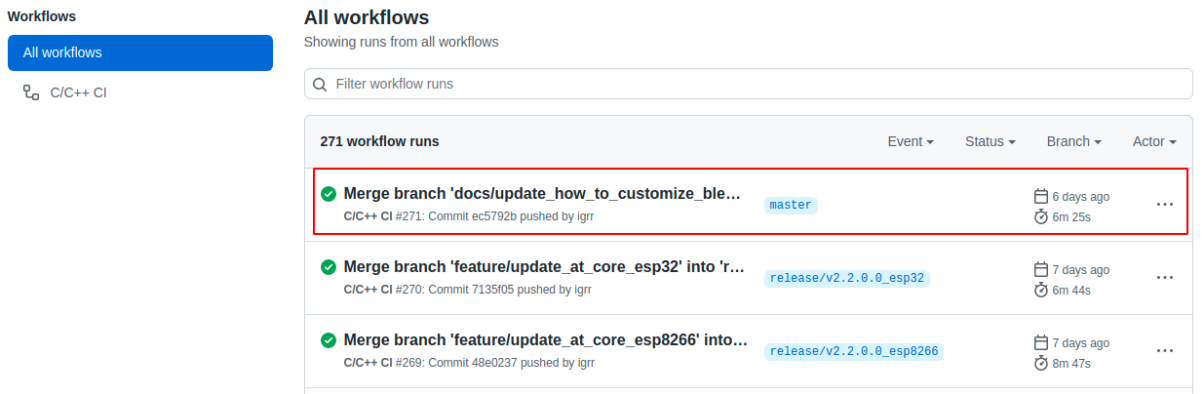


Fig. 34: Click the Latest Workflow

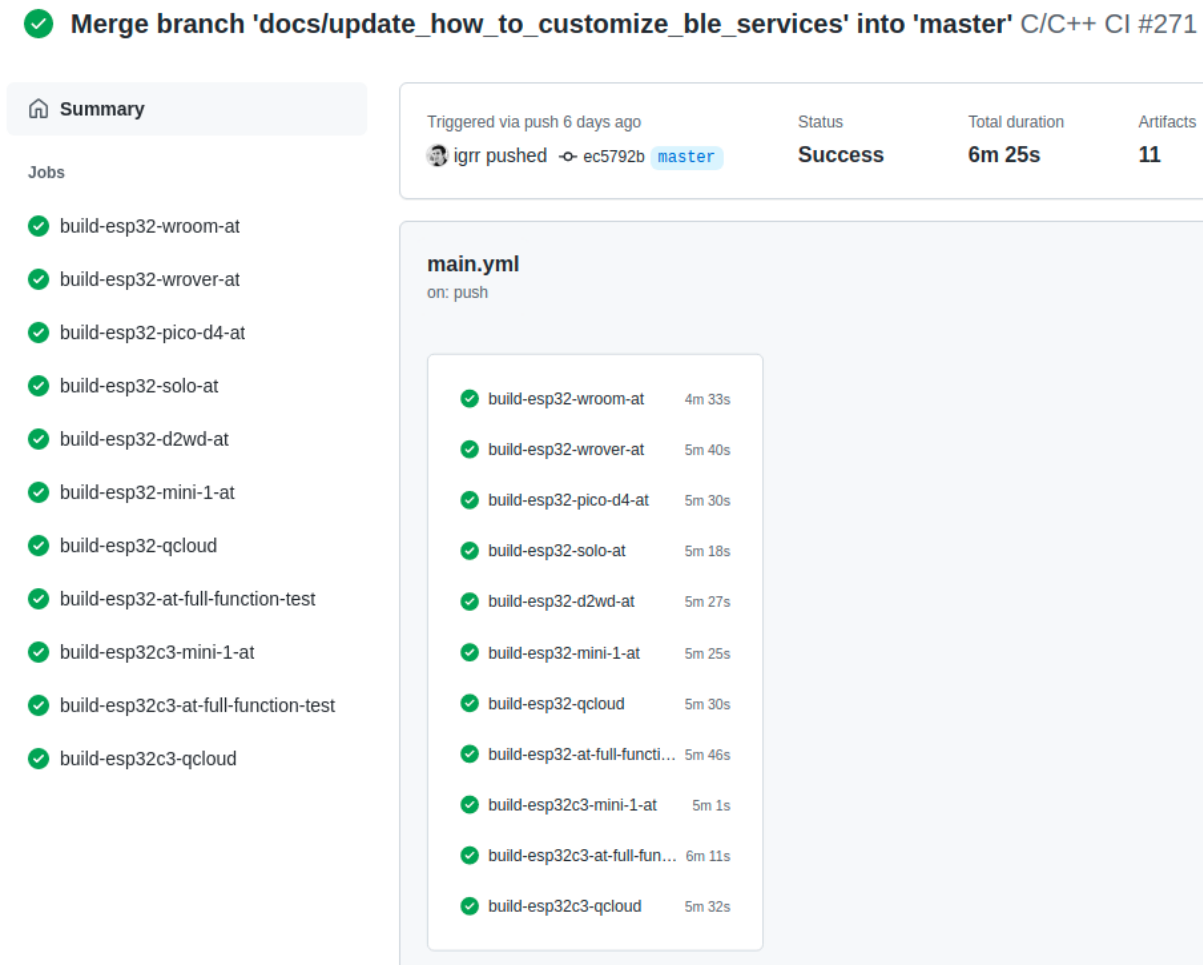


Fig. 35: Latest Workflow Page

Artifacts		
Produced during runtime		
Name		Size
 esp32-at-full-function-test		35.8 MB
 esp32-d2wd-at		26.2 MB
 esp32-mini-1-at		28.3 MB
 esp32-pico-d4-at		28.3 MB
 esp32-qcloud		29.2 MB
 esp32-solo-1-at		28.3 MB
 esp32-wroom-at		28.3 MB
 esp32-wrover-at		30.2 MB
 esp32c3-at-full-function-test		29.7 MB
 esp32c3-mini-1-at		27.5 MB
 esp32c3-qcloud		28.5 MB

Fig. 36: Artifacts Page

6.18 at.py Tool

The `at.py` tool is used to modify various parameter configurations in the packaged 2 MB/4 MB AT production firmware (namely, `factory_XXX.bin` in the `build/factory` directory). These configurations include Wi-Fi configurations, certificate and key configurations, UART configurations, GATTS configurations, and more. When the default firmware does not meet your requirements, you can use the `at.py` tool to modify these parameter configurations in the firmware.

6.18.1 Detailed Steps

Please follow the detailed steps below to download the `at.py` tool, modify the configurations in the firmware, and flash the firmware. **It is recommended to use a Linux system for development.**

- *Step 1: Install Python*
- *Step 2: Download at.py*
- *Step 3: Use at.py*
- *Step 4: Examples: Modify Firmware Configurations with at.py*
- *Step 5: Flash onto the Device*

6.18.2 Step 1: Install Python

If you have already installed Python on your local machine, you can skip this step. Otherwise, please download and install Python according to the official [Python documentation](#). **It is recommended to use Python 3.10.0 or later versions.**

After the installation is complete, you can open a command line and enter `python --version` to check the Python version.

6.18.3 Step 2: Download at.py

Visit the [at.py](#) webpage, click the “Download raw file” button to download the `at.py` file to your local machine.

6.18.4 Step 3: Use at.py

Currently, `at.py` supports modifying parameter configurations in the firmware. To view the supported usage and instructions, enter `python at.py modify_bin --help` in the command line for more details.

6.18.5 Step 4: Examples: Modify Firmware Configurations with at.py

- *Modify Wi-Fi Configuration*
- *Modify Certificate and Key Configuration*
- *Modify UART Configuration*
- *Modify GATTS Configuration*

Note:

- You can modify the parameter configurations of multiple functions at one go. For example, you can modify both Wi-Fi configuration and certificate and key configuration at the same time.
- The script does not check the validity of the modified parameter configurations. Please ensure that the input configurations are valid.

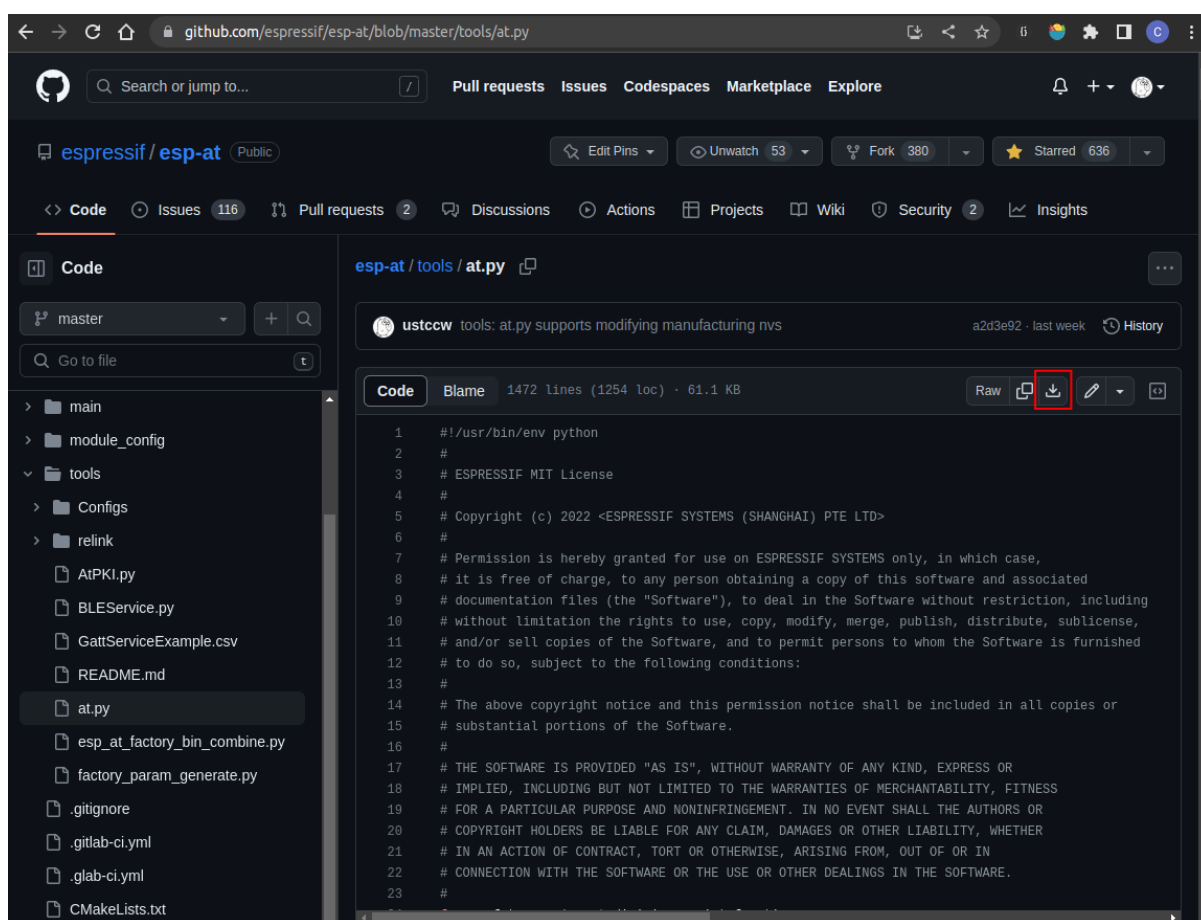


Fig. 37: Download at.py File

- The modified parameter configurations will have corresponding records in the `mfg_nvs.csv` file under the current `mfg_nvs` directory.

Modify Wi-Fi Configuration

The currently modifiable Wi-Fi parameter configurations are shown in the following table:

Parameter	Function	Description
<code>--tx_power</code>	Maximum transmission power of Wi-Fi	See ESP32 Maximum transmission power for more details.
<code>--country_code</code>	Wi-Fi country code	See the <code>cc</code> field of Wi-Fi Country Code
<code>--start_channel</code>	Wi-Fi starting channel	See the <code>schan</code> field of Wi-Fi Country Code
<code>--channel_number</code>	Total number of Wi-Fi channels	See the <code>nchan</code> field of Wi-Fi Country Code

For example, you can use the following command to modify the maximum transmission power of Wi-Fi to 18 dBm, set the country code to US, starting channel to 1, and total channel number to 11:

```
python at.py modify_bin --tx_power 72 --country_code "US" --start_channel 1 --
↪channel_number 11 --input factory_XXX.bin
```

- **--tx_power 72:** The unit is 0.25 dBm, and 72 represents 18 dBm.
- **--input factory_XXX.bin:** The input firmware file.

