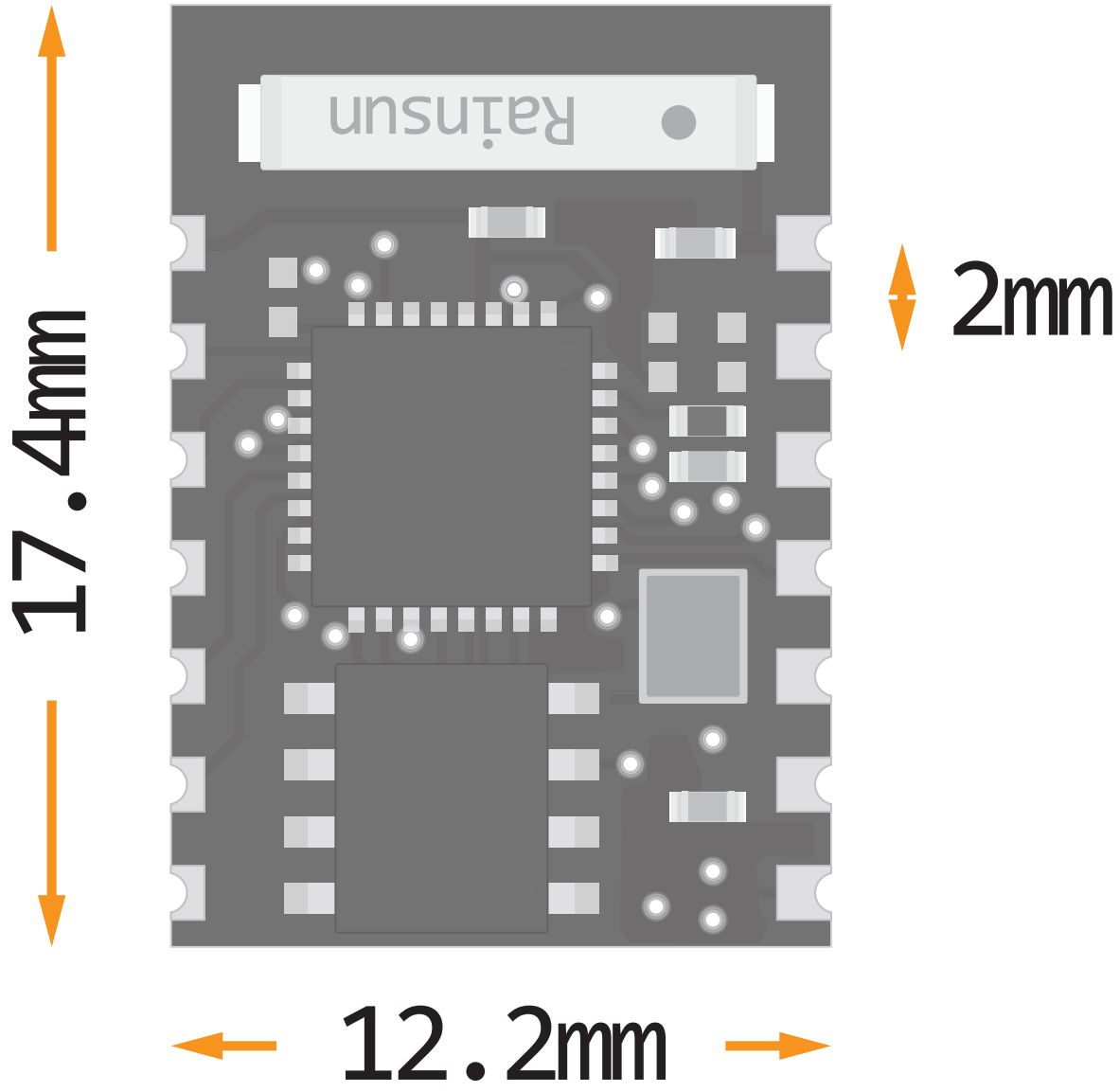
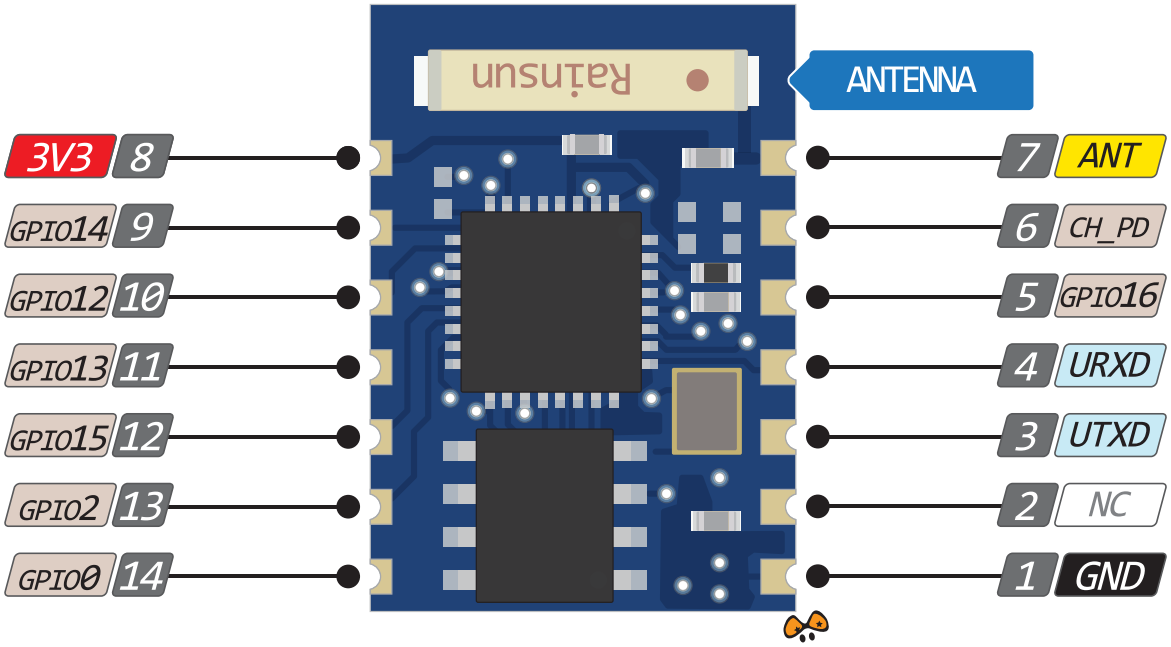
ESP8266

