



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)

## README

---

Generated on Thu Apr 9 2015 13:57:59 for API by [doxygen](#) 1.8.7



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)[Class List](#)[Class Index](#)[Class Members](#)

## Class List

Here are the classes, structs, unions and interfaces with brief descriptions:



**ESP8266** Provide an easy-to-use way to manipulate **ESP8266**

Generated on Thu Apr 9 2015 13:57:59 for API by [doxygen](#) 1.8.7



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)[Class List](#)[Class Index](#)[Class Members](#)[Public Member Functions](#) | [List of all members](#)

## ESP8266 Class Reference

Provide an easy-to-use way to manipulate **ESP8266**. More...

```
#include <ESP8266.h>
```

## Public Member Functions

bool **kick** (void)

Verify **ESP8266** whether live or not. [More...](#)

bool **restart** (void)

Restart **ESP8266** by "AT+RST". [More...](#)

String **getVersion** (void)

Get the version of AT Command Set. [More...](#)

bool **deepSleep** (uint32\_t time)

Start function of deep sleep. [More...](#)

bool **setEcho** (uint8\_t mode)

Switch the echo function. [More...](#)

bool **restore** (void)

Restore factory. [More...](#)

bool **setUart** (uint32\_t baudrate, uint8\_t pattern)

Set up a serial port configuration. [More...](#)

bool **setOprToStation** (uint8\_t pattern1=3, uint8\_t pattern2=3)

Set operation mode to station. [More...](#)

String **getWifiModeList** (void)

Get the model values list. [More...](#)

bool **setOprToSoftAP** (uint8\_t pattern1=3, uint8\_t pattern2=3)

Set operation mode to softap. [More...](#)

bool **setOprToStationSoftAP** (uint8\_t pattern1=3, uint8\_t pattern2=3)

Set operation mode to station + softap. [More...](#)

String **getAPList** (void)  
Search available AP list and return it. [More...](#)

String **getNowConecAp** (uint8\_t pattern=3)  
Search and returns the current connect AP. [More...](#)

bool **joinAP** (String ssid, String pwd, uint8\_t pattern=3)  
Join in AP. [More...](#)

bool **leaveAP** (void)  
Leave AP joined before. [More...](#)

bool **setSoftAPPParam** (String ssid, String pwd, uint8\_t chl=7,  
uint8\_t ecn=4, uint8\_t pattern=3)  
Set SoftAP parameters. [More...](#)

String **getSoftAPPParam** (uint8\_t pattern=3)  
get SoftAP parameters. [More...](#)

String **getJoinedDeviceIP** (void)  
Get the IP list of devices connected to SoftAP. [More...](#)

String **getDHCP** (uint8\_t pattern=3)  
Get the current state of DHCP. [More...](#)

bool **setDHCP** (uint8\_t mode, uint8\_t en, uint8\_t pattern=3)  
Set the state of DHCP. [More...](#)

bool **setAutoConnect** (uint8\_t en)  
make boot automatically connected. [More...](#)

String **getStationMac** (uint8\_t pattern=3)  
Get the station's MAC address. [More...](#)

bool **setStationMac** (String mac, uint8\_t pattern=3)

Set the station's MAC address. [More...](#)

String **getStationIp** (uint8\_t pattern=3)  
Get the station's IP. [More...](#)

bool **setStationIp** (String ip, String gateway, String netmask,  
uint8\_t pattern=3)  
Set the station's IP. [More...](#)

String **getAPIp** (uint8\_t pattern=3)  
Get the AP's IP. [More...](#)

bool **setAPIp** (String ip, uint8\_t pattern=3)  
Set the AP IP. [More...](#)

bool **startSmartConfig** (uint8\_t type)  
start smartconfig. [More...](#)

bool **stopSmartConfig** (void)  
stop smartconfig. [More...](#)

String **getIPStatus** (void)  
Get the current status of connection(UDP and TCP).  
[More...](#)

String **getLocalIP** (void)  
Get the IP address of **ESP8266**. [More...](#)

bool **enableMUX** (void)  
Enable IP MUX(multiple connection mode). [More...](#)

bool **disableMUX** (void)  
Disable IP MUX(single connection mode). [More...](#)

bool **createTCP** (String addr, uint32\_t port)  
Create TCP connection in single mode. [More...](#)

bool **releaseTCP** (void)

Release TCP connection in single mode. [More...](#)

bool **registerUDP** (String addr, uint32\_t port)

Register UDP port number in single mode. [More...](#)

bool **unregisterUDP** (void)