Modify Certificate and Key Configuration

The currently modifiable certificate and key configurations are shown in the following table:

Parameter	Function	Original File
<code>--server_ca</code>	CA certificate for TLS server	server_ca.crt
<code>--server_cert</code>	Certificate for TLS server	server_cert.crt
<code>--server_key</code>	Key for TLS server	server.key
<code>--client_ca0</code>	CA certificate for client 0	client_ca_00.crt
<code>--client_cert0</code>	Certificate for client 0	client_cert_00.crt
<code>--client_key0</code>	Key for client 0	client_key_00.key
<code>--client_ca1</code>	CA certificate for client 1	client_ca_01.crt
<code>--client_cert1</code>	Certificate for client 1	client_cert_01.crt
<code>--client_key1</code>	Key for client 1	client_key_01.key
<code>--mqtt_ca</code>	CA certificate for MQTT client	mqtt_ca.crt
<code>--mqtt_cert</code>	Certificate for MQTT client	mqtt_client.crt
<code>--mqtt_key</code>	Key for MQTT client	mqtt_client.key
<code>--wpa2_ca</code>	CA certificate for WPA2-Enterprise client	wpa2_ca.pem
<code>--wpa2_cert</code>	Certificate for WPA2-Enterprise client	wpa2_client.crt
<code>--wpa2_key</code>	Key for WPA2-Enterprise client	wpa2_client.key

For example, you can use the following command to modify the CA certificate, certificate, and key for the MQTT client:

```
python at.py modify_bin --mqtt_ca mqtt/mqtt_ca.crt --mqtt_cert mqtt/mqtt.crt --
↪mqtt_key mqtt/mqtt.key --input factory_XXX.bin
```

- **--input factory_XXX.bin:** The input firmware file.

Modify UART Configuration

The modifiable UART configuration includes only the UART configuration for the *AT command port*. The configurable parameter configurations are shown in the following table:

Parameter	Function	Description
--uart_num	UART number for the AT command port	Only needs to be modified if the AT command port is also used as the AT log port. Ensure that the <code>tx_pin</code> and <code>rx_pin</code> below have the same pins as the <i>AT log port</i> . If the <i>AT log port</i> is only configured with the <code>rx_pin</code> , you need to configure the following <code>tx_pin</code> to be consistent with the <code>tx_pin</code> of the UART of the Download Firmware Port (please refer to <i>Hardware Connection</i>).
--baud	Baud rate of the AT command port	Original value: 115200
--tx_pin	TX pin of the AT command port	Please ensure that the set pin is not used by other functions.
--rx_pin	RX pin of the AT command port	Please ensure that the set pin is not used by other functions.
--cts_pin	CTS pin of the AT command port	Please ensure that the set pin is not used by other functions. If flow control is not used, set this parameter to -1.
--rts_pin	RTS pin of the AT command port	Please ensure that the set pin is not used by other functions. If flow control is not used, set this parameter to -1.

For example, you can use the following command to modify the baud rate to 921600, set the TX pin to GPIO17, the RX pin to GPIO16, and disable flow control for the AT command port:

```
python at.py modify_bin --baud 921600 --tx_pin 17 --rx_pin 16 --cts_pin -1 --rts_
pin -1 --input factory_XXX.bin
```

- **--input factory_XXX.bin:** The input firmware file.

Modify GATTS Configuration

Before making modifications, please read the *How to Customize Bluetooth® LE Services* document to understand the meaning of each field in the GATTS configuration file `gatts_data.csv`.

The currently modifiable GATTS configurations are shown in the following table:

Parameter	Function
--gatts_cfg0	Update the row with index 0 in the <code>gatts_data.csv</code> file
--gatts_cfg1	Update the row with index 1 in the <code>gatts_data.csv</code> file
--gatts_cfg2	Update the row with index 2 in the <code>gatts_data.csv</code> file
--gatts_cfg3	Update the row with index 3 in the <code>gatts_data.csv</code> file
--gatts_cfg4	Update the row with index 4 in the <code>gatts_data.csv</code> file
--gatts_cfg5	Update the row with index 5 in the <code>gatts_data.csv</code> file
--gatts_cfg6	Update the row with index 6 in the <code>gatts_data.csv</code> file
--gatts_cfg7	Update the row with index 7 in the <code>gatts_data.csv</code> file
--gatts_cfg8	Update the row with index 8 in the <code>gatts_data.csv</code> file
--gatts_cfg9	Update the row with index 9 in the <code>gatts_data.csv</code> file
--gatts_cfg10	Update the row with index 10 in the <code>gatts_data.csv</code> file
--gatts_cfg11	Update the row with index 11 in the <code>gatts_data.csv</code> file

continues on next page

Table 3 – continued from previous page

Parameter	Function
--gatts_cfg12	Update the row with index 12 in the gatts_data.csv file
--gatts_cfg13	Update the row with index 13 in the gatts_data.csv file
--gatts_cfg14	Update the row with index 14 in the gatts_data.csv file
--gatts_cfg15	Update the row with index 15 in the gatts_data.csv file
--gatts_cfg16	Update the row with index 16 in the gatts_data.csv file
--gatts_cfg17	Update the row with index 17 in the gatts_data.csv file
--gatts_cfg18	Update the row with index 18 in the gatts_data.csv file
--gatts_cfg19	Update the row with index 19 in the gatts_data.csv file
--gatts_cfg20	Update the row with index 20 in the gatts_data.csv file
--gatts_cfg21	Update the row with index 21 in the gatts_data.csv file
--gatts_cfg22	Update the row with index 22 in the gatts_data.csv file
--gatts_cfg23	Update the row with index 23 in the gatts_data.csv file
--gatts_cfg24	Update the row with index 24 in the gatts_data.csv file
--gatts_cfg25	Update the row with index 25 in the gatts_data.csv file
--gatts_cfg26	Update the row with index 26 in the gatts_data.csv file
--gatts_cfg27	Update the row with index 27 in the gatts_data.csv file
--gatts_cfg28	Update the row with index 28 in the gatts_data.csv file
--gatts_cfg29	Update the row with index 29 in the gatts_data.csv file
--gatts_cfg30	Update the row with index 30 in the gatts_data.csv file

For example, you can use the following command to modify the “perm” permission of the row with index 0:

```
python at.py modify_bin --gatts_cfg0 "0,16,0x2800,0x011,2,2,A002" --input factory_
↪XXX.bin
```

- **--input factory_XXX.bin:** The input firmware file.

Attention: The AT firmware modified by `at.py` needs to be tested and verified for functionality based on your own product.

Please save the firmware before and after modification, and the download link, for possible issue debugging in the future.

Step 5: Flash onto the Device Please follow the [Flash firmware](#) to complete it.

6.19 AT API Reference

6.19.1 Header File

- [components/at/include/esp_at_core.h](#)

6.19.2 Functions

void **esp_at_module_init** (const uint8_t *custom_version)

This function should be called only once.

Parameters

custom_version –version information by custom

esp_at_para_parse_result_type **esp_at_get_para_as_digit** (int32_t para_index, int32_t *value)

Parse digit parameter from command string.

Parameters

- **para_index** –the index of parameter
- **value** –the value parsed

Returns

- ESP_AT_PARA_PARSE_RESULT_OK : succeed
- ESP_AT_PARA_PARSE_RESULT_FAIL : fail
- ESP_AT_PARA_PARSE_RESULT_OMITTED : this parameter is OMITTED

esp_at_para_parse_result_type **esp_at_get_para_as_float** (int32_t para_index, float *value)

Parse float parameter from command string.

Parameters

- **para_index** –the index of parameter
- **value** –the value parsed

Returns

- ESP_AT_PARA_PARSE_RESULT_OK : succeed
- ESP_AT_PARA_PARSE_RESULT_FAIL : fail
- ESP_AT_PARA_PARSE_RESULT_OMITTED : this parameter is OMITTED

esp_at_para_parse_result_type **esp_at_get_para_as_str** (int32_t para_index, uint8_t **result)

Parse string parameter from command string.

Parameters

- **para_index** –the index of parameter
- **result** –the pointer that point to the result.

Returns

- ESP_AT_PARA_PARSE_RESULT_OK : succeed
- ESP_AT_PARA_PARSE_RESULT_FAIL : fail
- ESP_AT_PARA_PARSE_RESULT_OMITTED : this parameter is OMITTED

void **esp_at_port_rcv_data_notify_from_isr** (int32_t len)

Calling the esp_at_port_rcv_data_notify_from_isr to notify at module that at port received data. When received this notice,at task will get data by calling get_data_length and read_data in esp_at_device_ops. This function MUST be used in isr.

Parameters

len –data length

bool **esp_at_port_rcv_data_notify** (int32_t len, uint32_t msec)

Calling the esp_at_port_rcv_data_notify to notify at module that at port received data. When received this notice,at task will get data by calling get_data_length and read_data in esp_at_device_ops. This function MUST NOT be used in isr.

Parameters

- **len** –data length
- **msec** –timeout time,The unit is millisecond. It waits forever,if msec is port-MAX_DELAY.

Returns

- true : succeed
- false : fail

void **esp_at_transmit_terminal_from_isr** (void)

terminal transparent transmit mode,This function MUST be used in isr.

void **esp_at_transmit_terminal** (void)

terminal transparent transmit mode,This function MUST NOT be used in isr.

bool **esp_at_custom_cmd_array_regist** (const *esp_at_cmd_struct* *custom_at_cmd_array, uint32_t cmd_num)

register at command set, which defined by custom,

Parameters

- **custom_at_cmd_array** –at command set
- **cmd_num** –command number

void **esp_at_device_ops_register** (*esp_at_device_ops_struct* *ops)

register device operate functions set,

Parameters

ops –device operate functions set

bool **esp_at_custom_net_ops_register** (int32_t link_id, *esp_at_custom_net_ops_struct* *ops)

bool **esp_at_custom_ble_ops_register** (int32_t conn_index, *esp_at_custom_ble_ops_struct* *ops)

void **esp_at_custom_ops_register** (*esp_at_custom_ops_struct* *ops)

register custom operate functions set interacting with AT,

Parameters

ops –custom operate functions set

uint32_t **esp_at_get_version** (void)

get at module version number,

Returns

at version bit31~bit24: at main version bit23~bit16: at sub version bit15~bit8 : at test version
bit7~bit0 : at custom version

void **esp_at_response_result** (uint8_t result_code)

response AT process result,

Parameters

result_code –see esp_at_result_code_string_index

int32_t **esp_at_port_write_data** (uint8_t *data, int32_t len)

write data into device,

Parameters

- **data** –data buffer to be written
- **len** –data length

Returns

- ≥ 0 : the real length of the data written
- others : fail.

int32_t **esp_at_port_active_write_data** (uint8_t *data, int32_t len)

call pre_active_write_data_callback() first and then write data into device,

Parameters

- **data** –data buffer to be written
- **len** –data length

Returns

- ≥ 0 : the real length of the data written
- others : fail.

int32_t **esp_at_port_read_data** (uint8_t *data, int32_t len)

read data from device,

Parameters

- **data** –data buffer
- **len** –data length

Returns

- ≥ 0 : the real length of the data read from device

- others : fail

bool **esp_at_port_wait_write_complete** (int32_t timeout_msec)

wait for transmitting data completely to peer device,

Parameters

timeout_msec –timeout time, The unit is millisecond.

Returns

- true : succeed, transmit data completely
- false : fail

int32_t **esp_at_port_get_data_length** (void)

get the length of the data received,

Returns

- >= 0 : the length of the data received
- others : fail

bool **esp_at_custom_cmd_line_terminator_set** (uint8_t *terminator)

Set AT command terminator, by default, the terminator is “\r\n” You can change it by calling this function, but it just supports one character now.

Parameters

terminator –the line terminator

Returns

- true : succeed, transmit data completely
- false : fail

uint8_t ***esp_at_custom_cmd_line_terminator_get** (void)

Get AT command line terminator, by default, the return string is “\r\n” .

Returns

the command line terminator

const esp_partition_t ***esp_at_custom_partition_find** (esp_partition_type_t type,
esp_partition_subtype_t subtype, const char
*label)

Find the partition which is defined in at_customize.csv.

Parameters

- **type** –the type of the partition
- **subtype** –the subtype of the partition
- **label** –Partition label

Returns

pointer to esp_partition_t structure, or NULL if no partition is found. This pointer is valid for the lifetime of the application

void **esp_at_port_enter_specific** (*esp_at_port_specific_callback_t* callback)

Set AT core as specific status, it will call callback if receiving data. for example:

```
static void wait_data_callback (void)
{
    xSemaphoreGive(sync_sema);
}

void process_task(void* para)
{
    vSemaphoreCreateBinary(sync_sema);
    xSemaphoreTake(sync_sema, portMAX_DELAY);
    esp_at_port_write_data((uint8_t *) ">", strlen(">"));
    esp_at_port_enter_specific(wait_data_callback);
}
```

(continues on next page)

(continued from previous page)

```
while(xSemaphoreTake(sync_sema,portMAX_DELAY)) {
    len = esp_at_port_read_data(data, data_len);
    // TODO:
}
}
```

Parameters**callback** –which will be called when received data from AT portvoid **esp_at_port_exit_specific** (void)

Exit AT core as specific status.

const uint8_t ***esp_at_get_current_cmd_name** (void)

Get current AT command name.

int32_t **esp_at_get_core_version** (char *buffer, uint32_t size)

Get the version of the AT core library.

Parameters

- **buffer** –buffer to store the version string
- **size** –size of the buffer

Returns

- > 0 : the real length of the version string
- others : fail

bool **at_str_is_null** (uint8_t *str)

Check if the string is NULL.