REFERENCE

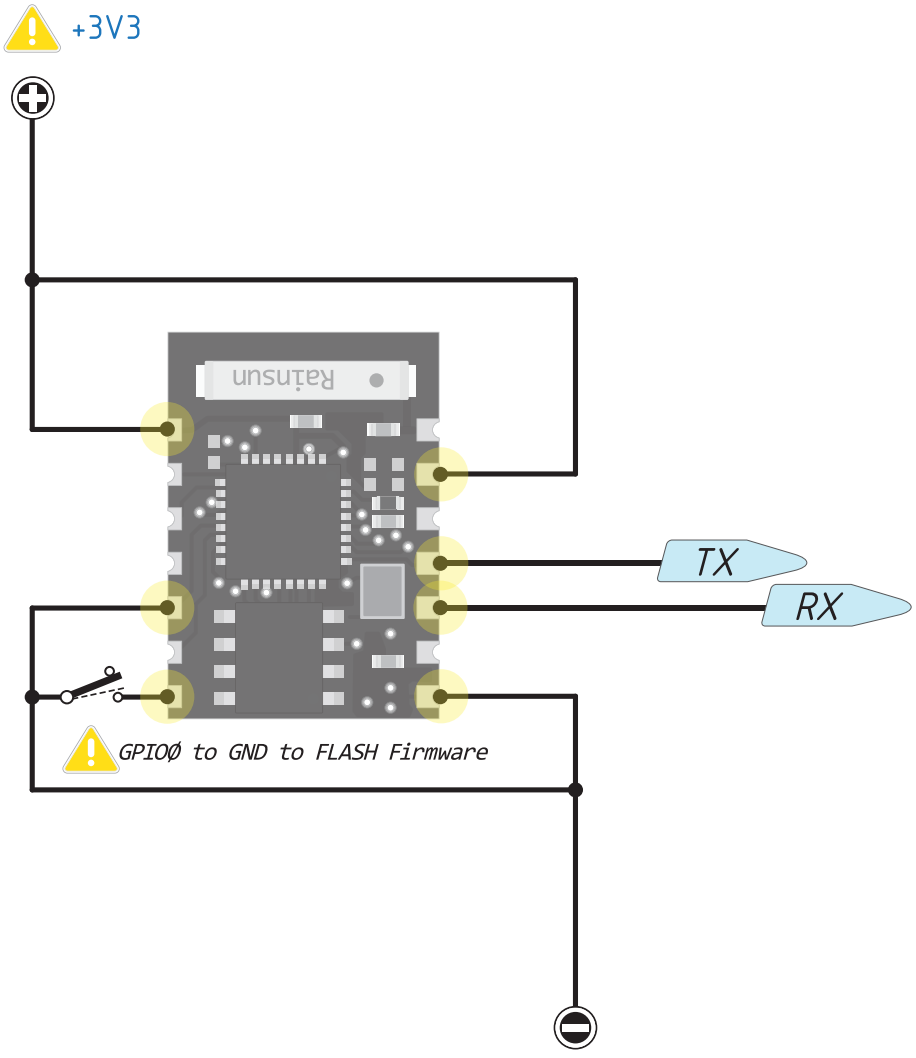
ESP8266 Dimensions



ESP8266 Pinout



ESP8266 Basic connect



AT Instruction Set
Overview

This is the documentation for ESP8266 AT command instruction set and usage. Instruction set is divided into: **Basic AT commands, WiFi function, TCP/IP commands.**

Version Info

Date	Version	Author	Changes
09 Dec 2014	1.0	Pighixxx	Draft
24 Jun 2015	1.1	Pighixxx	1st Revision

Disclaimer and Copyright Notice

Instruction Description

Each instruction set contains four types of AT commands.

Type	Format	Description
Test	**AT+<x>=?**	Query the Set command or internal parameters and its range values.
Query	**AT+<x>?**	Returns the current value of the parameter.
Set	**AT+<x>=<...>**	Set the value of user-defined parameters in commands and run.
Execute	**AT+<x>**	Runs commands with no user-defined parameters.

NOTE:

1. Not all AT instruction has four commands.
2. [] = default value, not required or may not appear
3. String values require double quotation marks, for example: **AT+CWSAP="ESP756190","21030826",1,4**
4. Baud rate = **115200**
5. AT instruction has to be capitalized and ends with **"\r\n"**

AT Instruction Listing

Instruction	Description
Basic	
AT	Test AT startup
AT+RST	Restart
AT+GMR	View version info
AT+GSLP	Enter deep-sleep mode
ATE	AT commands echo
AT+RESTORE	Factory Reset
AT+UART	UART configuration
AT+UART_CUR	UART current configuration
AT+UART_DEF	UART default configuration, save to flash
AT+SLEEP	Sleep mode
WiFi	
AT+CWMODE	WIFI mode(station/softAP/station+softAP)
AT+CWMODE_CUR	WIFI mode(station/softAP/station+softAP) Won't save to Flash
AT+CWMODE_DEF	WIFI mode(station/softAP/station+softAP) Save to Flash
AT+CWJAP	Connect to AP
AT+CWJAP_CUR	Connect to AP, Won't save to Flash
AT+CWJAP_DEF	Connect to AP, Save to Flash
AT+CWLAP	Lists available APs
AT+CWQAP	Disconnect from AP
AT+CWSAP	Set parameters under AP mode
AT+CWSAP_CUR	Set parameters under AP mode, Won't save to Flash
AT+CWSAP_DEF	Set parameters under AP mode, Save to Flash
AT+CWLIF	Get stations' ip which are connected to ESP8266 softAP
AT+CWDHCP	Enable/Disable DHCP
AT+CWDHCP_CUR	Enable/Disable DHCP, Won't save to Flash
AT+CWDHCP_DEF	Enable/Disable DHCP, Save to Flash
AT+CWAUTOCONN	Connect to AP automatically when power on
AT+CIPSTAMAC	Set mac address of ESP8266 station

Instruction	Description
AT+CIPSTAMAC_CUR	Set mac address of ESP8266 station Won't save to Flash
AT+CIPSTAMAC_DEF	Set mac address of ESP8266 station Save to Flash
AT+CIPAPMAC	Set mac address of ESP8266 softAP
AT+CIPAPMAC_CUR	Set mac address of ESP8266 softAP Won't save to Flash
AT+CIPAPMAC_DEF	Set mac address of ESP8266 softAP Save to Flash
AT+CIPSTA	Set ip address of ESP8266 station
AT+CIPSTA_CUR	Set ip address of ESP8266 station Won't save to Flash
AT+CIPSTA_DEF	Set ip address of ESP8266 station Save to Flash
AT+CIPAP	Set ip address of ESP8266 softAP
AT+CIPAP_CUR	Set ip address of ESP8266 softAP Won't save to Flash
AT+CIPAP_DEF	Set ip address of ESP8266 softAP Save to Flash
TCP/IP	
AT+CIPSTATUS	Get connection status
AT+CIPSTART	Establish TCP connection or register UDP port
AT+CIPSEND	Send data
AT+CIPSENDEX	Send data If <length> or "\0" is met, data will be sent
AT+CIPSENDBUF	Write data into TCP-send-buffer
AT+CIPBUFRESET	Reset segment ID count
AT+CIPBUFSTATUS	Check status of TCP-send-buffer
AT+CIPCHECKSEG	Check if a specific segment is sent or not
AT+CIPCLOSE	Close TCP/UDP connection
AT+CIFSR	Get local IP address
AT+CIPMUX	Set multiple connections mode
AT+CIPSERVER	Configure as server
AT+CIPMODE	Set transmission mode
AT+SAVETRANSLINK	Save transparent transmission link to Flash
AT+CIPSTO	Set timeout when ESP8266 runs as TCP server
AT+CIUPDATE	Force OTA(upgrade through network)

Instruction	Description
AT+PING	Function PING
+IPD	Data received from network

 Deprecated

Basic AT Instruction Set

Overview

Instruction	Description
AT	Test AT startup
AT+RST	Restart
AT+GMR	View version info
AT+GSLP	Enter deep-sleep mode
ATE	AT commands echo
AT+RESTORE	Factory Reset
AT+UART	UART configuration
AT+UART_CUR	UART current configuration
AT+UART_DEF	UART default configuration, save to flash
AT+SLEEP	Sleep mode

Deprecated

Instructions

AT – Test AT startup

The type of this command is "executed". It's used to test the setup function of your wireless WiFi module.

Instruction:	Response:
AT	**OK** _____ Param description: null

AT+RST – Restart Module

The type of this command is "executed". It's used to restart the module.

Instruction:	Response:
AT+RST	**OK** _____ Param description: null

AT+GMR – View version Info

This AT command is used to check the version of AT commands and SDK that you are using, the type of which is "executed".

Instruction:	Response:
AT+GMR	**<AT version info>** **<SDK version info>** **<compile time>** **OK** _____ Param description: <AT version info> Information about AT version <SDK version info> Information about SDK version <compile time> Time of the bin was compiled

AT+GSLP – Enter deep-sleep mode

*This command is used to invoke the deep-sleep mode of the module, the type of which is "set". A minor adjustment has to be made before the module enter this deep sleep mode, i.e., connect **XPD_DCDC** with **EXT_RSTB** via **0R**.*

Instruction:	Response:
AT+GLSP = <time>	**OK** _____ Param description: <time> The time unit of < time > is ms. ESP8266 will wake up after X ms and then enter the deep sleep mode.