Unregister UDP port number in single mode. [More...](#)

bool **createTCP** (uint8\_t mux\_id, String addr, uint32\_t port)

Create TCP connection in multiple mode. [More...](#)

bool **releaseTCP** (uint8\_t mux\_id)

Release TCP connection in multiple mode. [More...](#)

bool **registerUDP** (uint8\_t mux\_id, String addr, uint32\_t port)

Register UDP port number in multiple mode. [More...](#)

bool **unregisterUDP** (uint8\_t mux\_id)

Unregister UDP port number in multiple mode. [More...](#)

bool **setTCPServerTimeout** (uint32\_t timeout=180)

Set the timeout of TCP Server. [More...](#)

bool **startTCPServer** (uint32\_t port=333)

Start TCP Server(Only in multiple mode). [More...](#)

bool **stopTCPServer** (void)

Stop TCP Server(Only in multiple mode). [More...](#)

bool **setCIPMODE** (uint8\_t mode)

Set the module transfer mode. [More...](#)

bool **startServer** (uint32\_t port=333)

Start Server(Only in multiple mode). [More...](#)

bool **stopServer** (void)

Stop Server(Only in multiple mode). [More...](#)

bool **saveTransLink** (uint8\_t mode, String ip, uint32\_t port)

Save the passthrough links. [More...](#)

bool **setPing** (String ip)

PING COMMAND. [More...](#)

bool **send** (const uint8\_t \*buffer, uint32\_t len)

Send data based on TCP or UDP builded already in single mode. [More...](#)

bool **send** (uint8\_t mux\_id, const uint8\_t \*buffer, uint32\_t len)

Send data based on one of TCP or UDP builded already in multiple mode. [More...](#)

uint32\_t **recv** (uint8\_t \*buffer, uint32\_t buffer\_size, uint32\_t timeout=1000)

Receive data from TCP or UDP builded already in single mode. [More...](#)

uint32\_t **recv** (uint8\_t mux\_id, uint8\_t \*buffer, uint32\_t buffer\_size, uint32\_t timeout=1000)

Receive data from one of TCP or UDP builded already in multiple mode. [More...](#)

uint32\_t **recv** (uint8\_t \*coming\_mux\_id, uint8\_t \*buffer, uint32\_t buffer\_size, uint32\_t timeout=1000)

Receive data from all of TCP or UDP builded already in multiple mode. [More...](#)

## Detailed Description

---

Provide an easy-to-use way to manipulate **ESP8266**.

### Examples:

[ConnectWiFi.ino](#), [HTTPGET.ino](#), [TCPClientMultiple.ino](#),  
[TCPClientSingle.ino](#), [TCPClientSingleUNO.ino](#),  
[TCPServer.ino](#), [test.ino](#), [UDPClientMultiple.ino](#), and  
[UDPClientSingle.ino](#).

Definition at line **42** of file **ESP8266.h**.

## Member Function Documentation

---

```
bool ESP8266::createTCP( String    addr,  
                         uint32_t  port  
                       )
```

---

Create TCP connection in single mode.

### Parameters

**addr** - the IP or domain name of the target host.  
**port** - the port number of the target host.

### Return values

**true** - success.  
**false** - failure.

### Examples:

[HTTPGET.ino](#), [TCPClientMultiple.ino](#), [TCPClientSingle.ino](#),  
and [TCPClientSingleUNO.ino](#).

Definition at line **298** of file [ESP8266.cpp](#).

```
bool ESP8266::createTCP( uint8_t  mux_id,  
                         String    addr,  
                         uint32_t  port  
                       )
```

---

Create TCP connection in multiple mode.

### Parameters

**mux\_id** - the identifier of this TCP(available value: 0 - 4).  
**addr** - the IP or domain name of the target host.  
**port** - the port number of the target host.

## Return values

- true** - success.
- false** - failure.

Definition at line [318](#) of file [ESP8266.cpp](#).

## **bool ESP8266::deepSleep ( uint32\_t time )**

---

Start function of deep sleep.

### Parameters

- time** - the sleep time.

## Return values

- true** - success.
- false** - failure.

### Note

the feature requires hardware support.

Definition at line [101](#) of file [ESP8266.cpp](#).

## **bool ESP8266::disableMUX ( void )**

---

Disable IP MUX(single connection mode).

In single connection mode, only one TCP or UDP communication can be builded.

## Return values

- true** - success.
- false** - failure.