Parameters**str** –the string to be checked**Returns**

- true : the string is NULL
- false : the string is not NULL

void **at_handle_result_code** (*esp_at_result_code_string_index* code, void *pbuf)

6.19.3 Structures

struct **esp_at_cmd_struct***esp_at_cmd_struct* used for define at command**Public Members**char ***at_cmdName**

at command name

uint8_t (***at_testCmd**)(uint8_t *cmd_name)

Test Command function pointer

uint8_t (***at_queryCmd**)(uint8_t *cmd_name)

Query Command function pointer

uint8_t (***at_setupCmd**)(uint8_t para_num)

Setup Command function pointer

uint8_t (***at_exeCmd**)(uint8_t *cmd_name)

Execute Command function pointer

struct **esp_at_device_ops_struct**

esp_at_device_ops_struct device operate functions struct for AT

Public Members

int32_t (***read_data**)(uint8_t *data, int32_t len)

read data from device

int32_t (***write_data**)(uint8_t *data, int32_t len)

write data into device

int32_t (***get_data_length**)(void)

get the length of data received

bool (***wait_write_complete**)(int32_t timeout_msec)

wait write finish

struct **esp_at_custom_net_ops_struct**

esp_at_custom_net_ops_struct custom socket callback for AT

Public Members

int32_t (***recv_data**)(uint8_t *data, int32_t len)

callback when socket received data

void (***connect_cb**)(void)

callback when socket connection is built

void (***disconnect_cb**)(void)

callback when socket connection is disconnected

struct **esp_at_custom_ble_ops_struct**

esp_at_custom_ble_ops_struct custom ble callback for AT

Public Members

int32_t (***recv_data**)(uint8_t *data, int32_t len)

callback when ble received data

void (***connect_cb**)(void)
callback when ble connection is built

void (***disconnect_cb**)(void)
callback when ble connection is disconnected

struct **esp_at_custom_ops_struct**
esp_at_ops_struct some custom function interacting with AT

Public Members

void (***status_callback**)(*esp_at_status_type* status)
callback when AT status changes

void (***pre_sleep_callback**)(*at_sleep_mode_t* mode)
callback before entering light sleep

void (***pre_wakeup_callback**)(void)
callback before waking up from light sleep

void (***pre_deepsleep_callback**)(void)
callback before enter deep sleep

void (***pre_restart_callback**)(void)
callback before restart

void (***pre_active_write_data_callback**)(*at_write_data_fn_t*)
callback before write data

6.19.4 Macros

at_min (x, y)

at_max (x, y)

ESP_AT_ERROR_NO (subcategory, extension)

ESP_AT_CMD_ERROR_OK

No Error

ESP_AT_CMD_ERROR_NON_FINISH

terminator character not found (“\r\n” expected)

ESP_AT_CMD_ERROR_NOT_FOUND_AT

Starting “AT” not found (or at, At or aT entered)

ESP_AT_CMD_ERROR_PARA_LENGTH (which_para)

parameter length mismatch

ESP_AT_CMD_ERROR_PARA_TYPE (which_para)

parameter type mismatch

ESP_AT_CMD_ERROR_PARA_NUM (need, given)

parameter number mismatch

ESP_AT_CMD_ERROR_PARA_INVALID (which_para)

the parameter is invalid

ESP_AT_CMD_ERROR_PARA_PARSE_FAIL (which_para)

parse parameter fail

ESP_AT_CMD_ERROR_CMD_UNSUPPORT

the command is not supported

ESP_AT_CMD_ERROR_CMD_EXEC_FAIL (result)

the command execution failed

ESP_AT_CMD_ERROR_CMD_PROCESSING

processing of previous command is in progress

ESP_AT_CMD_ERROR_CMD_OP_ERROR

the command operation type is error

6.19.5 Type Definitions

typedef int32_t (***at_read_data_fn_t**)(uint8_t *data, int32_t len)

typedef int32_t (***at_write_data_fn_t**)(uint8_t *data, int32_t len)

typedef int32_t (***at_get_data_len_fn_t**)(void)

typedef void (***esp_at_port_specific_callback_t**)(void)

AT specific callback type.

6.19.6 Enumerations

enum **esp_at_status_type**

esp_at_status some custom function interacting with AT

Values:

enumerator **ESP_AT_STATUS_NORMAL**

Normal mode. Now mcu can send AT command

enumerator **ESP_AT_STATUS_TRANSMIT**

Transparent Transmission mode

enum **at_sleep_mode_t**

Values:

enumerator **AT_DISABLE_SLEEP**

enumerator **AT_MIN_MODEM_SLEEP**

enumerator **AT_LIGHT_SLEEP**

enumerator **AT_MAX_MODEM_SLEEP**

enumerator **AT_SLEEP_MAX**

enum **esp_at_module**

module number, Now just AT module

Values:

enumerator **ESP_AT_MODULE_NUM**

AT module

enum **esp_at_error_code**

subcategory number

Values:

enumerator **ESP_AT_SUB_OK**

OK

enumerator **ESP_AT_SUB_COMMON_ERROR**

reserved

enumerator **ESP_AT_SUB_NO_TERMINATOR**

terminator character not found (“\r\n” expected)

enumerator **ESP_AT_SUB_NO_AT**

Starting “AT” not found (or at, At or aT entered)

enumerator **ESP_AT_SUB_PARA_LENGTH_MISMATCH**

parameter length mismatch

enumerator **ESP_AT_SUB_PARA_TYPE_MISMATCH**

parameter type mismatch

enumerator **ESP_AT_SUB_PARA_NUM_MISMATCH**

parameter number mismatch

enumerator **ESP_AT_SUB_PARA_INVALID**

the parameter is invalid

enumerator **ESP_AT_SUB_PARA_PARSE_FAIL**

parse parameter fail

enumerator **ESP_AT_SUB_UNSupport_CMD**

the command is not supported

enumerator **ESP_AT_SUB_CMD_EXEC_FAIL**

the command execution failed

enumerator **ESP_AT_SUB_CMD_PROCESSING**

processing of previous command is in progress

enumerator **ESP_AT_SUB_CMD_OP_ERROR**

the command operation type is error

enum **esp_at_para_parse_result_type**

the result of AT parse

Values:

enumerator **ESP_AT_PARA_PARSE_RESULT_FAIL**

parse fail,Maybe the type of parameter is mismatched,or out of range

enumerator **ESP_AT_PARA_PARSE_RESULT_OK**

Successful

enumerator **ESP_AT_PARA_PARSE_RESULT_OMITTED**

the parameter is OMITTED.

enum **esp_at_result_code_string_index**

the result code of AT command processing

Values:

enumerator **ESP_AT_RESULT_CODE_OK**

“OK”

enumerator **ESP_AT_RESULT_CODE_ERROR**

“ERROR”

enumerator **ESP_AT_RESULT_CODE_FAIL**

“ERROR”

enumerator **ESP_AT_RESULT_CODE_SEND_OK**

“SEND OK”

enumerator **ESP_AT_RESULT_CODE_SEND_FAIL**

“SEND FAIL”

enumerator **ESP_AT_RESULT_CODE_IGNORE**

response nothing, just change internal status

enumerator **ESP_AT_RESULT_CODE_PROCESS_DONE**

response nothing, just change internal status

enumerator **ESP_AT_RESULT_CODE_OK_AND_INPUT_PROMPT**

enumerator **ESP_AT_RESULT_CODE_MAX**

6.19.7 Header File

- [components/at/include/esp_at.h](#)

6.19.8 Functions

const char ***esp_at_get_current_module_name** (void)

get current module name

const char ***esp_at_get_module_name_by_id** (uint32_t id)

get module name by index

uint32_t **esp_at_get_module_id** (void)

get current module id

void **esp_at_set_module_id** (uint32_t id)

Set current module id.

Parameters

id –[in] the module id to set

void **esp_at_set_module_id_by_str** (const char *buffer)

Get module id by module name.

Parameters

buffer –[in] pointer to a module name string

void **esp_at_main_preprocess** (void)

some workarounds for esp-at project

[at_mfg_params_storage_mode_t](#) **at_get_mfg_params_storage_mode** (void)

get storage mode of mfg parameters

Returns

[at_mfg_params_storage_mode_t](#)

void **esp_at_ready_before** (void)

Do some things before esp-at is ready.

Note: This function can be overridden with custom implementation. For example, you can override this function to: a) execute some preset AT commands by calling `at_exe_cmd()` API. b) do some initializations by calling APIs from `esp-idf` or `esp-at`.

void **esp_at_log_write** (esp_log_level_t level, const char *tag, const char *format, ...)

Write message into the log.

This function is not intended to be used directly. Instead, use one of `ESP_AT_LOGE`, `ESP_AT_LOGW`, `ESP_AT_LOGI`, `ESP_AT_LOGD`, `ESP_AT_LOGV` macros.

This function or these macros should not be used from an interrupt.

esp_err_t **esp_at_nvs_set_str** (nvs_handle_t handle, const char *key, const char *value)

NVS operations for string and blob.

See also:

nvs_set_str, nvs_get_str, nvs_set_blob, nvs_get_blob

Note: These functions can be overridden with custom implementations. For example: you can override these functions to encrypt the data before writing to NVS (decrypt the data after reading from NVS).

esp_err_t **esp_at_nvs_get_str** (nvs_handle_t handle, const char *key, char *out_value, size_t *length)

esp_err_t **esp_at_nvs_set_blob** (nvs_handle_t handle, const char *key, const void *value, size_t length)

esp_err_t **esp_at_nvs_get_blob** (nvs_handle_t handle, const char *key, void *out_value, size_t *length)

void **esp_at_yield_if_idle_timeout** (uint32_t idle_timeout_ms, uint32_t yield_ticks)

Yield the current task if the system has not recently executed the idle task.

This function measures the elapsed time since the last execution of the FreeRTOS idle hook. If that duration exceeds the specified threshold, the current task voluntarily delays for a given number of RTOS ticks to allow lower-priority tasks (including the idle task) to run.

This mechanism is intended to prevent long-running tasks from monopolizing the CPU under high workload conditions (e.g., heavy RX traffic), which may otherwise trigger the task watchdog.

Note: This function must not be called from ISR context.

Parameters

- **idle_timeout_ms** –[in] Threshold in milliseconds. If the time since the last idle hook execution exceeds this value, a yield is performed.
- **yield_ticks** –[in] Number of RTOS ticks to delay when yielding.

const char ***esp_at_fs_get_mount_point** (void)

Get the mount point of the filesystem.

Returns

pointer to the mount point string

bool **esp_at_fs_mount** (void)

Mount the filesystem.

Returns

- true: mount successfully
- false: mount failed

bool **esp_at_fs_unmount** (void)

Unmount the filesystem.

Returns

- true: unmount successfully
- false: unmount failed

esp_err_t **esp_at_get_fs_info** (uint32_t *out_total_bytes, uint32_t *out_free_bytes)

Get the filesystem information.

Parameters

- **out_total_bytes** –[out] pointer to store total bytes of the filesystem
- **out_free_bytes** –[out] pointer to store free bytes of the filesystem

Returns

- ESP_OK: get filesystem info successfully
- others: see esp_err.h

at_fs_type_t **esp_at_fs_get_type** (void)

Get the filesystem type.

Returns

the filesystem type, see at_fs_type_t

6.19.9 Macros

ESP_AT_PORT_TX_WAIT_MS_MAX

AT_BUFFER_ON_STACK_SIZE

ESP_AT_LOG_BUFFER_HEXDUMP (tag, buffer, buff_len, level)

Same to ESP_LOG_BUFFER_HEXDUMP, but only output when buffer is not NULL.

CONFIG_AT_LOG_DEFAULT_LEVEL

ESP_AT_LOGE (tag, format, ...)

runtime macro to output logs at a specified level.

See also:

`printf`

Parameters

- **tag**—tag of the log, which can be used to change the log level by `esp_log_level_set` at runtime.
- **format**—format of the output log. See `printf`
- **...**—variables to be replaced into the log. See `printf`

ESP_AT_LOGW (tag, format, ...)

ESP_AT_LOGI (tag, format, ...)

ESP_AT_LOGD (tag, format, ...)

ESP_AT_LOGV (tag, format, ...)

6.19.10 Enumerations

enum **at_mfg_params_storage_mode_t**

Values:

enumerator **AT_PARAMS_NONE**

enumerator **AT_PARAMS_IN_MFG_NVS**

enumerator **AT_PARAMS_IN_PARTITION**

6.20 How to Enable Silence Mode

6.20.1 Introduction

Silence mode is a compilation configuration option for ESP-AT firmware that controls the behavior of AT logs (i.e., logs output from the [AT log port](#)). It is generally recommended to disable this option. While enabling it may reduce the size of the application firmware, you should **prioritize** the following two methods for reducing the firmware size:

- In [python build.py menuconfig](#), navigate to Component config > AT to disable the unnecessary AT features.
- Refer to the optimization methods outlined in the [Minimizing Binary Size](#) documentation.