ATE – AT commands echo

This command ATE is an AT trigger command echo. It means that entered commands can be echoed back to the sender when ATE command is used. Two parameters are possible. The command returns "OK" in normal cases and "ERROR" when a parameter other than 0 or 1 was specified.

Instruction:	Response:
ATE<x>	**OK** _____ Param description: <x> 0: – Switch echo off 1: – Switch echo on

AT+RESTORE – Factory reset

This command is used to reset all parameters saved in flash (according to appendix), restore the factory default settings of the module. The chip will be restarted when this command is executed.

Instruction:	Response:
AT+RESTORE	**OK** _____ Param description: null

AT+UART – UART Configuration

This command sets the UART configuration and writes the new configuration to the flash. It is stored as the default parameter and will also be used as the default baudrate henceforth. ***[THIS API IS DEPRECATED.]***

Instruction:

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

Response:

OK

Param description:

<baudrate>
Baudrate range:
110 to 115200*40 (4.608 Mega)

<databits>
5 – 5 bits data
6 – 6 bits data
7 – 7 bits data
8 – 8 bits data

<stopbits>
1 – 1 bit stop bit
2 – 1.5 stop bit
3 – 2 bit stop bit

<parity>
0 – NONE
1 – ODD
2 – EVEN

<flow control>
0 – Disable flow control
1 – Enable RTS
2 – Enable CTS
3 – Enable both CTS and RTS

Example:

AT+UART=115200,8,1,0,3

Note:

1. This configuration will also store the baud rate as the default rate in the user parameter area in the Flash for boot up.
2. **Flow control needs hardware support:**
   MTCK is UART0 CTS and MTDO is UART0 RTS

Deprecated

AT+UART_CUR – Current UART Configuration

This command sets the current UART configuration; it does not write to the flash device. Hence there is no change in the default baudrate.

Instruction:

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

Response:

OK

Param description:

<baudrate>
Baudrate range:
110 to 115200*40 (4.608 Mega)

<databits>
5 – 5 bits data
6 – 6 bits data
7 – 7 bits data
8 – 8 bits data

<stopbits>
1 – 1 bit stop bit
2 – 1.5 stop bit
3 – 2 bit stop bit

<parity>
0 – NONE
1 – ODD
2 – EVEN

<flow control>
0 – Disable flow control
1 – Enable RTS
2 – Enable CTS
3 – Enable both CTS and RTS

Example:

AT+UART=115200,8,1,0,3

Note:

1. This configuration will NOT be stored in the flash unlike the AT+UART command.
2. **Flow control needs hardware support:**
   MTCK is UART0 CTS and MTDO is UART0 RTS

AT+UART_DEF – Default UART Configuration

This command sets the UART configuration and save it to flash. It is stored as the default parameter and will also be used as the default baudrate henceforth.

Instruction:

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

Response:

OK

Param description:

<baudrate>
Baudrate range:
110 to 115200*40 (4.608 Mega)

<databits>
5 – 5 bits data
6 – 6 bits data
7 – 7 bits data
8 – 8 bits data

<stopbits>
1 – 1 bit stop bit
2 – 1.5 stop bit
3 – 2 bit stop bit

<parity>
0 – NONE
1 – ODD
2 – EVEN

<flow control>
0 – Disable flow control
1 – Enable RTS
2 – Enable CTS
3 – Enable both CTS and RTS

Example:

AT+UART=115200,8,1,0,3

Note:

1. This configuration will be stored in the flash user parameter area.
2. **Flow control needs hardware support:**
   MTCK is UART0 CTS and MTDO is UART0 RTS

AT+SLEEP – sleep mode

This command sets ESP8266 sleep mode. It can only be used in station mode, default to be modem sleep mode.

Type: **query** Function: **Query ESP8266's current sleep mode.** Instruction: **AT+SLEEP?**	Response: **+SLEEP:<sleep mode>** **OK** Param description: <sleep mode> 0 – Disable sleep mode 1 – Light-sleep mode 2 – Modem-sleep mode
Type: **set** Function: **Set ESP8266 sleep mode.** Instruction: **AT+SLEEP=<sleep mode>**	Response: **OK** Param description: <sleep mode> 0 – Disable sleep mode 1 – Light-sleep mode 2 – Modem-sleep mode

WiFi Functions
Overview

Instruction	Description
AT+CWMODE	WIFI mode(station/softAP/station+softAP)
AT+CWMODE_CUR	WIFI mode(station/softAP/station+softAP) Won't save to Flash
AT+CWMODE_DEF	WIFI mode(station/softAP/station+softAP) Save to Flash
AT+CWJAP	Connect to AP
AT+CWJAP_CUR	Connect to AP, Won't save to Flash
AT+CWJAP_DEF	Connect to AP, Save to Flash
AT+CWLAP	Lists available APs
AT+CWQAP	Disconnect from AP
AT+CWSAP	Set parameters under AP mode
AT+CWSAP_CUR	Set parameters under AP mode, Won't save to Flash
AT+CWSAP_DEF	Set parameters under AP mode, Save to Flash
AT+CWLIF	Get stations' ip which are connected to ESP8266 softAP
AT+CWDHCP	Enable/Disable DHCP
AT+CWDHCP_CUR	Enable/Disable DHCP, Won't save to Flash
AT+CWDHCP_DEF	Enable/Disable DHCP, Save to Flash
AT+CWAUTOCONN	Connect to AP automatically when power on
AT+CIPSTAMAC	Set mac address of ESP8266 station
AT +CIPSTAMAC_CUR	Set mac address of ESP8266 station Won't save to Flash
AT +CIPSTAMAC_DEF	Set mac address of ESP8266 station Save to Flash
AT+CIPAPMAC	Set mac address of ESP8266 softAP
AT +CIPAPMAC_CUR	Set mac address of ESP8266 softAP Won't save to Flash
AT +CIPAPMAC_DEF	Set mac address of ESP8266 softAP Save to Flash
AT+CIPSTA	Set ip address of ESP8266 station
AT+CIPSTA_CUR	Set ip address of ESP8266 station Won't save to Flash
AT+CIPSTA_DEF	Set ip address of ESP8266 station Save to Flash
AT+CIPAP	Set ip address of ESP8266 softAP

Instruction	Description
AT+CIPAP_CUR	Set ip address of ESP8266 softAP Won’t save to Flash
AT+CIPAP_DEF	Set ip address of ESP8266 softAP Save to Flash
AT+CWSTARTSMART	Start SmartConfig
AT+CWSTOPSMART	Stop SmartConfig

 Deprecated

Instructions

AT+CWMODE – WiFi mode(station/softAP/station+softAP) *The function of this AT command is to get the value scope of WiFi mode, including station mode, softAP mode, and station+softAP mode, enquiry about the information of WiFi mode, or set the WiFi mode.*	
Type: **test** Function: **Get value scope of WiFi mode.** Instruction: **AT+CWMODE=?**	Response: **+CWMODE:(value scope of <mode>)** **OK** Param description: <mode> 1 – Station Mode 2 – AP Mode 3 – AP + Station Mode
Type: **query** Function: **Query ESP8266's current wifi mode.** Instruction: **AT+CWMODE?**	Response: **+CWMODE:<mode>** **OK** Param description: <mode> 1 – Station Mode 2 – AP Mode 3 – AP + Station Mode
Type: **set** Function: **Set ESP8266 wifi mode.** Instruction: **AT+CWMODE=<mode>**	Response: **OK** Param description: <mode> 1 – Station Mode 2 – AP Mode 3 – AP + Station Mode
Note: 1. This configuration will be stored in the flash user parameter area. 2. It won't be erased even when the power is off and restarted.	

Deprecated

AT+CWMODE_CUR – Current WiFi mode

There are three WiFi working modes: Station mode, softAP mode, and the co-existence of Station mode and softAP mode. This command is used to acquire the existing WiFi mode, or to set a customized WiFi mode.