### Examples:

[HTTPGET.ino](#), [TCPClientSingle.ino](#),

[TCPClientSingleUNO.ino](#), and [UDPClientSingle.ino](#).

Definition at line [293](#) of file [ESP8266.cpp](#).

### **bool ESP8266::enableMUX ( void )**

---

Enable IP MUX(multiple connection mode).

In multiple connection mode, a couple of TCP and UDP communication can be builded. They can be distinguished by the identifier of TCP or UDP named mux\_id.

#### **Return values**

**true** - success.

**false** - failure.

#### **Examples:**

[TCPClientMultiple.ino](#), [TCPServer.ino](#), and [UDPClientMultiple.ino](#).

Definition at line [288](#) of file [ESP8266.cpp](#).

### **String ESP8266::getAPIp ( uint8\_t pattern = 3 )**

---

Get the AP's IP.

#### **Parameters**

**pattern** -1 send "AT+CIPAP\_DEF?" -2 send "AT+CIPAP\_CUR?" -3 send "AT+CIPAP?".

#### **Returns**

ap's ip.

#### **Note**

This method should not be called when station mode.

#### **Examples:**

[test.ino](#).

Definition at line [249](#) of file [ESP8266.cpp](#).

## **String ESP8266::getAPList ( void )**

---

Search available AP list and return it.

### **Returns**

the list of available APs.

### **Note**

This method will occupy a lot of memory(hundreds of Bytes to a couple of KBytes). Do not call this method unless you must and ensure that your board has enough memory left.

### **Examples:**

[test.ino](#).

Definition at line [171](#) of file [ESP8266.cpp](#).

## **String ESP8266::getDHCP ( uint8\_t pattern = 3 )**

---

Get the current state of DHCP.

### **Parameters**

**pattern** -1 send "AT+CWDHCP\_DEF?" -2 send "AT+CWDHCP\_CUR?" -3 send "AT+CWDHCP?".

### **Returns**

the state of DHCP.

### **Examples:**

[test.ino](#).

Definition at line [209](#) of file [ESP8266.cpp](#).

## **String ESP8266::getIPStatus ( void )**

---

Get the current status of connection(UDP and TCP).

### **Returns**

the status.

### **Examples:**

[TCPserver.ino](#), and [test.ino](#).

Definition at line [274](#) of file [ESP8266.cpp](#).

## **String ESP8266::getJoinedDeviceIP ( void )**

---

Get the IP list of devices connected to SoftAP.

### **Returns**

the list of IP.

### **Note**

This method should not be called when station mode.

### **Examples:**

[test.ino](#).

Definition at line [202](#) of file [ESP8266.cpp](#).

## **String ESP8266::getLocalIP ( void )**

---

Get the IP address of [ESP8266](#).

### **Returns**

the IP list.

### **Examples:**

[ConnectWiFi.ino](#), [HTTPGET.ino](#), [TCPClientMultiple.ino](#),  
[TCPClientSingle.ino](#), [TCPClientSingleUNO.ino](#),  
[TCPserver.ino](#), [UDPClientMultiple.ino](#), and

## **UDPClientSingle.ino.**

Definition at line [281](#) of file [ESP8266.cpp](#).

---

### **String ESP8266::getNowConecAp ( uint8\_t pattern = 3 )**

Search and returns the current connect AP.

#### **Parameters**

**pattern** -1, send "AT+CWJAP\_DEF?", -2, send  
"AT+CWJAP\_CUR?", -3, send "AT+CWJAP?".

#### **Returns**

the ssid of AP connected now.

#### **Examples:**

[test.ino](#).

Definition at line [163](#) of file [ESP8266.cpp](#).

---

### **String ESP8266::getSoftAPPParam ( uint8\_t pattern = 3 )**

get SoftAP parameters.

#### **Parameters**

**pattern** -1 send "AT+CWSAP\_DEF?" -2 send  
"AT+CWSAP\_CUR?" -3 send "AT+CWSAP?".

#### **Note**

This method should not be called when station mode.

#### **Examples:**

[test.ino](#).

Definition at line [188](#) of file [ESP8266.cpp](#).

## **String ESP8266::getStationIp ( uint8\_t pattern = 3 )**

Get the station's IP.

### **Parameters**

**pattern** -1 send "AT+CIPSTA\_DEF?" -2 send  
"AT+CIPSTA\_CUR?" -3 send "AT+CIPSTA?".

### **Returns**

the station's IP.

### **Note**

This method should not be called when ap mode.

### **Examples:**

[test.ino](#).

Definition at line [237](#) of file [ESP8266.cpp](#).