Table 4: Advantages and Disadvantages of Enabling and Disabling Silence Mode

Silence Mode	Advantages	Disadvantages
Disabled (i.e., set to 0, recommended)	Enable logging for some AT functions	The size of the application firmware (esp-at.bin) will increase, potentially exceeding the allocated application partition size in the code and resulting in a compilation failure (refer to compilation failure log).
Enabled (i.e., set to 1, not recommended)	Reduced size of the application firmware (esp-at.bin)	This mode will remove some AT logs and make the <code>CONFIG_LOG_DEFAULT_LEVEL</code> configuration ineffective, preventing AT from outputting logs at the corresponding log levels, which will increase the difficulty of debugging AT functions.

Compilation Failure Log Example:

```
$ ./build.py build

... (more lines of build ESP-AT)

Generated /path/to/esp-at/build/esp-at.bin
[836/838] cd /path/to/esp-at/build/esp-at.bin
FAILED: esp-idf/esptool_py/CMakeFiles/app_check_size
cd /path/to/esp-at/build/esp-idf/esptool_py && [...] /bin/python /path/to/esp-at/
↳ esp-idf/components/partition_table/check_sizes.py --offset 0x8000 partition --
↳ type app /path/to/esp-at/build/partition_table/partition-table.bin /path/to/esp-
↳ at/build/esp-at.bin
Error: app partition is too small for binary esp-at.bin size 0x135ae0:
- Part 'ota_0' 0/16 @ 0xd0000 size 0x130000 (overflow 0x5ae0)
ninja: build stopped: subcommand failed.
```

6.20.2 Configuration

- Enable or disable the silence mode configuration:
 - [Compile ESP-AT Project Locally](#)
Configure the `silence_mode` as No (or Yes) in step 3 Install Environment to disable (or enable) the silence mode.
 - [Compile ESP-AT Project on the GitHub Webpage](#)
Modify the value of `silence_mode` in the [esp-at/.github/workflows/build_template_esp32.yml](#) file in step 5.3 Commit changes to enable or disable the silence mode. For example, to enable silence mode, set it to:

```

- name: Configure prerequisites
  run: |
    # set module information
    silence_mode=1
    # set the silence_sdkconfig.defaults file path
    silence_sdkconfig=module_config/module_<your_module_name>/sdkconfig_
    ↪silence.defaults
    mkdir build
    echo -e "{\"platform\": \"PLATFORM_ESP32\", \"module\": \"${ inputs.
    ↪module_name }}\", \"silence\": ${silence_mode}\" > build/module_info.json

```

- Please flash the compiled AT firmware to the module, and then execute the **AT+GMR** command to confirm whether the enable or disable of silence mode is successful.

- If silence mode is enabled, the response of the AT+GMR command should include s-, for example:

```

AT+GMR
AT version:3.5.0.0-dev(s-88b4ea4...

... (more lines of version information)

OK

```

- If silence mode is disabled, the response of the AT+GMR command should **NOT** include s-, for example:

```

AT+GMR
AT version:3.5.0.0-dev(88b4ea4...

... (more lines of version information)

OK

```

6.21 How to Enable More AT Debug Logs

This article mainly introduces how to enable log configuration options. To obtain more log information for debugging ESP-AT functionalities, please refer to the guide to [compile the ESP-AT project](#), and in step three, set the silence mode to No. In step five, based on your actual situation, enable the specific logging configuration options.

- [AT Wi-Fi Function Debugging](#)
- [AT Network Function Debugging](#)
- [AT BLE Function Debugging](#)
- [AT Interface Function Debugging](#)
- [Debugging Examples](#)

The final debugging logs will be output from the **AT log port**. Please refer to the [Hardware Connection](#) document to find the TX GPIO pin corresponding to the **Output Log** and use a serial tool to view the logs on that pin.

Note: Enabling log configuration options will significantly increase the log output. Therefore, it is recommended to first confirm which module the issue belongs to and then enable the corresponding module's log configuration. This prevents issues such as being unable to reproduce or encountering new problems like `task wdt` due to excessive log output.

Enabling log configuration options will increase the firmware size. This may lead to compilation failures. To resolve this, you can reduce the firmware size by disabling unused AT functionalities via the configuration menu: `python build.py menuconfig>Component config>AT`.

6.21.1 AT Wi-Fi Function Debugging

- You can enable the following configurations to turn on logging for Wi-Fi packet statistics:

```
Component config > Log > Log Level > Default log verbosity > Info (or Info and
↪above levels)
Component config > AT > Enable ESP-AT Debug
Component config > AT > Enable ESP-AT Debug > Periodically dump Wi-Fi
↪statistics
Component config > AT > Enable ESP-AT Debug > Periodically dump Wi-Fi
↪statistics > The interval of dumping Wi-Fi statistics (ms)
```

- You can enable the following configurations to turn on debugging logs for the Wi-Fi interaction process:

```
Component config > Log > Log Level > Default log verbosity > Debug
Component config > Supplicant > Print debug messages from WPA Supplicant
```

Note: When analyzing logs, you may encounter Wi-Fi reason codes. For the specific meanings, please refer to the [Wi-Fi Reason Code](#).

6.21.2 AT Network Function Debugging

- [TCP Debugging](#)
- [UDP Debugging](#)
- [SSL Debugging](#)
- [ICMP Debugging](#)
- [lwIP Network Debugging](#)

TCP Debugging

To monitor outgoing (TX) and incoming (RX) packets for TCP or TCP-based protocols (such as SSL, HTTP, MQTT, WebSocket, etc.), enable the following configuration. The final logs will print TCP packet information, including IP total length, TCP data length, TCP sequence number, TCP acknowledgment number, TCP source port, destination port, and TCP flags. The TCP flags are cumulative.

```
Component config > Log > Log Level > Default log verbosity > Info (or
↪Info and above levels)
Component config > AT > Enable ESP-AT Debug
Component config > AT > Enable ESP-AT Debug > Enable Network Debug
Component config > AT > Enable ESP-AT Debug > Enable Network Debug >
↪Enable the TCP packet debug messages
Component config > AT > Enable ESP-AT Debug > Enable Network Debug >
↪Enable the TCP packet debug messages > Specify the list of TCP port
↪numbers to monitor > 0
```

UDP Debugging

To monitor outgoing (TX) and incoming (RX) packets for UDP or UDP-based protocols (such as DHCP, DNS, SNTP, mDNS, etc.), enable the following configuration. The final logs will print UDP packet information, including IP total length, source port, destination port, and UDP data length.

```
Component config > Log > Log Level > Default log verbosity > Info (or
↪Info and above levels)
```

(continues on next page)

(continued from previous page)

```

Component config > AT > Enable ESP-AT Debug
Component config > AT > Enable ESP-AT Debug > Enable Network Debug
Component config > AT > Enable ESP-AT Debug > Enable Network Debug >_
  ↳ Enable the UDP packet debug messages
Component config > AT > Enable ESP-AT Debug > Enable Network Debug >_
  ↳ Enable the UDP packet debug messages > Specify the list of outgoing UDP_
  ↳ (UDP TX) port numbers to monitor
Component config > AT > Enable ESP-AT Debug > Enable Network Debug >_
  ↳ Enable the UDP packet debug messages > Specify the list of incoming UDP_
  ↳ (UDP RX) port numbers to monitor

```

SSL Debugging

You can enable the following configurations to enable debugging logs for SSL functionality:

```

Component config > Log > Log Level > Default log verbosity > Verbose
Component config > mbedTLS > Enable mbedTLS debugging
Component config > mbedTLS > Enable mbedTLS debugging > Set mbedTLS_
  ↳ debugging level > Verbose

```

ICMP Debugging

You can enable the following configurations to enable debugging logs for the ICMP functionality (*AT+PING*):

```

Component config > Log > Log Level > Default log verbosity > Info (or_
  ↳ Info and above levels)
Component config > AT > Enable ESP-AT Debug
Component config > AT > Enable ESP-AT Debug > Enable Network Debug
Component config > AT > Enable ESP-AT Debug > Enable Network Debug >_
  ↳ Enable the ICMP packet debug messages

```

LWIP Network Debugging

If the logs output by enabling the above configurations still do not meet your debugging needs, you can enable the required debugging configurations under the path `Component config > LWIP > Enable LWIP Debug`.

6.21.3 AT BLE Function Debugging

- You can directly view the disconnection reason code on the *AT log port* to debug the BLE disconnection process.
- You can enable the following configurations to enable debugging logs for BLE scanning, connection, advertising, data transmission, and other interaction processes:

```

Component config > Log > Log Level > Default log verbosity > Debug
Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL > HCI_
  ↳ layer > DEBUG

```

- You can enable the following configurations to turn on debugging logs for the BLE pairing process:

```

Component config > Log > Log Level > Default log verbosity > Debug
Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL > HCI_
  ↳ layer > DEBUG
Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL > SMP_
  ↳ layer > DEBUG

```

- You can enable the following configurations to turn on BLE GATT layer debugging logs:

```
Component config > Log > Log Level > Default log verbosity > Debug
Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL > GATT_
↳layer > DEBUG
```

- You can enable the following configurations to turn on debugging logs for the AT BluFi functionality:

```
Component config > Log > Log Level > Default log verbosity > Debug
Component config > Bluetooth > Bluedroid Options > BT DEBUG LOG LEVEL > BLUFI_
↳layer > DEBUG
```

If the above configurations do not meet your debugging needs, you can enable other required debugging configurations under Component config>Bluetooth>Bluedroid Options>BT DEBUG LOG LEVEL.

6.21.4 AT Interface Function Debugging

- When you want to obtain the data sent from AT to MCU through TX at the *AT log port*, you can enable the following configurations:

```
Component config > Log > Log Level > Default log verbosity > Info (or Info and_
↳above levels)
Component config > AT > Enable ESP-AT Debug
Component config > AT > Enable ESP-AT Debug > Logging the data sent from AT to_
↳MCU (AT ---> MCU)
Component config > AT > Enable ESP-AT Debug > Logging the data sent from AT to_
↳MCU (AT ---> MCU) > The maximum length of the data sent from AT to MCU to be_
↳logged > 8192
```

- When you want to obtain the data sent from MCU to AT through RX at the *AT log port*, you can enable the following configurations:

```
Component config > Log > Log Level > Default log verbosity > Info (or Info and_
↳above levels)
Component config > AT > Enable ESP-AT Debug
Component config > AT > Enable ESP-AT Debug > Logging the data received by AT_
↳from MCU (AT <---- MCU)
Component config > AT > Enable ESP-AT Debug > Logging the data received by AT_
↳from MCU (AT <---- MCU) > The maximum length of the data received by AT from_
↳MCU to be logged > 8192
```

6.21.5 Debugging Examples

Example 1: Debugging the Data Sending and Receiving Process of TCP Connections

Scene: With a baud rate of 115200 and using full-duplex flow control, the ESP module establishes a TCP connection as a TCP client with the server. It enters transparent transmission mode and sends data to the server.

As shown by the arrows in the figure:

- The Data stream sent by ESP-AT is (TX): S1 -> S2 -> S3 -> S4 -> S5 -> S6 -> S7 -> S8
- The Data stream received by ESP-AT is (RX): R8 -> R7 -> R6 -> R5 -> R4 -> R3 -> R2 -> R1

To troubleshoot data transmission issues in the AT firmware across various communication layers (UART, AT interface, network, Wi-Fi), follow the structured guidance below and choose the appropriate debugging method:

- For S1/R1 (UART Communication):
Use a logic analyzer to capture the serial data sent from the MCU to the ESP module (TX -> RX lines) to verify UART communication. If any anomalies are detected (typically manifested as data errors or data loss), please check the wiring first. Common causes include excessively high baud rate, overly long UART connections between the MCU and ESP, incorrect bidirectional flow control configuration, or improper hardware design.