Type: **test** Function: **Get value scope of WiFi mode.** Instruction: **AT+CWMODE_CUR=?**	Response: **+CWMODE_CUR:(value scope of <mode>)** **OK** Param description: <mode> 1 – Station Mode 2 – AP Mode 3 – AP + Station Mode
Type: **query** Function: **Query ESP8266's current wifi mode.** Instruction: **AT+CWMODE_CUR?**	Response: **+CWMODE_CUR:<mode>** **OK** Param description: <mode> 1 – Station Mode 2 – AP Mode 3 – AP + Station Mode
Type: **set** Function: **Set ESP8266 wifi mode.** Instruction: **AT+CWMODE_CUR=<mode>**	Response: **OK** Param description: <mode> 1 – Station Mode 2 – AP Mode 3 – AP + Station Mode
Note: 1. This configuration will not store in Flash.	

AT+CWMODE_DEF – Default WiFi mode

There are three WiFi working modes: Station mode, softAP mode, and the co-existence of Station mode and softAP mode. This command is used to acquire the existing WiFi mode, or to set a customized WiFi mode.

Type: **test** Function: **Get value scope of WiFi mode.** Instruction: **AT+CWMODE_DEF=?**	Response: **+CWMODE_DEF:(value scope of <mode>)** **OK** Param description: <mode> 1 – Station Mode 2 – AP Mode 3 – AP + Station Mode
Type: **query** Function: **Query ESP8266's current wifi mode.** Instruction: **AT+CWMODE_DEF?**	Response: **+CWMODE_DEF:<mode>** **OK** Param description: <mode> 1 – Station Mode 2 – AP Mode 3 – AP + Station Mode
Type: **set** Function: **Set ESP8266 wifi mode.** Instruction: **AT+CWMODE_DEF=<mode>**	Response: **OK** Param description: <mode> 1 – Station Mode 2 – AP Mode 3 – AP + Station Mode
Note: 1. This configuration will be stored in the flash user parameter area.	

AT+CWJAP – Connect to AP

Please use AT+CWJAP_CUR or AT+CWJAP_DEF instead.

Type: **test** Function: **Query AP's info which is connect by ESP8266.** Instruction: **AT+CWJAP?**	Response: **+CWJAP:<ssid>** **OK** Param description: <ssid> string, AP's SSID
Type: **set** Function: **Set AP's info which will be connect by ESP8266.** Instruction: **AT+CWJAP=<ssid>,<pwd>[,<bssid>]**	Response: **OK** **ERROR** Param description: <ssid> string, AP's SSID <pwd> string, MAX: 64bytes <bssid> string, AP's bssid(MAC address), for several APs may have the same SSID

Example:

AT+CWJAP ="abc", "0123456789"
If SSID is "ab/,c" and password is "0123456789"/"

AT+CWJAP =“ab///,c", "0123456789/"//"
If several APs have the same SSID as “abc”,target AP can be found by bssid: **AT+CWJAP =“abc”,“0123456789”,”ca:d7:19:d8:a6:44”**

Note:

1. This configuration will be stored in the flash user parameter area.
2. This command needs station mode enable. Escape character syntax is needed if "SSID" or "password" contains any special characters (’,’、’"’and’/’)

Deprecated

AT+CWJAP_CUR – Connect to AP

Connect to AP, for current.

Type: **test**

Function:

Query AP's info which is connect by ESP8266.

Instruction:

AT+CWJAP_CUR?

Response:

+CWJAP_CUR:<ssid>
OK

Param description:

<ssid>
string, AP's SSID

Type: **set**

Function:

Set AP's info which will be connect by ESP8266.

Instruction:

AT+CWJAP_CUR=<ssid>,<pwd>[,<bssid>]

Response:

OK
ERROR

Param description:

<ssid>
string, AP's SSID
<pwd>
string, MAX: 64bytes
<bssid>
string, AP's bssid(MAC address), for several APs may have the same SSID

Example:

AT+CWJAP_CUR ="abc", "0123456789"
If SSID is "ab/,c" and password is "0123456789"/"

AT+CWJAP_CUR ="ab///,c", "0123456789/"//"
If several APs have the same SSID as "abc",target AP can be found by bssid: **AT+CWJAP_CUR ="abc","0123456789","ca:d7:19:d8:a6:44"**

Note:

1. This configuration will not store in Flash.
2. This command needs station mode enable. Escape character syntax is needed if "SSID" or "password" contains any special characters (',' '"'and'/')

AT+CWJAP_DEF – Connect to AP

Connect to AP, save as default.

Type: **test**

Function:

Query AP's info which is connect by ESP8266.

Instruction:

AT+CWJAP_DEF?

Response:

+CWJAP_DEF:<ssid>
OK

Param description:

<ssid>
string, AP's SSID

Type: **set**

Function:

Set AP's info which will be connect by ESP8266.

Instruction:

AT+CWJAP_DEF=<ssid>,<pwd>[,<bssid>]

Response:

OK
ERROR

Param description:

<ssid>
string, AP's SSID
<pwd>
string, MAX: 64bytes
<bssid>
string, AP's bssid(MAC address), for several APs may have the same SSID

Example:

AT+CWJAP_DEF ="abc", "0123456789"
If SSID is "ab/,c" and password is "0123456789"/"

AT+CWJAP_DEF ="ab///,c", "0123456789/"//"
If several APs have the same SSID as "abc",target AP can be found by bssid: **AT+CWJAP_DEF ="abc","0123456789","ca:d7:19:d8:a6:44"**

Note:

1. This configuration will store in Flash system parameter area.
2. This command needs station mode enable. Escape character syntax is needed if "SSID" or "password" contains any special characters (',' '"'and'/')

AT+CWLAP

List available APs

Type: **set** Function: **Search available APs with specific conditions.** Instruction: **AT+ CWLAP = <ssid>,<mac>,<ch>**	Response: **+CWLAP:<ecn>,<ssid>,<rssi>,<mac>** **OK** **ERROR** Param description: <ssid> string, AP's SSID
Type: **execute** Function: **Lists all available APs.** Instruction: **AT+CWLAP**	Response: **+CWLAP:<ecn>,<ssid>,<rssi>,<mac>** **OK** **ERROR** Param description: <ecn> 0 – OPEN 1 – WEP 2 – WPA_PSK 3 – WPA2_PSK 4 – WPA_WPA2_PSK <ssid> string, AP's SSID <rssi> signal strength <mac> string, MAC address

Example:

AT+CWLAP
List of all available AP's detected by ESP8266

AT+CWLAP="wifi","ca:d7:19:d8:a6:44",6
Find AP with specific SSID and MAC at specific channel.

AT+CWLAP="wifi"
Find AP with specific SSID

AT+CWQAP

Disconnect from AP

Type: **test** Function: **Only for test.** Instruction: **AT+CWQAP=?**	Response: **OK** Param description: null
Type: **execute** Function: **Disconnect from AP.** Instruction: **AT+CWQAP**	Response: **OK** Param description: *null*

AT+CWSAP – Configuration of softAP mode

Please use AT+CWSAP_CUR or AT+CWSAP_DEF instead.

Type: **query** Function: **Query configuration of softAP mode.** Instruction: **AT+ CWSAP?**	Response: **+CWSAP:<ssid>,<pwd>,<chl>,<ecn>** **OK** **ERROR** Param description: *The same as below*
Type: **set** Function: **Set configuration of softAP mode.** Instruction: **AT+CWSAP=<ssid>,<pwd>,<chl>,<ecn>**	Response: **OK** **ERROR** Param description: <ecn> 0 – OPEN 1 – WEP 2 – WPA_PSK 3 – WPA2_PSK 4 – WPA_WPA2_PSK <ssid> string, AP's SSID <pwd> string, MAX: 64bytes <chl> channel ID

Example:

AT+CWSAP="ESP8266","1234567890",5,3

Note:

1. This CMD is only available when softAP mode enable ESP8266 softAP don't support WEP.

Deprecated

AT+CWSAP_CUR – Current config of softAP mode

Set configuration of softAP mode, won't save to Flash.