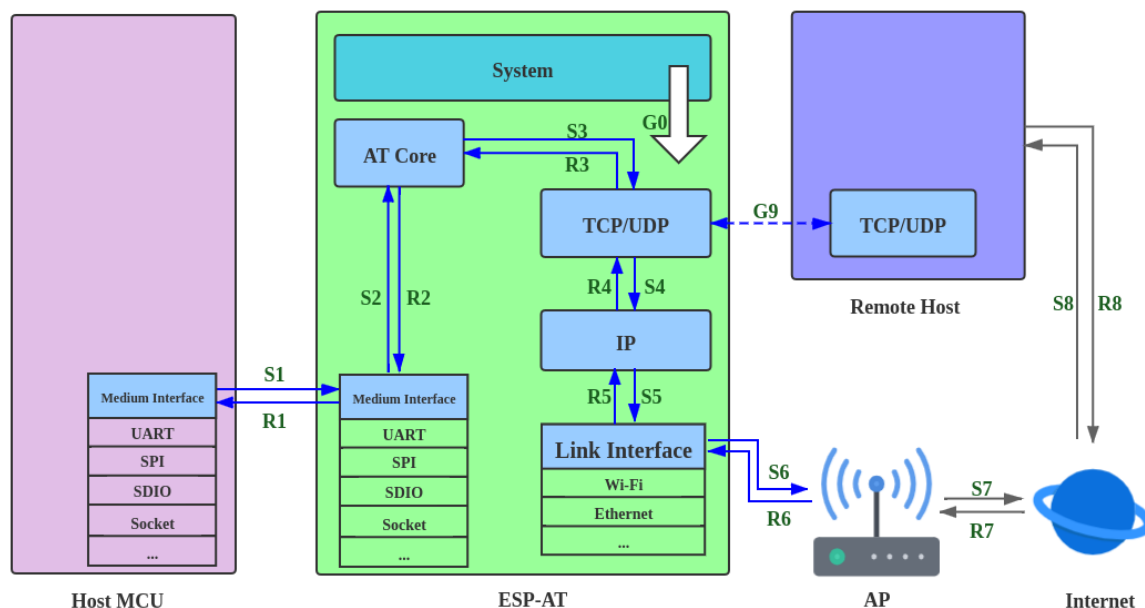


Fig. 38: Data Stream

- For S2/R2 (AT Interface Layer):
Enable the *AT Interface Debugging* configuration and recompile the AT firmware. During the retest, check the logs from the *AT log port* to ensure RX (receive) and TX (transmit) data are functioning correctly. If data loss is detected, please first confirm whether hardware flow control is enabled. If the issue is confirmed to be caused by AT, please submit an issue at [esp-at/issues](#).
- For S3/R3 (AT Internal Processing):
No user intervention is required.
- For S4-S5/R4-R5 (Network Layer):
Enable the *TCP Debugging* configuration and recompile the AT firmware. During the retest, analyze the logs from the *AT log port* to verify LwIP layer transmission. If you find issues, submit an issue at [esp-at/issues](#).
- For S6/R6 (Wi-Fi Layer):
Enable the *Wi-Fi Debugging* configuration and recompile the AT firmware. During the retest, refer to the *Espressif Wireshark User Guide* to capture airborne packets, and analyze the Wi-Fi layer data transmission alongside the logs from the *AT log port*. If issues are confirmed at the Wi-Fi layer, submit an issue at [esp-at/issues](#). If the problem originates from the server side, please investigate accordingly.

Example 2: Debugging BLE connection and scanning processes

Scene: The ESP module, as a BLE client, repeatedly establishes and disconnects BLE connection with the mobile phone.

- If you want to identify the cause of the BLE disconnection, please capture the debug logs from the *AT log port* during the retest for analysis. The logs may contain disconnection reason codes, which are common to BLE and not specific to AT. Example:

```
Disconnect reason = 0x13
```

- If you want to debug processes such as BLE scanning or connection, please enable the configurations for *AT BLE Function Debugging* and compile the AT firmware. During the retest, capture the debug logs from the *AT log port* for analysis.

Chapter 7

Customized AT Commands and Firmware For Third-party Open Cloud Platforms

To be developed.

If you have any requirements on custom AT commands and firmware for third-party open cloud platforms, please contact [Espressif's Business Team](#) for an evaluation.

Chapter 8

AT FAQ

- *AT Documentation*
 - *How to select the correct AT documentation version?*
- *AT Firmware*
 - *There is no released firmware for my module*
 - *How to get the source code of AT firmware?*
 - *How to download the AT firmware?*
 - *How to combine all the bin files compiled by ESP-AT?*
 - *Why does a newly purchased ESP series module print “flash read err” error after power-up? How to use AT commands?*
 - *Does the AT firmware shipped in modules support flow control?*
- *AT Commands and Responses*
 - *Why does AT return busy p...?*
 - *Why does the first command after power-up return an error?*
 - *What commands are supported by default AT firmware?*
 - *MCU sends AT command with no response*
 - *How to handle special characters in AT commands?*
 - *Can the UART baudrate be modified?*
 - *How to modify the UART pins of the AT command port?*
 - *How to solve TCP/SSL data loss during transmission?*
 - *Can passthrough mode notify hotspot disconnection?*
 - *Why does Wi-Fi disconnect?*
 - *What are the common Wi-Fi compatibility issues?*
 - *Do AT commands support ESP-WIFI-MESH?*
 - *Are there examples for connecting to cloud services?*
 - *Set ESP32-WROOM-32 module to HID keyboard mode?*
 - *Can AT command set Bluetooth LE transmit power?*
 - *How to enable notify and indicate on BLE clients?*
- *Hardware*
 - *How big is the chip flash required for AT firmware?*
 - *How to view the error log of AT firmware?*
 - *UART1 communication pin inconsistent with datasheet?*
- *Performance*
 - *How long does it take for AT to connect to Wi-Fi?*
 - *Is it possible to change the TCP send window size?*
 - *How to test and optimize the throughput?*

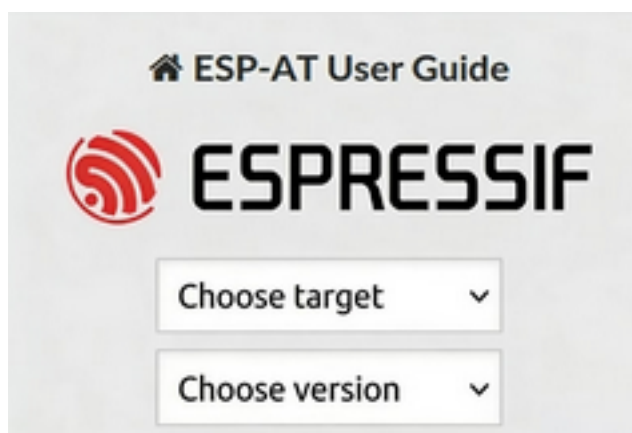
- *How to modify TCP segment retransmissions?*
- *Other*
 - *What interfaces can be used to transmit AT commands?*
 - *How to enable debug log for AT?*
 - *How to modify the default Wi-Fi mode?*
 - *How to implement HTTP resume transfer functionality?*
 - *How to download files from HTTP server and store them to FATFS, or upload files from FATFS file system to HTTP server?*
 - *How to conduct BQB certification?*
 - *Does AT support PPP?*

8.1 AT Documentation

8.1.1 How to select the correct AT documentation version?

Please select the corresponding chip platform and documentation version based on the actual AT firmware version you are using; there may be functional and behavioral differences between different firmware versions.

- For *Officially Released Firmware (Recommended)*, you can directly obtain the corresponding document link through the *AT+USERDOCS?* command, without needing to confirm the version information.
- For non-official released firmware, please confirm the following information first, and then select the corresponding document version:
 - The chip platform of the AT firmware you are using (e.g., ESP32, ESP32-C3, etc.)
 - The version number of the AT firmware (you can use the *AT+GMR* command to view the version number of the AT core library)



8.2 AT Firmware

8.2.1 There is no released firmware for my module. Where can I get the firmware for it?

If there is no released firmware for your module in *AT Binary Lists*, consider:

- Use the firmware for a module with the same hardware configuration (see *ESP-AT Firmware Differences*)
- Refer to *Which Type of AT Firmware Shall I Choose?*

8.2.2 How to get the source code of AT firmware?

The esp-at project is distributed as a combination of source code and pre-compiled libraries in this repository. The pre-compiled core libraries (under the directory: `esp-at/components/at/lib/`) are closed-source, and there is no plan for open-sourcing them.

8.2.3 How to download the AT firmware on Espressif's official website?

Download the [Flash Download Tool](#). Refer to the [Flash Download Tool User Guide](#) for instructions. See [AT Downloading Guide](#) for the download address.

8.2.4 How to combine all the bin files compiled by ESP-AT?

No manual combination is required. The released and compiled firmware already includes pre-combined 2MB/4MB bin files in the `build/factory` directory.

8.2.5 Why does a newly purchased ESP series module print “flash read err” error after power-up? How to use AT commands?

ESP series modules are shipped without ESP-AT firmware in Flash, so the error “flash read err” is reported after power-up. To use AT commands, refer to [AT Downloading Guide](#).

8.2.6 Does the AT firmware shipped in modules support flow control?

Hardware flow control is supported, but software flow control is not. Configure hardware flow control using `AT+UART_CUR` or `AT+UART_DEF`. See [Hardware connection](#) for wiring details.

8.3 AT Commands and Responses

8.3.1 Why does AT prompt busy p...?

The “busy p...” prompt indicates that the previous command is being executed, and the system cannot respond to the current input. AT commands are processed serially, so you must wait for the previous command to complete before receiving the next one.

When there are extra invisible characters (such as spaces after a command), the system will also prompt “busy p...” or “ERROR”, because any serial port input is considered a command input.

If the MCU sends `AT+CIPSEND` and receives a busy p... response, it means the previous command is still executing and the current input is invalid. Wait for the previous command response to complete before sending a new command.

8.3.2 Why does the AT firmware always return the following message after I power up the device and sent the first command?

```
ERR CODE:0x010b0000
busy p...
```

This message means “the previous command is being executed”. Normally only “busy p...” is displayed. The ERR CODE is shown because the error code prompt is enabled.

If you receive this error on the first command after power-up, possible reasons:

- The command is followed by unnecessary newline/space/other symbols

- Multiple AT commands were sent in succession

8.3.3 What commands are supported by the default AT firmware on different modules, and from which version are they supported?

Commands supported by default firmware on different modules: refer to [ESP-AT Firmware Differences](#)

Command version support and fixes: refer to [release notes](#)

8.3.4 When the host MCU sends an AT command to the ESP32 device, there is no response. What is the reason?

A terminator (“AT\r\n”) must be added after an AT command. See [Check Whether AT Works](#).

8.3.5 How to handle special characters in AT commands?

Refer to the escape character syntax described in the [AT Command Types](#) section.

8.3.6 Can the serial port baudrate be modified in AT Commands? (Default: 115200)

Yes, you can use either of the three ways below to modify it:

- Use the command [AT+UART_CUR](#) or [AT+UART_DEF](#).
- Re-compile the AT firmware: [establish the compiling environment](#) and [change the UART baudrate](#).
- Use the [AT+SYSMFG](#) command to modify the factory default UART configuration, please refer to [AT+SYSMFG command examples](#).

8.3.7 How to modify the UART pins of the AT command port?

You can use either of the three ways below to modify it:

- Please refer to the [How to set AT port pins](#) document. Modify the UART configuration parameters in the `factory_param_data.csv` file, then recompile and flash the firmware.
- If you want to modify the UART configuration at runtime through AT commands without recompiling the firmware, please refer to [AT+SYSMFG command examples](#). This method allows you to dynamically modify the factory default UART configuration, including baudrate, pins, and other parameters.
- If you already have AT firmware and only need to modify the UART pins, you can use the [at.py tool](#) to directly modify the firmware parameters without recompiling.

8.3.8 How to solve TCP/SSL data loss during transmission?

If data loss occurs without hardware flow control enabled, it is recommended to enable hardware flow control ([AT+UART_CUR](#) or [AT+UART_DEF](#)) to verify if it is caused by the lack of flow control. If the problem persists after enabling flow control, please refer to [Debugging the Data Sending and Receiving Process of TCP Connections](#) for further troubleshooting; if it is confirmed to be an issue caused by AT, please submit an issue on [esp-at/issues](#).

8.3.9 After ESP32 enters the passthrough mode using AT commands, can ESP32 give a message if the connected hotspot is disconnected?

Configure with [AT+SYSMFG](#), set AT+SYSMFG=4. The serial port will report `WIFI_DISCONNECT\r\n` when the connected hotspot is disconnected.

Note: This command was added after AT v2.1.0.0. It is not available for v2.1.0.0 and earlier versions.

8.3.10 Why is Wi-Fi disconnected (WIFI DISCONNECT)?

Check the disconnection reason on [AT log port](#), which prints `wifi disconnected, rc:<reason_code>`. See [Wi-Fi Reason Code](#) for details.

Generic firmware may not provide detailed reasons. To obtain specific information, register the Wi-Fi disconnect event in code and output reason code.

8.3.11 What are the common Wi-Fi compatibility issues?

- AMPDU compatibility issue.
 - If the router does not support AMPDU, ESP32 will automatically disable the AMPDU function when interacting with the router.
 - If the router supports AMPDU but there is a compatibility issue with AMPDU transmission between the router and ESP32, it is recommended to disable the function on the router or ESP32. For information on how to disable it on ESP32, please refer to [Compile ESP-AT Project Locally](#) and select the following options in the fifth step of configuring the project:
 - * Disable Component `config>Wi-Fi>WiFi AMPDU TX`
 - * Disable Component `config>Wi-Fi>WiFi AMPDU RX`
- Phy mode compatibility issue. If there is a compatibility issue with the phy mode between the router and ESP32, it is recommended to switch it on the router or ESP32. For how to switch it on ESP32, please refer to the [AT+CWSTAPROTO](#) command.

8.3.12 Do AT commands support ESP-WIFI-MESH?

Currently, AT commands do not support ESP-WIFI-MESH.

8.3.13 Is it possible to set the ESP32-WROOM-32 module to HID keyboard mode with AT commands?

Yes, refer to [Bluetooth LE AT Commands](#).

8.3.14 Can AT command set Bluetooth LE transmit power?

Yes, use the [AT+RFPOWER](#) command to set Bluetooth LE transmit power. ESP32 Wi-Fi and Bluetooth LE share the same antenna.