Type: **query** Function: **Query configuration of softAP mode.** Instruction: **AT+ CWSAP_CUR?**	Response: **+CWSAP_CUR:<ssid>,<pwd>,<chl>,<ecn>** **OK** **ERROR** Param description: *The same as below*
Type: **set** Function: **Set configuration of softAP mode.** Instruction: **AT +CWSAP_CUR=<ssid>,<pwd>,<chl>,<ecn>**	Response: **OK** **ERROR** Param description: <ecn> 0 – OPEN 1 – WEP 2 – WPA_PSK 3 – WPA2_PSK 4 – WPA_WPA2_PSK <ssid> string, AP's SSID <pwd> string, MAX: 64bytes <chl> channel ID

Example:

AT+CWSAP_CUR="ESP8266","1234567890",5,3

Note:

1. This configuration will not store in Flash.
2. This CMD is only available when softAP mode enable ESP8266 softAP don't support WEP.

AT+CWSAP_DEF – Default config of softAP mode

Set configuration of softAP mode, save to Flash.

Type: **query**

Function:

Query configuration of softAP mode.

Instruction:

AT+ CWSAP_DEF?

Response:

+CWSAP_DEF:<ssid>,<pwd>,<chl>,<ecn>
OK
ERROR

Param description:

The same as below

Type: **set**

Function:

Set configuration of softAP mode.

Instruction:

AT +CWSAP_DEF=<ssid>,<pwd>,<chl>,<ecn>

Response:

OK
ERROR

Param description:

<ecn>
0 – OPEN
1 – WEP
2 – WPA_PSK
3 – WPA2_PSK
4 – WPA_WPA2_PSK
<ssid>
string, AP's SSID
<pwd>
string, MAX: 64bytes
<chl>
channel ID

Example:

AT+CWSAP_DEF="ESP8266","1234567890",5,3

Note:

1. This configuration will store in Flash system parameter area.
2. This CMD is only available when softAP mode enable ESP8266 softAP don't support WEP.

AT+CWLIF — ip of stations which are connected to ESP8266 softAP

This command is used to get the IP of stations that are connected to ESP8266 softAP.

Type: **execute**

Function:

Get ip of stations which are connected to ESP8266 softAP.

Instruction:

AT+CWLIF

Response:

<ip addr>,<mac>
OK

Param description:

<ip addr>
ip address of stations which are connected to ESP8266 softAP

<mac>
mac address of stations which are connected to ESP8266 softAP

Note:
1. This command only available when ESP8266 softAP DHCP enable.

AT+CWDHCP — Enable/Disable DHCP

Please use AT+CWDHCP_CUR or AT+CWDHCP_DEF instead.

Type: **query** Function: **Query if DHCP is enabled** Instruction: **AT+CWDHCP?**	Response: **Bit0** **Bit1** Param description: bit0 0 – softap DHCP disabled 1 – softap DHCP enabled bit1 0 – station DHCP disabled 1 – station DHCP enabled
Type: **set** Function: **Enable/Disable DHCP.** Instruction: **AT+CWDHCP=<mode>,<en>**	Response: **OK** Param description: <mode> 0 – Set ESP8266 softAP 1 – Set ESP8266 station 2 – both softAP and station <en> 0 – disable DHCP 1 – enable DHCP

Note:

1. This configuration will store in Flash user parameter area.
2. This configuration interact with static IP related AT commands(AT +CIPSTA related and AT+CIPAP related): If enable DHCP, static IP will be disabled; If enable static IP, DHCP will be disabled; This will depends on the last configuration.

Deprecated

AT+CWDHCP_CUR – Enable/Disable DHCP

Enable/Disable DHCP, won't save to Flash.

Type: **query** Function: **Query if DHCP is enabled** Instruction: **AT+CWDHCP_CUR?**	Response: **Bit0** **Bit1** Param description: bit0 0 – softap DHCP disabled 1 – softap DHCP enabled bit1 0 – station DHCP disabled 1 – station DHCP enabled
Type: **set** Function: **Enable/Disable DHCP.** Instruction: **AT+CWDHCP_CUR=<mode>,<en>**	Response: **OK** Param description: <mode> 0 – Set ESP8266 softAP 1 – Set ESP8266 station 2 – both softAP and station <en> 0 – disable DHCP 1 – enable DHCP

Note:

1. This configuration will not store in Flash.
2. This configuration interact with static IP related AT commands(AT +CIPSTA related and AT+CIPAP related): If enable DHCP, static IP will be disabled; If enable static IP, DHCP will be disabled; This will depends on the last configuration.

AT+CWDHCP_DEF – Enable/Disable DHCP

Enable/Disable DHCP and save to Flash.

Type: **query** Function: **Query if DHCP is enabled** Instruction: **AT+CWDHCP_DEF?**	Response: **Bit0** **Bit1** Param description: <mode> 0 – softap DHCP disabled 1 – softap DHCP enabled <en> 0 – station DHCP disabled 1 – station DHCP enabled
Type: **set** Function: **Enable/Disable DHCP.** Instruction: **AT+CWDHCP_DEF=<mode>,<en>**	Response: **OK** Param description: <mode> 0 – Set ESP8266 softAP 1 – Set ESP8266 station 2 – both softAP and station <en> 0 – disable DHCP 1 – enable DHCP

Note:

1. This configuration will store in Flash user parameter area.
2. This configuration interact with static IP related AT commands(AT +CIPSTA related and AT+CIPAP related): If enable DHCP, static IP will be disabled; If enable static IP, DHCP will be disabled; This will depends on the last configuration.

AT+CWAUTOCONN – Auto connect to AP or not

Connect to AP automatically or not.

Type: **set** Function: **Set mac address of ESP8266 station.** Instruction: **AT+CWAUTOCONN=<enable>**	Response: **OK** Param description: <enable> 0 – do not auto connect to AP when power on. 1 – connect to AP automatically when power on

Note:

1. This configuration will store in Flash user parameter area.
2. Default is enable, ESP8266 station will connect to AP automatically when power on.

AT+CIPSTAMAC – Set mac address of station

Use AT+CIPSTAMAC_CUR or AT+CIPSTAMAC_DEF instead.

Type: **query** Function: **Get mac address of ESP8266 station.** Instruction: **AT+CIPSTAMAC?**	Response: **+CIPSTAMAC:<mac>** **OK** Param description: <mac> string, mac address of station
Type: **set** Function: **Set mac address of ESP8266 station.** Instruction: **AT+CIPSTAMAC=<mac>**	Response: **OK** Param description: <mac> string, mac address of station

Example:

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

Note:

1. This configuration will store in Flash user parameter area.

Deprecated

AT+CIPSTAMAC_CUR – Set mac address of station

Set mac address of ESP8266 station, won't save to Flash.

Type: **query** Function: **Get mac address of ESP8266 station.** Instruction: **AT+CIPSTAMAC_CUR?**	Response: **+CIPSTAMAC_CUR:<mac>** **OK** Param description: <mac> string, mac address of station
Type: **set** Function: **Set mac address of ESP8266 station.** Instruction: **AT+CIPSTAMAC_CUR=<mac>**	Response: **OK** Param description: <mac> string, mac address of station

Example:

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

Note:

1. This configuration will not store in Flash.

AT+CIPSTAMAC_DEF – Set mac address of station

Set mac address of ESP8266 station, save to Flash.

Type: **query** Function: **Get mac address of ESP8266 station.** Instruction: **AT+CIPSTAMAC_DEF?**	Response: **+CIPSTAMAC_DEF:<mac>** **OK** Param description: <mac> string, mac address of station
Type: **set** Function: **Set mac address of ESP8266 station.** Instruction: **AT+CIPSTAMAC_DEF=<mac>**	Response: **OK** Param description: <mac> string, mac address of station

Example:

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

Note:

1. This configuration will store in Flash user parameter area.

AT+CIPAPMAC — Set mac address of softAP

Use AT+CIPAPMAC_CUR or AT+CIPAPMAC_DEF instead.