8.3.15 How to enable the notify and indicate functions on Bluetooth LE clients?

- Besides the read and write properties, Bluetooth LE characteristics also have the `notify` and `indicate` properties, which allow the server to send data to the client, but the client must first register for notification by writing the value of “CCCD”.
- To enable `notify`, write 0x01. To enable `indicate`, write 0x02 (for the descriptor “0x2902”). To enable both `notify` and `indicate`, write 0x03.
- The example below demonstrates how to enable the `notify` and `indicate` properties for the descriptor 0x2902 under the services 0xC305 and 0xC306 respectively. ESP-AT includes these two services by default. You can `notify` 0xC305 and `indicate` 0xC306.

```
AT+BLEGATTCWR=0,3,6,1,2
>
// Write low byte 0x01 high byte 0x00 (if you want to use hex format, it is:
→0100)
OK
// Server: +WRITE:0,1,6,1,2,<0x01>,<0x00>
AT+BLEGATTCWR=0,3,7,1,2
>
// Write low byte 0x02 high byte 0x00 (if you want to use hex format, it is:
→0200)
OK
// Server: +WRITE:0,1,6,1,2,<0x02>,<0x00>
// Writing ccc is a prerequisite for the server to be able to send notify and
→indicate
```

8.4 Hardware

8.4.1 How big is the chip flash required for AT firmware on different modules?

Refer to [ESP-AT Firmware Differences](#).

8.4.2 How to view the error log of AT firmware?

Usually check the error log from [AT log port](#).

8.4.3 The UART1 communication pin used by AT on the ESP32 module is inconsistent with the default UART1 pin described in the ESP32 module's datasheet?

ESP32 supports IO matrix. When compiling ESP-AT, you can configure UART1 pins in menuconfig, so they may be inconsistent with the pins described in the module datasheet. See [factory_param_data.csv](#) for details.

8.5 Performance

8.5.1 How long does it take for AT to connect to Wi-Fi?

In an office scenario, AT takes about 5 seconds to connect. Actual connection time depends on router performance, network environment, module antenna performance, etc. The maximum timeout can be set using the `<jap_timeout>` parameter of [AT+CWJAP](#).

8.5.2 Is it possible to change the TCP send window size in AT firmware?

It cannot be changed by AT commands. Re-compile the firmware and configure in menuconfig: Component config>LWIP>TCP>Default send buffer size.

8.5.3 How to test and optimize the AT throughput?

It is recommended to use the iperf example in esp-idf for testing (use passthrough mode, adjust data length to 1460 bytes, and send data continuously). If the test rate does not meet your requirements, refer to [How to Improve ESP-AT Throughput Performance](#).

8.5.4 How to modify the default maximum number of TCP segment retransmissions for AT?

By default, the maximum number of TCP segment retransmissions is 6 (range: [3-12]). Reconfigure it as follows:

- **Local compilation:** Refer to [compile ESP-AT project locally](#), configure in menuconfig:

```
Component config > LWIP > TCP > Maximum number of retransmissions of data_
↳ segments
```

- **Web compilation:** Refer to [compile ESP-AT project on the webpage](#), modify the value of `CONFIG_LWIP_TCP_MAXRTX`.

8.6 Other

8.6.1 What interfaces of ESP32 chips can be used to transmit AT commands?

- ESP32 can transmit AT commands through UART and SDIO
- The default firmware uses UART for transmission. If you need SDIO or SPI interface, configure it when compiling the ESP-AT project yourself. See [How to Compile and Develop Your Own AT Project](#) for details
- Reference documents: [AT through SDIO](#) , [AT through SPI](#) , [AT through socket](#)

8.6.2 How to enable debug log for AT?

Refer to the [How to enable more AT debug logs](#) document.

8.6.3 How to modify the default Wi-Fi mode when AT firmware starts for the first time?

Modify the source code. Call `esp_wifi_set_mode()` in the `esp_at_ready_before()` function to set the Wi-Fi mode, for example: `esp_wifi_set_mode(WIFI_MODE_STA)`.

8.6.4 How does the AT command implement the functionality of resuming HTTP transfers after interrupts?

Two methods:

- Use the HTTP Range field to specify the data range to be read. Refer to the example of [AT+HTTPCHEAD](#)
- Construct an HTTP GET request using AT TCP commands. Between steps 6 and 7 of [ESP32 obtains socket data in passive receiving mode](#), use [AT+CIPSEND](#) to send your own HTTP GET request. In passive receive mode, use [AT+CIPRECVDATA](#) to read data

8.6.5 How to download files from HTTP server and store them to FATFS, or upload files from FATFS file system to HTTP server?

Refer to GitHub examples:

- [at_http_get_to_fs](#) - Download files from HTTP server and store them to file system
- [at_fs_to_http_server](#) - Upload files from file system to HTTP server

8.6.6 How to conduct BQB certification?

Please contact [Espressif](#) for solutions.

8.6.7 Does AT support PPP?

Not supported, refer to [pppos_client](#) demos for your own implementation.

Chapter 9

Index of Abbreviations

A2DP

Advanced Audio Distribution Profile
高级音频分发框架

ADC

Analog-to-Digital Converter
模拟数字转换器

ALPN

Application Layer Protocol Negotiation
应用层协议协商

AT

AT stands for “attention” .
AT 是 attention 的缩写。

AT command port

The port that is used to send AT commands and receive responses. More details are in the [AT port](#) introduction.

AT 命令端口

也称为 AT 命令口，用于发送 AT 命令和接收响应的端口。更多介绍请参考[AT 端口](#)。

AT log port

The port that is used to output log. More details are in the [AT port](#) introduction.

AT 日志端口

也称为 AT 日志口，用于输出 AT 日志的端口。更多介绍请参考[AT 端口](#)。

AT port

AT port is the general name of AT log port (that is used to output log) and AT command port (that is used to send AT commands and receive responses). Please refer to [Hardware Connection](#) for default AT port pins and [How to Set AT Port Pins](#) for how to customize them.

AT 端口

AT 端口是 AT 日志端口（用于输出日志）和 AT 命令端口（用于发送 AT 命令和接收响应）的总称。请参考[Hardware Connection](#) 了解默认的 AT 端口管脚，参考[How to Set AT Port Pins](#) 了解如何自定义 AT 端口管脚。

Bluetooth LE

Bluetooth Low Energy
低功耗蓝牙

BluFi

Wi-Fi network configuration function via Bluetooth channel
BluFi 是一款基于蓝牙通道的 Wi-Fi 网络配置功能

Command Mode

Default operating mode of AT. In the command mode, any character received by the AT command port will be treated as an AT command, and AT returns the command execution result to the AT command port. AT enters [Data Mode](#) from [Command Mode](#) in the following cases.

- After sending the [AT+CIPSEND](#) set command successfully and returns >.
- After sending the [AT+CIPSEND](#) execute command successfully and returns >.
- After sending the [AT+CIPSENDL](#) set command successfully and returns >.

- After sending the *AT+CIPSEND* set command successfully and returns >.
- After sending the *AT+SAVETRANSLINK* set command successfully and sending the *AT+RST* command and restart the module.
- After sending the *AT+BTSPSEND* execute command successfully and returns >.
- After sending the *AT+BLESPP* execute command successfully and returns >.

In the data mode, send the *+++* command, AT will exit from *Data Mode* and enter the *Command Mode*.

命令模式

AT 的默认工作模式。在命令模式下，AT 命令端口收到的任何字符都会被当作 AT 命令进行处理，同时 AT 会在命令端口回复命令执行结果。AT 在下列情况下，会从命令模式进入数据模式。

- 发送*AT+CIPSEND* 设置命令成功，回复 > 之后
- 发送*AT+CIPSEND* 执行命令成功，回复 > 之后
- 发送*AT+CIPSENDL* 设置命令成功，回复 > 之后
- 发送*AT+CIPSENDEX* 设置命令成功，回复 > 之后
- 发送*AT+SAVETRANSLINK* 设置命令成功，再发送 (*AT+RST*) 命令，模组重启之后
- 发送*AT+BTSPSEND* 执行命令成功，回复 > 之后
- 发送*AT+BLESPP* 执行命令成功，回复 > 之后

在数据模式下，发送*+++* 命令，会从数据模式退出，进入命令模式。

Data Mode

In the data mode, any character received by the AT command port will be treated as data (except for special *+++*) instead of the AT command, and these data will be sent to the opposite end without modification. AT enters *Data Mode* from *Command Mode* in the following cases.

- After sending the *AT+CIPSEND* set command successfully and returns >.
- After sending the *AT+CIPSEND* execute command successfully and returns >.
- After sending the *AT+CIPSENDL* set command successfully and returns >.
- After sending the *AT+CIPSENDEX* set command successfully and returns >.
- After sending the *AT+SAVETRANSLINK* set command successfully and sending the *AT+RST* command and restart the module.
- After sending the *AT+BTSPSEND* execute command successfully and returns >.
- After sending the *AT+BLESPP* execute command successfully and returns >.

In the data mode, send the *+++* command, AT will exit from *Data Mode* and enter the *Command Mode*.

数据模式

在数据模式下，AT 命令端口收到的任何字符都会被当作数据（除了特殊的*+++*），而不是 AT 命令，这些数据会无修改的发往对端。AT 在下列情况下，会从命令模式进入数据模式。

- 发送*AT+CIPSEND* 设置命令成功，回复 > 之后
- 发送*AT+CIPSEND* 执行命令成功，回复 > 之后
- 发送*AT+CIPSENDL* 设置命令成功，回复 > 之后
- 发送*AT+CIPSENDEX* 设置命令成功，回复 > 之后
- 发送*AT+SAVETRANSLINK* 设置命令成功，再发送*AT+RST* 命令，模组重启之后
- 发送*AT+BTSPSEND* 执行命令成功，回复 > 之后
- 发送*AT+BLESPP* 执行命令成功，回复 > 之后

在数据模式下，发送*+++* 命令，会从数据模式退出，进入命令模式。

DHCP

Dynamic Host Configuration Protocol
动态主机配置协议

DNS

Domain Name System
域名系统

DTIM

Delivery Traffic Indication Map
延迟传输指示映射

GATTC

Generic Attributes client
GATT 客户端

GATTS

Generic Attributes server

	GATT 服务器
HID	Human Interface Device 人机接口设备
I2C	Inter-Integrated Circuit 集成电路总线
ICMP	Internet Control Message Protocol 因特网控制报文协议
LwIP	A Lightweight TCP/IP stack 一个轻量级的 TCP/IP 协议栈
LWT	Last Will and Testament 遗嘱
MAC	Media Access Control MAC 地址
mDNS	Multicast Domain Name System 多播 DNS
manufacturing nvs	Manufacturing Non-Volatile Storage. <code>manufacturing nvs</code> stores all certificates, private keys, GATTS data, module information, Wi-Fi configurations, UART configurations, etc. The default values of these configurations are defined in <code>raw_data</code> . These configurations are finally built into the <code>mfg_nvs.bin</code> file and downloaded to the flash at the address defined in <code>at_customize.csv</code> . 一个适用于量产的 NVS。manufacturing nvs 中存储了 AT 固件默认所用到的所有证书、私钥、GATTS 数据、模组信息、Wi-Fi 配置、UART 配置等。这些配置信息，默认值在 <code>raw_data</code> 里，最终生成了 <code>mfg_nvs.bin</code> ，烧录到 <code>at_customize.csv</code> 中定义的位置。
MSB	Most Significant Bit 最高有效位
MTU	maximum transmission unit 最大传输单元
NVS	Non-Volatile Storage 非易失性存储器
Normal Transmission Mode	Default Transmission Mode In normal transmission mode, users can send AT commands. For examples, users can send MCU data received by AT command port to the opposite end of transmission by <i>AT+CIPSEND</i> ; and the data received from the opposite end of transmission will also be returned to MCU through AT command port with additional prompt: <i>+IPD</i> . During a normal transmission, if the connection breaks, ESP32 will give a prompt and will not attempt to reconnect. More details are in <i>Transmission Mode Shift Diagram</i> .
普通传输模式	默认传输模式 在普通传输模式下，用户可以发送 AT 命令。例如，用户可以通过 <i>AT+CIPSEND</i> 命令，发送 AT 命令口收到的 MCU 数据到传输对端。从传输对端收到的数据，会通过 AT 命令口返回给 MCU，同时会附带 <i>+IPD</i> 信息。 普通传输模式时，如果连接断开，ESP32 不会重连，并提示连接断开。 更多介绍请参考 <i>Transmission Mode Shift Diagram</i> 。
OWE	Opportunistic Wireless Encryption. OWE is a Wi-Fi standard which ensures that the communication between each pair of endpoints is protected from other endpoints.

More details are in [Wikipedia](#).

机会性无线加密。OWE 是一种 Wi-Fi 标准，它确保每对端点之间的通信受到保护，不受其他端点的影响。

更多介绍请参考 [维基百科](#)。

Passthrough Mode

Also called as “Passthrough Sending & Receiving Mode”.

In passthrough mode, users cannot send AT commands except special `+++` command. All MCU data received by AT command port will be sent to the opposite end of transmission without any modification; and the data received from the opposite end of transmission will also be returned to MCU through AT command port without any modification.

Passthrough mode can be classified into Network Passthrough Mode and Bluetooth Passthrough Mode according to the transmission method. Network Passthrough Mode can be further classified into TCP Passthrough Mode, UDP Passthrough Mode and SSL Passthrough Mode according to the supported connection methods. Each connection method supports both client and server roles.

During the Network passthrough transmission, if the connection breaks, ESP32 (as client) will keep trying to reconnect until `+++` is input to exit the passthrough transmission; ESP32 (as server) will shutdown the old connection and listen for a new connection until `+++` is input to exit the passthrough transmission.

More details are in [Transmission Mode Shift Diagram](#).

透传模式

也称为“透传发送接收模式”。

在透传模式下，除特殊命令`+++`外，用户无法发送其他 AT 命令。所有通过 AT 命令端口接收的 MCU 数据将被原样转发至传输对端；同时，从传输对端接收的数据也将通过 AT 命令端口无损回传至 MCU。

透传模式按照传输方式可分为 Network 透传模式和 Bluetooth 透传模式。Network 透传模式按照支持的连接方式可分为 TCP 透传模式、UDP 透传模式和 SSL 透传模式，每种连接方式均支持客户端和服务端两种角色。

Network 透传模式传输时，如果连接断开，ESP32 作为客户端时，会不停地尝试重连，此时单独输入`+++`退出透传，则停止重连；ESP32 作为服务器时，会关闭连接同时监听新的连接，此时单独输入`+++`退出透传。

更多介绍请参考[Transmission Mode Shift Diagram](#)。

Transmission Mode Shift Diagram

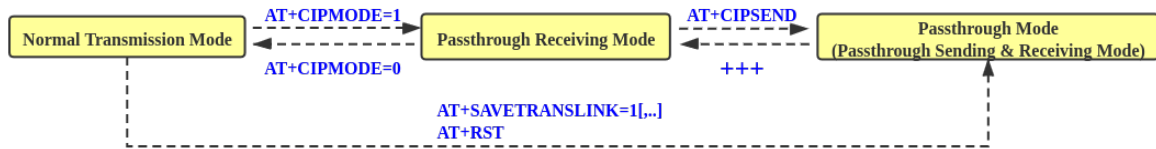


Fig. 1: Transmission Mode Shift Diagram

More details are in the following introduction.

- [Normal Transmission Mode](#) (普通传输模式)
- [Passthrough Receiving Mode](#) (透传接收模式)
- [Passthrough Mode](#) (透传模式)
- [AT+CIPMODE](#)
- [AT+CIPSEND](#)
- [+++](#)
- [AT+SAVETRANSLINK](#)

Passthrough Receiving Mode

The temporary mode between [Normal Transmission Mode](#) and [Passthrough Mode](#).

In passthrough receiving mode, AT cannot send any data to the opposite end of transmission; but the data received from the opposite end of transmission can be returned to MCU through AT command port without any modification. More details are in [Transmission Mode Shift Diagram](#).

透传接收模式

在普通传输模式和透传模式之间的一个临时模式。

在透传接收模式，AT 不能发送数据到传输对端；但 AT 可以收到来自传输对端的数据，通过 AT 命令口无修改地返回给 MCU。更多介绍请参考[Transmission Mode Shift Diagram](#)。

PBC

Push Button Configuration
按钮配置

PCI Authentication

Payment Card Industry Authentication. In ESP-AT project, it refers to all Wi-Fi authentication modes except OPEN and WEP.

PCI 认证，在 ESP-AT 工程中指的是除 OPEN 和 WEP 以外的 Wi-Fi 认证模式。

PKI

A public key infrastructure (PKI) is a set of roles, policies, hardware, software and procedures needed to create, manage, distribute, use, store and revoke digital certificates and manage public-key encryption.

More details are in [Public Key Infrastructure](#).

公开密钥基础建设。公开密钥基础建设 (PKI) 是一组由硬件、软件、参与者、管理政策与流程组成的基础架构，其目的在于创造、管理、分配、使用、存储以及撤销数字证书。

更多介绍请参考 [公开密钥基础建设](#)。

PLCP

Physical Layer Convergence Procedure

PLCP 协议，即物理层会聚协议

PMF

protected management frame

受保护的管理帧

PSK

Pre-shared Key

预共享密钥

PWM

Pulse-Width Modulation

脉冲宽度调制

QoS

Quality of Service

服务质量

RTC

Real Time Controller. A group of circuits in SoC that keeps working in any chip mode and at any time.

实时控制器，为 SoC 中的一组电路，在任何芯片模式下都能随时保持工作。

SMP

Security Manager Protocol

安全管理协议

SNI

Server Name Indication

服务器名称指示

SNTP

Simple Network Time Protocol

简单网络时间协议

SPI

Serial Peripheral Interface

串行外设接口

SPP

Serial Port Profile

SPP 协议，即串口协议

SSL

Secure Sockets Layer

SSL 协议，即安全套接字协议

system message

Data sent via AT command port to MCU. Each system message usually ends with `\r\n`. Detailed system message descriptions are available at [AT Messages](#).

系统消息

AT 命令口发往 MCU 的数据。每条系统消息通常以 `\r\n` 结尾。详细的系统消息说明见 [AT 消息](#)。

TLS

Transport Layer Security

TLS 协议，即传输层安全性协议

URC

Unsolicited Result Code

非请求结果码，一般为模组给 MCU 的串口返回

UTC

Coordinated Universal Time

协调世界时

UUID

universally unique identifier

通用唯一识别码

WEP

Wired-Equivalent Privacy

WEP 加密方式，即有线等效加密

WPA

Wi-Fi Protected Access

Wi-Fi 保护访问

WPA2

Wi-Fi Protected Access II

Wi-Fi 保护访问 II

WAPI

WLAN Authentication and Privacy Infrastructure. WAPI is a wireless security protocol independently developed in China and designated as a national standard (GB 15629.11). It provides authentication and encryption mechanisms for wireless networks, using a different approach from WPA/WPA2.

More details are in [Wikipedia](#).

无线局域网鉴别和保密基础结构。WAPI 是中国自主研发的无线安全协议，是中国国家标准 (GB 15629.11)。它为无线网络提供身份验证和加密机制，其实现方式不同于 WPA/WPA2。

更多介绍请参考 [维基百科](#)。

WPS

Wi-Fi Protected Setup

Wi-Fi 保护设置

Chapter 10

Disclaimer

Thank you for choosing to use [Solution/Firmware ESP-AT]. To ensure the safe, legal, and effective use of this solution (firmware), please carefully read the following Disclaimer. This Disclaimer aims to clarify the terms and precautions that users must follow when using this solution (firmware). By continuing to use this solution (firmware), you indicate your acceptance and agreement to comply with these terms.

10.1 Application Scenario Verification

This solution (firmware) is a general version. Despite extensive testing across various scenarios, there remains a risk that certain use cases may not be covered. Prior to making commercial production decisions based on this solution (firmware), please conduct thorough and comprehensive validation tailored to your specific application scenarios.

10.2 Compatibility

Given the rapid advancement of technology, the emergence of new technologies, solutions, or standards may lead to compatibility issues. Users are advised to conduct appropriate testing and validation based on their products and to stay informed about new version releases of this solution (firmware)

10.3 Server Upgrade Considerations

When upgrading servers, including but not limited to operating system upgrades, tool upgrades, and certificate upgrades, it is essential to conduct testing and validation prior to any formal upgrades to prevent connectivity issues arising from server changes.

10.4 Wireless Solution Issue Handling

Wireless solutions are susceptible to external factors and can present complex issue diagnostics. Users should plan remote OTA and remote issue information retrieval solutions in conjunction with their products and cloud platforms. Otherwise, users may need to provide on-site logs, data packets, and other necessary information for problem analysis.

10.5 Usage Restrictions

Users must not utilize this solution (firmware) for illegal activities or purposes that violate applicable laws and regulations. Users are responsible for ensuring that their use of this solution (this firmware) complies with all relevant laws and regulations. We accept no responsibility for any consequences arising from illegal use of this solution (firmware).

10.6 Limitation of Liability

We shall not be liable for any direct, indirect, incidental, special, or consequential damages (including but not limited to data loss, profit loss, business interruption, etc.) resulting from the use of this solution (firmware), even if we have been advised of the possibility of such damages.

10.7 Third-Party Software and Technology

This solution (firmware) may contain third-party software or technology. We assume no responsibility for the applicability and compliance of such third-party software or technology. Users should adhere to the licensing agreements and usage terms of third parties.

10.8 Data Protection

Users are solely responsible for the protection and backup of any data generated during the use of this solution (firmware). We accept no liability for any losses resulting from data loss or leakage.

10.9 Version Updates and Maintenance

This solution (firmware) will undergo periodic maintenance updates and version releases. Users are encouraged to monitor version status, and no separate notification will be provided.

10.10 Disclaimer Updates

The terms of this Disclaimer may be updated periodically without prior notice.

10.11 User Responsibility

Users are strongly encouraged to read and strictly adhere to the terms of this Disclaimer. Any risks arising from non-compliance shall be borne by the user.

Thank you for your understanding and cooperation!

Chapter 11

About

This is documentation of [ESP-AT](#), a solution developed by Espressif to quickly and easily interface with ESP32 products.

Espressif Wi-Fi and Bluetooth chipsets are often used as add-on modules to seamlessly integrate wireless connectivity features into new and existing products. In an effort to facilitate this and cut down on engineering costs, Espressif Systems has developed [AT firmware](#) as well as a rich set of [AT commands](#) that can be used to interface with Espressif products.



Fig. 1: ESP-AT Solution

The AT firmware allows for rapid integration by providing:

- In-built TCP/IP stack and data buffering
- Easy integration with resource-constrained host platforms
- Easy-to-parse command-response protocols
- Customized, user-defined AT commands

11.1 Other Espressif Connectivity Solutions

In addition to ESP-AT, Espressif also provides other solutions to meet different integration needs:

- **ESP-Hosted** is a software solution that enables Espressif series chips to act as wireless communication co-processors for external host systems. It allows host devices (Linux-based systems or microcontrollers, MCUs) to add Wi-Fi and Bluetooth/BLE capabilities via standard interfaces like SPI, SDIO, or UART. You can refer to [ESP-Hosted GitHub](#) for details.

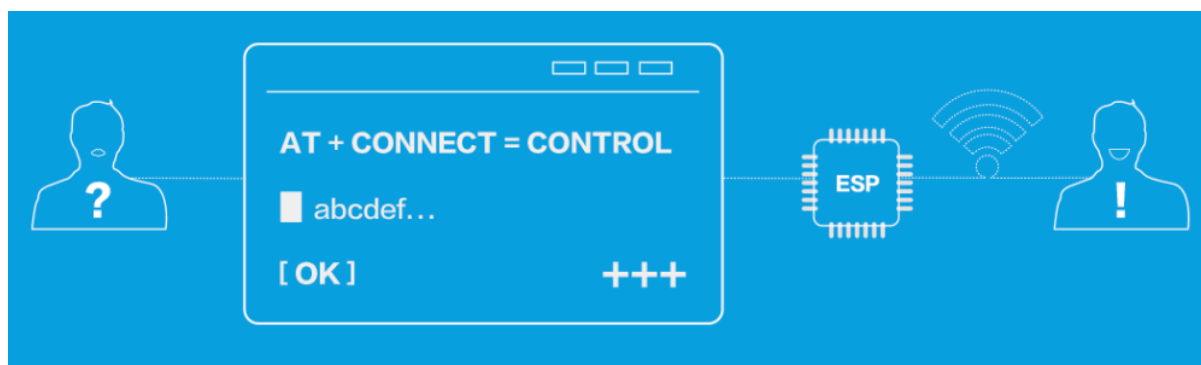


Fig. 2: ESP-AT Commands

- **ESP-IoT-Bridge Solution** focuses on connectivity and communication between various network interfaces in IoT application scenarios, such as SPI, SDIO, USB, Wi-Fi, Ethernet and other network interfaces. In this solution, the bridge device can not only provide net access for other devices, but also be a separate equipment to connect the remote server. You can refer to [ESP-IoT-Bridge GitHub](#) for details.

These solutions target different system architectures and requirements. For users who are considering alternatives to the AT firmware for integrating wireless connectivity, these options may provide a better fit depending on host platform capabilities and design goals.