Type: **query** Function: **Get mac address of ESP8266 softAP.** Instruction: **AT+CIPAPMAC?**	Response: **+CIPAPMAC:<mac>** **OK** Param description: <mac> string, mac address of softAP
Type: **set** Function: **Set mac address of ESP8266 softAP.** Instruction: **AT+CIPAPMAC=<mac>**	Response: **OK** Param description: <mac> string, mac address of softAP
Example: **AT+CIPAPMAC="1a:fe:36:97:d5:7b"**	
Note: 1. This configuration will store in Flash user parameter area.	

Deprecated

AT+CIPAPMAC_CUR — Set mac address of softAP

Set mac addr of ESP8266 softAP, won't save to Flash.

Type: **query** Function: **Get mac address of ESP8266 softAP.** Instruction: **AT+CIPAPMAC_CUR?**	Response: **+CIPAPMAC_CUR:<mac>** **OK** Param description: <mac> string, mac address of softAP
Type: **set** Function: **Set mac address of ESP8266 softAP.** Instruction: **AT+CIPAPMAC_CUR=<mac>**	Response: **OK** Param description: <mac> string, mac address of softAP

Example:

AT+CIPAPMAC_CUR="1a:fe:36:97:d5:7b"

Note:

1. This configuration will not store in Flash.

AT+CIPAPMAC_DEF — Set mac address of softAP

Set mac address of ESP8266 softAP, save to Flash.

Type: **query** Function: **Get mac address of ESP8266 softAP.** Instruction: **AT+CIPAPMAC_DEF?**	Response: **+CIPAPMAC_DEF:<mac>** **OK** Param description: <mac> string, mac address of softAP
Type: **set** Function: **Set mac address of ESP8266 softAP.** Instruction: **AT+CIPAPMAC_DEF=<mac>**	Response: **OK** Param description: <mac> string, mac address of softAP

Example:

AT+CIPAPMAC_DEF="1a:fe:36:97:d5:7b"

Note:

1. This configuration will store in Flash user parameter area.

AT+CIPSTA — Set ip address of ESP8266 station

Please use AT+CIPSTA_CUR or AT+CIPSTA_DEF instead.

Type: **query** Function: **Get ip address of ESP8266 station.** Instruction: **AT+CIPSTA?**	Response: **+CIPSTA:<ip>** **OK** Param description: <ip> string, ip address of station
Type: **set** Function: **Set ip address of ESP8266 station.** Instruction: **AT+CIPSTA=<ip>[,<gateway>,<netmask>]**	Response: **OK** Param description: <ip> string, ip address of station <gateway> gateway <netmask> netmask

Example:

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

Note:

1. This configuration will store in Flash user parameter area.
2. This configuration interacts with DHCP related AT commands(AT+CWDHCP related): If enable static IP, DHCP will be disabled; If enable DHCP, static IP will be disabled; This will depend on the last configuration.

Deprecated

AT+CIPSTA_CUR – Set ip address of ESP8266 station

Set IP address of ESP8266 station, won't save to Flash.

Type: **query** Function: **Get ip address of ESP8266 station.** Instruction: **AT+CIPSTA_CUR?**	Response: **+CIPSTA_CUR:<ip>** **OK** Param description: <ip> string, ip address of station
Type: **set** Function: **Set ip address of ESP8266 station.** Instruction: **AT+CIPSTA_CUR=<ip>[,<gateway>,<netmask>]**	Response: **OK** Param description: <ip> string, ip address of station <gateway> gateway <netmask> netmask

Example:

AT+CIPSTA_CUR="192.168.6.100","192.168.6.1","255.255.255.0"

Note:

1. This configuration will not store in Flash.
2. This configuration interacts with DHCP related AT commands(AT+CWDHCP related): If enable static IP, DHCP will be disabled; If enable DHCP, static IP will be disabled; This will depend on the last configuration.

AT+CIPSTA_DEF — Set ip address of ESP8266 station

Set IP address of ESP8266 station, save to Flash.

Type: **query** Function: **Get ip address of ESP8266 station.** Instruction: **AT+CIPSTA_DEF?**	Response: **+CIPSTA_DEF:<ip>** **OK** Param description: <ip> string, ip address of station
Type: **set** Function: **Set ip address of ESP8266 station.** Instruction: **AT+CIPSTA_DEF=<ip>[,<gateway>,<netmask>]**	Response: **OK** Param description: <ip> string, ip address of station <gateway> gateway <netmask> netmask

Example:

AT+CIPSTA_DEF="192.168.6.100","192.168.6.1","255.255.255.0"

Note:

1. This configuration will store in Flash user parameter area.
2. This configuration interacts with DHCP related AT commands(AT+CWDHCP related): If enable static IP, DHCP will be disabled; If enable DHCP, static IP will be disabled; This will depend on the last configuration.

AT+CIPAP — Set ip address of ESP8266 softAP

Please use AT+CIPAP_CUR or AT+CIPAP_DEF instead.

Type: **query** Function: **Get ip address of ESP8266 softAP.** Instruction: **AT+CIPAP?**	Response: **+CIPAP:<ip>** **OK** Param description: <ip> string, ip address of softAP
Type: **set** Function: **Set ip address of ESP8266 softAP.** Instruction: **AT+CIPAP=<ip>[,<gateway>,<netmask>]**	Response: **OK** Param description: <ip> string, ip address of softAP <gateway> gateway <netmask> netmask

Example:

AT+CIPAP="192.168.5.1",“192.168.5.1”,“255.255.255.0”

Note:

1. This configuration will store in Flash user parameter area.
2. ESP8266 only support class C IP address
3. This configuration interacts with DHCP related AT commands(AT+CWDHCP related): If enable static IP, DHCP will be disabled; If enable DHCP, static IP will be disabled; This will depend on the last configuration.

Deprecated

AT+CIPAP_CUR — Set ip address of ESP8266 softAP

Set IP address of ESP8266 softAP, won't save to Flash.

Type: **query** Function: **Get ip address of ESP8266 softAP.** Instruction: **AT+CIPAP_CUR?**	Response: **+CIPAP_CUR:<ip>** **OK** Param description: <ip> string, ip address of softAP
Type: **set** Function: **Set ip address of ESP8266 softAP.** Instruction: **AT+CIPAP_CUR=<ip>[,<gateway>,<netmask>]**	Response: **OK** Param description: <ip> string, ip address of softAP <gateway> gateway <netmask> netmask

Example:

AT+CIPAP_CUR="192.168.5.1","192.168.5.1","255.255.255.0"

Note:

1. This configuration will not store in Flash.
2. ESP8266 only support class C IP address
3. This configuration interacts with DHCP related AT commands(AT+CWDHCP related): If enable static IP, DHCP will be disabled; If enable DHCP, static IP will be disabled; This will depend on the last configuration.

AT+CIPAP_DEF — Set ip address of ESP8266 softAP

Set IP address of ESP8266 softAP, save to Flash.

Type: **query** Function: **Get ip address of ESP8266 softAP.** Instruction: **AT+CIPAP_DEF?**	Response: **+CIPAP_DEF:<ip>** **OK** Param description: <ip> string, ip address of softAP
Type: **set** Function: **Set ip address of ESP8266 softAP.** Instruction: **AT+CIPAP_DEF=<ip>[,<gateway>,<netmask>]**	Response: **OK** Param description: <ip> string, ip address of softAP <gateway> gateway <netmask> netmask

Example:

AT+CIPAP_DEF="192.168.5.1","192.168.5.1","255.255.255.0"

Note:

1. This configuration will store in Flash user parameter area.
2. ESP8266 only support class C IP address
3. This configuration interacts with DHCP related AT commands(AT+CWDHCP related): If enable static IP, DHCP will be disabled; If enable DHCP, static IP will be disabled; This will depend on the last configuration.

AT+CWSTARTSMART

Start SmartConfig

Type: **execute** Function: **Start SmartConfig** Instruction: **AT+CWSTARTSMART=<type>**	Response: **OK** Param description: <type> 1 – ESP_TOUCH 2 – AirKiss

Example:

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

Note:

1. You can apply for more documents about our SmartConfig from Espressif.
2. Has to be ESP8266 station mode
3. Message "Smart get wifi info" means Smart Config get AP's information successfully, then ESP8266 try to connect to target AP, print "WIFI CONNECTED" and "WIFI GOT IP" if succeed.
4. ESP8266 can't do anything during SmartConfig so please wait till it succeed or use command "AT+CWSTOPSMART" to stop SmartConfig.

AT+CWSTOPSMART

Stop SmartConfig

Type: **execute** Function: **Stop SmartConfig.** Instruction: **AT+CWSTOPSMART**	Response: **OK** Param description: null

Note:

1. No matter SmartConfig succeed or not, please always call "AT +CWSTOPSMART" to release the buffer it took.

TCP/IP Related
Overview

Instruction	Description
AT+CIPSTATUS	Get connection status
AT+CIPSTART	Establish TCP connection or register UDP port
AT+CIPSEND	Send data
AT+CIPSENDEX	Send data If <length> or “\0” is met, data will be sent
AT+CIPSENDBUF	Write data into TCP-send-buffer
AT+CIPBUFRESET	Reset segment ID count
AT+CIPBUFSTATUS	Check status of TCP-send-buffer
AT+CIPCHECKSEG	Check if a specific segment is sent or not
AT+CIPCLOSE	Close TCP/UDP connection
AT+CIFSR	Get local IP address
AT+CIPMUX	Set multiple connections mode
AT+CIPSERVER	Configure as server
AT+CIPMODE	Set transmission mode
AT+SAVETRANSLINK	Save transparent transmission link to Flash
AT+CIPSTO	Set timeout when ESP8266 runs as TCP server
AT+CIUPDATE	Force OTA(upgrade through network)
AT+PING	Function PING
+IPD	Data received from network

Instructions

AT+CIPSTATUS

Information about connection.

Type: **execute**

Function:

Get information about connection

Instruction:

AT+CIPSTATUS

Response:

STATUS:<stat>
+CIPSTATUS:<id>,<type>,<addr>,<port>,<tetype>

OK

Param description:

<stat>
2 – Got IP
3 – Connected
4 – Disconnected

<id>
id of the connection (0~4), for multi-connect

<type>
“TCP” or “UDP”

<addr>
string, IP address

<port>
port number

<tetype>
0 – ESP8266 run as a client
1 – ESP8266 run as a server

AT+CIPSTART

Establish TCP connection or register UDP port, start connection.

Type: **test**

Function:

Get the information of param.

Instruction:

AT+CIPSTART=?

Response:

If AT+CIPMUX=0
+CIPSTART:(<type>),(<IPaddress>),(<port>)[,(<localport>),(<mode>)]
+CIPSTART:(<type>),(<domainname>),(<port>)[,(<localport>),(<mode>)]
OK

If AT+CIPMUX=1
+CIPSTART:(id),(<type>),(<IPaddress>),(<port>)[,(<localport>),(<mode>)] +CIPSTART:(id),(<type>),(<domain name>),(<port>)[,(<localport>),(<mode>)]
OK

Param description:

null

Type: **set**

Function:

Start a connection as client.

Instruction:

SINGLE CONNECTION
(+CIPMUX=0)
AT+CIPSTART=<type>,<addr>,<port> [,(<localport>),(<mode>)][,<TCP keep alive>]

MULTIPLE CONNECTIONS
(+CIPMUX=1)
AT+CIPSTART=<id><type>,<addr>,<port> [,(<localport>),(<mode>)][,<TCP keep alive>]

Response:

OK
ERROR
ALREADY CONNECT

Param description:

<id>
ID of connection (0-4)
<type>
"TCP" or "UDP"
<addr>
string, remote IP
<port>
string, remote port
<localport>
for UDP only
<mode>
for UDP only
0 - destination peer entity of UDP will not change.
1 - destination peer entity of UDP can change once
2 - destination peer entity of UDP is allowed to change.
<TCP keep alive>
0 - default
1 ~ 7200 Keep alive interval

Example:

AT+CIPSTART="TCP","192.168.101.110",1000

AT+CIPSEND
Send data

Type: **test**

Function:

Only for test.

Instruction:

AT+CIPSEND=?

Response:

OK

Param description:

null

Type: **set**

Function:

Set length of the data that will be sent. For normal send.

Instruction:

SINGLE CONNECTION
(+CIPMUX=0)
AT+CIPSEND=<length>

MULTIPLE CONNECTIONS
(+CIPMUX=1)
AT+CIPSEND=<id>,<length>
[,<remote IP>,<remote port>]

Response:

Wrap return “>” after set command. Begins receive of serial data, when data length is met, starts transmission of data.

If connection cannot be established or gets disconnected during send, returns

ERROR

If data is transmitted successfully, returns

SEND OK

Param description:

<id>
ID of transmit connection
<length>
data length, MAX 2048 bytes
<remote IP>
UDP transmission can set remote IP when send data
<remote port>
UDP transmission can set remote port when send data

Type: **execute**

Function:

Send data. For unvarnished transmission mode.

Instruction:

AT+CIPSEND

Response:

Wrap return “>” after execute command. Enters unvarnished transmission, 20ms interval between each packet, maximum 2048 bytes per packet. When single packet containing “+++” is received, it returns to command mode.

This command can only be used in unvarnished transmission mode which require to be single connection mode.

AT+CIPSENDEX
Send data

Type: **test**

Function:

Only for test.

Instruction:

AT+CIPSENDEX=?

Response:

OK

Param description:

null

Type: **set**

Instruction:

SINGLE CONNECTION
(+CIPMUX=0)
AT+CIPSENDEX=<length>

MULTIPLE CONNECTIONS
(+CIPMUX=1)
AT+CIPSENDEX=<id>,<length>
[,<remote IP>,<remote port>]

Response:

Wrap return ">" after set command.
Begins receive of serial data, when data length or “\0” is met, starts transmission of data.
So if sending “\0” is needed, please send it as “\\0”
If connection cannot be established or gets disconnected during send, returns

ERROR

If data is transmitted successfully, returns

SEND OK

Param description:

<id>
ID of the connection (0~4), for multi-connect
<length>
data length, MAX 2048 bytes
<remote IP>
UDP transmission can set remote IP when send data
<remote port>
UDP transmission can set remote port when send data

AT+CIPSENDBUF

Write data into TCP-send-buffer

Type: **set**

Instruction:

SINGLE CONNECTION
(+CIPMUX=0)
AT+CIPSENDBUF=<length>

MULTIPLE CONNECTIONS
(+CIPMUX=1)
AT+CIPSENDEX=<id>,<length>

Response:

<current segment ID>,<segment ID of which sent successfully>
OK
>

Wrap return ">" begins receiving of serial data, when data <length> is met, send it; data more than <length> will be discarded, and returns “busy” If connection cannot be established,or it’s not a TCP connection or buffer full, or some other error occurred, returns

ERROR

If data is transmitted successfully, for a single connection returns

<segment ID>,SEND OK

for a multiple connection returns

<id>,<segment ID>,SEND OK

Param description:

<id>
: ID of the connection (0~4), for multi-connect

<segment ID>
: uint32, starts from 1, add 1 every time be called

<length>
: data length, MAX 2048 bytes

Note:

1. This command only write data into TCP-send-buffer, so it can be called continually, needn’t wait for **“SEND OK”**;if a TCP segment is sent successfully, it will return **<segment ID>,SEND OK**

AT+CIPSENDBUFRESET

Reset segment ID count

Type: **set**

Function:

Set length of the data that will be sent. For normal send.