- [genindex](#)

Index

A

A2DP, [485](#)
ADC, [485](#)
ALPN, [485](#)
AT, [485](#)
AT command port, [485](#)
AT log port, [485](#)
AT port, [485](#)
AT 命令端口, [485](#)
AT 日志端口, [485](#)
AT 端口, [485](#)
AT_BUFFER_ON_STACK_SIZE (C macro), [467](#)
at_get_data_len_fn_t (C++ type), [462](#)
at_get_mfg_params_storage_mode (C++ function), [465](#)
at_handle_result_code (C++ function), [459](#)
at_max (C macro), [461](#)
at_mfg_params_storage_mode_t (C++ enum), [467](#)
at_mfg_params_storage_mode_t::AT_PARAMS_IN_MFG_NVFS (C++ enumerator), [467](#)
at_mfg_params_storage_mode_t::AT_PARAMS_IN_PARTITION (C++ enumerator), [467](#)
at_mfg_params_storage_mode_t::AT_PARAMS_NONE (C++ enumerator), [467](#)
at_min (C macro), [461](#)
at_read_data_fn_t (C++ type), [462](#)
at_sleep_mode_t (C++ enum), [462](#)
at_sleep_mode_t::AT_DISABLE_SLEEP (C++ enumerator), [462](#)
at_sleep_mode_t::AT_LIGHT_SLEEP (C++ enumerator), [463](#)
at_sleep_mode_t::AT_MAX_MODEM_SLEEP (C++ enumerator), [463](#)
at_sleep_mode_t::AT_MIN_MODEM_SLEEP (C++ enumerator), [463](#)
at_sleep_mode_t::AT_SLEEP_MAX (C++ enumerator), [463](#)
at_str_is_null (C++ function), [459](#)
at_write_data_fn_t (C++ type), [462](#)

B

Bluetooth LE, [485](#)
BluFi, [485](#)

C

Command Mode, [485](#)

CONFIG_AT_LOG_DEFAULT_LEVEL (C macro), [467](#)

D

Data Mode, [486](#)
DHCP, [486](#)
DNS, [486](#)
DTIM, [486](#)

E

ESP_AT_CMD_ERROR_CMD_EXEC_FAIL (C macro), [462](#)
ESP_AT_CMD_ERROR_CMD_OP_ERROR (C macro), [462](#)
ESP_AT_CMD_ERROR_CMD_PROCESSING (C macro), [462](#)
ESP_AT_CMD_ERROR_CMD_UNSUPPORTED (C macro), [462](#)
ESP_AT_CMD_ERROR_NON_FINISH (C macro), [461](#)
ESP_AT_CMD_ERROR_NOT_FOUND_AT (C macro), [461](#)
ESP_AT_CMD_ERROR_OK (C macro), [461](#)
ESP_AT_CMD_ERROR_PARA_INVALID (C macro), [462](#)
ESP_AT_CMD_ERROR_PARA_LENGTH (C macro), [461](#)
ESP_AT_CMD_ERROR_PARA_NUM (C macro), [462](#)
ESP_AT_CMD_ERROR_PARA_PARSE_FAIL (C macro), [462](#)
ESP_AT_CMD_ERROR_PARA_TYPE (C macro), [461](#)
esp_at_cmd_struct (C++ struct), [459](#)
esp_at_cmd_struct::at_cmdName (C++ member), [459](#)
esp_at_cmd_struct::at_exeCmd (C++ member), [460](#)
esp_at_cmd_struct::at_queryCmd (C++ member), [459](#)
esp_at_cmd_struct::at_setupCmd (C++ member), [459](#)
esp_at_cmd_struct::at_testCmd (C++ member), [459](#)
esp_at_custom_ble_ops_regist (C++ function), [457](#)
esp_at_custom_ble_ops_struct (C++ struct), [460](#)
esp_at_custom_ble_ops_struct::connect_cb (C++ member), [460](#)

esp_at_custom_ble_ops_struct::disconnect_cb (C++ enumerator), 463
 (C++ member), 461
 esp_at_custom_ble_ops_struct::recv_data (C++ enumerator), 463
 (C++ member), 460
 esp_at_custom_cmd_array_regist (C++ function), 456
 esp_at_custom_cmd_line_terminator_get (C++ function), 458
 esp_at_custom_cmd_line_terminator_set (C++ function), 458
 esp_at_custom_net_ops_regist (C++ function), 457
 esp_at_custom_net_ops_struct (C++ struct), 460
 esp_at_custom_net_ops_struct::connect_cb (C++ enumerator), 463
 (C++ member), 460
 esp_at_custom_net_ops_struct::disconnect_cb (C++ enumerator), 463
 (C++ member), 460
 esp_at_custom_net_ops_struct::recv_data (C++ member), 460
 (C++ member), 460
 esp_at_custom_ops_regist (C++ function), 457
 esp_at_custom_ops_struct (C++ struct), 461
 esp_at_custom_ops_struct::pre_active_write_data_callback (C++ member), 461
 esp_at_custom_ops_struct::pre_deepsleep_callback (C++ member), 461
 esp_at_custom_ops_struct::pre_restart_callback (C++ member), 461
 esp_at_custom_ops_struct::pre_sleep_callback (C++ member), 461
 esp_at_custom_ops_struct::pre_wakeup_callback (C++ member), 461
 esp_at_custom_ops_struct::status_callback (C++ member), 461
 esp_at_custom_partition_find (C++ function), 458
 esp_at_device_ops_regist (C++ function), 457
 esp_at_device_ops_struct (C++ struct), 460
 esp_at_device_ops_struct::get_data_length (C++ member), 460
 esp_at_device_ops_struct::read_data (C++ member), 460
 esp_at_device_ops_struct::wait_write_complete (C++ member), 460
 esp_at_device_ops_struct::write_data (C++ member), 460
 esp_at_error_code (C++ enum), 463
 esp_at_error_code::ESP_AT_SUB_CMD_EXEC_FAIL (C++ enumerator), 464
 esp_at_error_code::ESP_AT_SUB_CMD_OP_ERROR (C++ enumerator), 464
 esp_at_error_code::ESP_AT_SUB_CMD_PROCESSING (C++ enumerator), 464
 esp_at_error_code::ESP_AT_SUB_COMMON_ERROR (C++ enumerator), 463
 esp_at_error_code::ESP_AT_SUB_NO_AT (C++ enumerator), 464
 esp_at_error_code::ESP_AT_SUB_NO_TERMINATOR (C++ enumerator), 463
 esp_at_error_code::ESP_AT_SUB_OK (C++ enumerator), 463
 esp_at_error_code::ESP_AT_SUB_PARA_INVALID (C++ enumerator), 463
 esp_at_error_code::ESP_AT_SUB_PARA_LENGTH_MISMATCH (C++ enumerator), 463
 esp_at_error_code::ESP_AT_SUB_PARA_NUM_MISMATCH (C++ enumerator), 463
 esp_at_error_code::ESP_AT_SUB_PARA_PARSE_FAIL (C++ enumerator), 463
 esp_at_error_code::ESP_AT_SUB_PARA_TYPE_MISMATCH (C++ enumerator), 463
 esp_at_error_code::ESP_AT_SUB_UNSUPPORT_CMD (C++ enumerator), 463
 ESP_AT_ERROR_NO (C macro), 461
 esp_at_fs_get_mount_point (C++ function), 466
 esp_at_fs_get_type (C++ function), 467
 esp_at_fs_mount (C++ function), 466
 esp_at_fs_unmount (C++ function), 466
 esp_at_get_addr_version (C++ function), 459
 esp_at_get_current_cmd_name (C++ function), 459
 esp_at_get_current_module_name (C++ function), 465
 esp_at_get_fs_info (C++ function), 466
 esp_at_get_module_id (C++ function), 465
 esp_at_get_module_name_by_id (C++ function), 465
 esp_at_get_para_as_digit (C++ function), 455
 esp_at_get_para_as_float (C++ function), 456
 esp_at_get_para_as_str (C++ function), 456
 esp_at_get_version (C++ function), 457
 ESP_AT_LOG_BUFFER_HEXDUMP (C macro), 467
 esp_at_log_write (C++ function), 465
 ESP_AT_LOGD (C macro), 467
 ESP_AT_LOGE (C macro), 467
 ESP_AT_LOGI (C macro), 467
 ESP_AT_LOGV (C macro), 467
 ESP_AT_LOGW (C macro), 467
 esp_at_main_preprocess (C++ function), 465
 esp_at_module (C++ enum), 463
 esp_at_module::ESP_AT_MODULE_NUM (C++ enumerator), 463
 esp_at_module_init (C++ function), 455
 esp_at_nvs_get_blob (C++ function), 466
 esp_at_nvs_get_str (C++ function), 466
 esp_at_nvs_set_blob (C++ function), 466
 esp_at_nvs_set_str (C++ function), 465
 esp_at_para_parse_result_type (C++ enum), 464
 esp_at_para_parse_result_type::ESP_AT_PARA_PARSE_FAIL (C++ enumerator), 464

- [esp_at_para_parse_result_type::ESP_AT_PARA_PARSE_RESULT_OK](#)
 (C++ enumerator), [464](#)
- [esp_at_para_parse_result_type::ESP_AT_PARA_PARSE_RESULT_OMITTED](#)
 (C++ enumerator), [464](#)
- [esp_at_port_active_write_data](#) (C++ function), [457](#)
- [esp_at_port_enter_specific](#) (C++ function), [458](#)
- [esp_at_port_exit_specific](#) (C++ function), [459](#)
- [esp_at_port_get_data_length](#) (C++ function), [458](#)
- [esp_at_port_read_data](#) (C++ function), [457](#)
- [esp_at_port_rcv_data_notify](#) (C++ function), [456](#)
- [esp_at_port_rcv_data_notify_from_isr](#)
 (C++ function), [456](#)
- [esp_at_port_specific_callback_t](#) (C++ type), [462](#)
- [ESP_AT_PORT_TX_WAIT_MS_MAX](#) (C macro), [467](#)
- [esp_at_port_wait_write_complete](#) (C++ function), [458](#)
- [esp_at_port_write_data](#) (C++ function), [457](#)
- [esp_at_ready_before](#) (C++ function), [465](#)
- [esp_at_response_result](#) (C++ function), [457](#)
- [esp_at_result_code_string_index](#) (C++ enum), [464](#)
- [esp_at_result_code_string_index::ESP_AT_RESULT_CODE_ERROR](#)
 (C++ enumerator), [464](#)
- [esp_at_result_code_string_index::ESP_AT_RESULT_CODE_FAIL](#)
 (C++ enumerator), [464](#)
- [esp_at_result_code_string_index::ESP_AT_RESULT_CODE_IGNORE](#)
 (C++ enumerator), [464](#)
- [esp_at_result_code_string_index::ESP_AT_RESULT_CODE_MAX](#)
 (C++ enumerator), [465](#)
- [esp_at_result_code_string_index::ESP_AT_RESULT_CODE_OK](#)
 (C++ enumerator), [464](#)
- [esp_at_result_code_string_index::ESP_AT_RESULT_CODE_OK_AND_INPUT_PROMPT](#)
 (C++ enumerator), [465](#)
- [esp_at_result_code_string_index::ESP_AT_RESULT_CODE_PROCESS_DONE](#)
 (C++ enumerator), [464](#)
- [esp_at_result_code_string_index::ESP_AT_RESULT_CODE_SEND_FAIL](#)
 (C++ enumerator), [464](#)
- [esp_at_result_code_string_index::ESP_AT_RESULT_CODE_SEND_OK](#)
 (C++ enumerator), [464](#)
- [esp_at_set_module_id](#) (C++ function), [465](#)
- [esp_at_set_module_id_by_str](#) (C++ function), [465](#)
- [esp_at_status_type](#) (C++ enum), [462](#)
- [esp_at_status_type::ESP_AT_STATUS_NORMAL](#)
 (C++ enumerator), [462](#)
- [esp_at_status_type::ESP_AT_STATUS_TRANSMIT](#)
 (C++ enumerator), [462](#)
- [esp_at_transmit_terminal](#) (C++ function), [456](#)
- [esp_at_transmit_terminal_from_isr](#)
 (C++ function), [456](#)
- [esp_at_yield_if_idle_timeout](#) (C++ function), [465](#)
- ## G
- [GATTC](#), [486](#)
- [GATTS](#), [486](#)
- ## H
- [HID](#), [487](#)
- ## I
- [I2C](#), [487](#)
- [ICMP](#), [487](#)
- ## L
- [LwIP](#), [487](#)
- [LWT](#), [487](#)
- ## M
- [MAC](#), [487](#)
- [manufacturing nvs](#), [487](#)
- [mDNS](#), [487](#)
- [MSB](#), [487](#)
- [MTU](#), [487](#)
- ## N
- [Normal Transmission Mode](#), [487](#)
- [NVS](#), [487](#)
- ## O
- [OWE](#), [487](#)
- ## P
- [Passthrough Mode](#), [488](#)
- [Passthrough Receiving Mode](#), [488](#)
- [PBC](#), [489](#)
- [PCI Authentication](#), [489](#)
- [PKI](#), [489](#)
- [PLCP](#), [489](#)
- [PME](#), [489](#)
- [PSK](#), [489](#)
- [PWM](#), [489](#)
- ## Q
- [QoS](#), [489](#)
- ## R
- [RTC](#), [489](#)
- ## S
- [SMP](#), [489](#)
- [SMT](#), [489](#)
- [SNTP](#), [489](#)
- [SPI](#), [489](#)
- [SPP](#), [489](#)
- [SSL](#), [489](#)
- [system message](#), [489](#)

T

TLS, [489](#)

Transmission Mode Shift Diagram, [488](#)

U

URC, [490](#)

UTC, [490](#)

UUID, [490](#)

W

WAPI, [490](#)

WEP, [490](#)

WPA, [490](#)

WPA2, [490](#)

WPS, [490](#)



命令模式, [486](#)



数据模式, [486](#)

普通传输模式, [487](#)



系统消息, [489](#)



透传接收模式, [488](#)

透传模式, [488](#)