Instruction:

SINGLE CONNECTION
(+CIPMUX=0)
AT+CIPBUFRESET

MULTIPLE CONNECTIONS
(+CIPMUX=1)
AT+CIPBUFRESET=<id>

Response:

OK

If connection is not established or there are still TCP data wait for sending, returns

ERROR

If data is transmitted successfully, for a single connection returns

---

Param description:

<id>
ID of the connection (0~4), for multi-connect

AT+CIPBUFSTATUS

Check status of TCP-send-buffer

Type: **test**

Instruction:

SINGLE CONNECTION
(+CIPMUX=0)
AT+CIPBUFSTATUS

MULTIPLE CONNECTIONS
(+CIPMUX=1)
AT+CIPBUFSTATUS=<id>

Response:

<next segment ID>, < segment ID of which has sent >, < segment ID of which sent successfully>, <remain buffer size>, <queue number>

OK

If connection is not established returns

ERROR

Param description:

<id>
ID of the connection (0~4), for multi-connect

<next segment ID>
next segment ID will be got by AT +CIPSENDBUF

<segment ID of which has sent>
the latest segment that sent (may not succeed;

<segment ID sent successfully>
the latest segment that sent successfully;

<remain buffer size>
TCP-send-buffer remain buffer size

<que number>
available TCP queue number, it's not reliable; when queue number is 0, no more TCP data can be sent.

Example:

Single connection, **AT+CIPBUFSTATUS** returns
20,15,10,200,7

20 : means the latest segment ID is **19**, next time we call AT+CIPSENDBUF, the segment ID returned will be **20**;
15: means TCP segment of which ID is **15** is the latest segment that sent (may not succeed;
10: means TCP segment of which ID is **10** sent successfully;
200: TCP-send-buffer remain **200** bytes that available;
7: available TCP queue number, it's not reliable; when queue number is 0, no more TCP data can be sent.

AT+CIPCHECKSEG

Check if specific segment sent successfully or not

Type: **test**

Function:

Set length of the data that will be sent. For normal send.

Instruction:

SINGLE CONNECTION
(+CIPMUX=0)
AT+CIPCHECKSEG=<segment ID>

MULTIPLE CONNECTIONS
(+CIPMUX=1)
AT+CIPCHECKSEG=[<id>,]<segment ID>,<status>

Response:

[<link ID>,]<segment ID> ,<status>

If connection is not established, returns

ERROR

If data is transmitted successfully, for a single connection returns

---

Param description:

<id>
ID of the connection (0~4), for multi-connect

<segment ID>
segment ID got by AT+CIPSENDBUF

<status>
TRUE, sent successfully; FALSE, send fail

Note:

1. Only keep status of the latest 32 segments at most.

AT+CIPCLOSE
Close TCP or UDP connection

Type: **test** Function: **Only for test.** Instruction: **AT+CIPCLOSE=?**	Response: **OK** Param description: null
Type: **set** Function: **Close TCP or UDP connection.** Instruction: MULTIPLE CONNECTIONS **AT+CIPCLOSE=<id>**	Response: **OK** or **ERROR** Param description: <id> ID no. of connection to close, when id=5, all connections will be closed.
Type: **execute** Function: **For single connection mode** Instruction: **AT+CIPCLOSE**	Response: **OK** If no such connection, returns **ERROR**

Note:
1. id=5 has no effect in server mode.

AT+CIFSR
Get local IP address

Type: **test** Function: **Only for test.** Instruction: **AT+CIFSR=?**	Response: **OK** Param description: null
Type: **execute** Function: **Get local IP address.** Instruction: **AT+CIFSR**	Response: **+CIFSR:<IP address>** **OK** **ERROR** Param description: <IP address> IP address of ESP8266

Note:
1. <IP address> for softAP or station.

AT+CIPMUX

Enable multiple connections or not

Type: **query** Function: **Get param config.** Instruction: **AT+CIPMUX?**	Response: **+CIPMUX:<mode>** **OK** Param description: The same as below
Type: **set** Function: **Set connection mode.** Instruction: **AT+CIPMUX=<mode>**	Response: **OK** If already connected, returns **LINK IS BUILDED** Param description: <mode> 0 – Single connection 1 – Multiple connections

Note:

1. **"AT+CIPMUX=1"** can only be set when transparent transmission is disabled (**"AT+CIPMODE=0"**)
2. This mode can only be changed after all connections are disconnected.
3. If TCP server is started, has to delete TCP server first, then change to single connection is allowed.

AT+CIPSERVER
Configure as TCP server

Type: **set** Function: **Set TCP server.** Instruction: **AT+CIPSERVER= <mode>[,<port>]**	Response: **OK** Param description: <mode> 0 – Delete server (need to follow by restart) 1 – Create server <port> port number, default is 333

Note:
1. Server can only be created when **AT+CIPMUX=1**
2. Server monitor will automatically be created when Server is created.
3. When a client is connected to the server, it will take up one connection,be gave an id.

AT+CIPMODE
Set transfer mode

Type: **query** Function: **Query transfer mode.** Instruction: **AT+CIPMODE?**	Response: **+CIPMODE:<mode>** **OK** Param description: The same as below
Type: **set** Function: **Set transfer mode.** Instruction: **AT+CIPMODE=<mode>**	Response: **OK** If already connected, returns **LINK IS BUILDED** Param description: <mode> 0 – Normal mode 1 – UART-WiFi passthrough mode, only for TCP single connection.

AT+SAVETRANSLINK

Save transparent transmission link to Flash

Type: **set**

Function:

Set TCP server.

Instruction:

AT+SAVETRANSLINK =<mode>,<IP>,<port>[,<type>]

Response:

OK

or

ERROR

Param description:

<mode>
0 – normal mode, cancel enter UART-WiFi passthrough mode when power on
1 – save UART-WiFi passthrough mode

<IP>
remote IP

<port>
remote port

<type>
TCP or UDP, default to be “TCP”

Example:

AT+SAVETRANSLINK=1,”192.168.6.110”,1002,“TCP”

Note:

1. This command will save the UART-WiFi passthrough mode and its link into Flash user parameter area, ESP8266 will enter UART-WiFi passthrough mode since next power on.
2. As long as the IP, port numerical conformance to specification, we will save them to Flash

AT+CIPSTO
Set server timeout

Type: **query** Function: **Query server timeout.** Instruction: **AT+CIPSTO?**	Response: **+CIPSTO:<time>** **OK** Param description: The same as below
Type: **set** Function: **Set server timeout.** Instruction: **AT+CIPSTO=<time>**	Response: **OK** Param description: <time> server timeout, range 0–7200 seconds

Example:

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

Note:

1. ESP8266 as TCP server, will disconnect to TCP client that didn't communicate with it even if timeout.
2. If AT+CIPSTO=0, it will never timeout. We don't recommend that.

AT+CIUPDATE
Update through network

Type: **execute** Function: **Start upgrade.** Instruction: **AT+CIUPDATE**	Response: **+CIPUPDATE:<n>** **OK** Param description: <mode> 0 – found server 1 – connect to server 2 – download firmware 4 – start update

AT+PING
Function Ping

Type: **execute**

Function:

Ping a server.

Instruction:

AT+PING=<IP>

Response:

+<time>

OK

or

ERROR
Means ping fail

Param description:

<IP>
string, host IP or domain name
<time>
response time of ping

Example:

AT+PING="192.168.1.1"
AT+PING="www.bq.com"

+IPD
Receive network data

Instruction:

SINGLE CONNECTION
(+CIPMUX=0)
+IPD,<len>:<data>

MULTIPLE CONNECTIONS
(+CIPMUX=1)
+IPD,<id>,<len>:<data>

Param description:

<id>
number ID of connection
<len>
data length
<data>
data received

Note:
1. When the module receives network data, it will send the data through the serial port using +IPD command