## **String ESP8266::getStationMac ( uint8\_t pattern = 3 )**

Get the station's MAC address.

### **Parameters**

**pattern** -1 send "AT+CIPSTAMAC\_DEF?=" -2 send  
"AT+CIPSTAMAC\_CUR?" -3 send "AT+CIPSTAMAC?".

### **Returns**

mac address.

### **Note**

This method should not be called when ap mode.

### **Examples:**

[test.ino](#).

Definition at line [224](#) of file [ESP8266.cpp](#).

## **String ESP8266::getVersion ( void )**

---

Get the version of AT Command Set.

### **Returns**

the string of version.

### **Examples:**

[ConnectWiFi.ino](#), [HTTPGET.ino](#), [TCPClientMultiple.ino](#),  
[TCPClientSingle.ino](#), [TCPClientSingleUNO.ino](#),  
[TCPServer.ino](#), [UDPClientMultiple.ino](#), and  
[UDPClientSingle.ino](#).

Definition at line **80** of file [ESP8266.cpp](#).

## **String ESP8266::getWifiModeList ( void )**

---

Get the model values list.

### **Returns**

the list of model.

### **Examples:**

[test.ino](#).

Definition at line **123** of file [ESP8266.cpp](#).

```
bool ESP8266::joinAP ( String ssid,
                      String pwd,
                      uint8_t pattern = 3
)
```

---

Join in AP.

### **Parameters**

**pattern** -1 send "AT+CWJAP\_DEF=" -2 send  
"AT+CWJAP\_CUR=" -3 send "AT+CWJAP=".

**ssid** - SSID of AP to join in.  
**pwd** - Password of AP to join in.

### Return values

**true** - success.  
**false** - failure.

### Note

This method will take a couple of seconds.

### Examples:

[ConnectWiFi.ino](#), [HTTPGET.ino](#), [TCPClientMultiple.ino](#),  
[TCPClientSingle.ino](#), [TCPClientSingleUNO.ino](#),  
[TCPServer.ino](#), [test.ino](#), [UDPClientMultiple.ino](#), and  
[UDPClientSingle.ino](#).

Definition at line [178](#) of file [ESP8266.cpp](#).

---

## bool ESP8266::kick ( void )

---

Verify [ESP8266](#) whether live or not.

Actually, this method will send command "AT" to [ESP8266](#) and waiting for "OK".

### Return values

**true** - alive.  
**false** - dead.

Definition at line [58](#) of file [ESP8266.cpp](#).

---

## bool ESP8266::leaveAP ( void )

---

Leave AP joined before.

### Return values

**true** - success.

**false** - failure.

**Examples:**

[test.ino](#).

Definition at line [183](#) of file [ESP8266.cpp](#).

```
uint32_t ESP8266::recv ( uint8_t * buffer,  
                          uint32_t buffer_size,  
                          uint32_t timeout = 1000  
                        )
```

---

Receive data from TCP or UDP builded already in single mode.

**Parameters**

**buffer** - the buffer for storing data.  
**buffer\_size** - the length of the buffer.  
**timeout** - the time waiting data.

**Returns**

the length of data received actually.

**Examples:**

[HTTPGET.ino](#), [TCPClientMultiple.ino](#), [TCPClientSingle.ino](#),  
[TCPClientSingleUNO.ino](#), [TCPServer.ino](#),  
[UDPClientMultiple.ino](#), and [UDPClientSingle.ino](#).

Definition at line [396](#) of file [ESP8266.cpp](#).

```
uint32_t ESP8266::recv ( uint8_t mux_id,  
                          uint8_t * buffer,  
                          uint32_t buffer_size,  
                          uint32_t timeout = 1000  
                        )
```

---

Receive data from one of TCP or UDP builded already in multiple

mode.

### Parameters

- mux\_id** - the identifier of this TCP(available value: 0 - 4).
- buffer** - the buffer for storing data.
- buffer\_size** - the length of the buffer.
- timeout** - the time waiting data.

### Returns

the length of data received actually.

Definition at line [401](#) of file [ESP8266.cpp](#).

```
uint32_t ESP8266::recv ( uint8_t * coming_mux_id,  
                         uint8_t * buffer,  
                         uint32_t buffer_size,  
                         uint32_t timeout = 1000  
                     )
```

---

Receive data from all of TCP or UDP builded already in multiple mode.

After return, coming\_mux\_id store the id of TCP or UDP from which data coming. User should read the value of coming\_mux\_id and decide what next to do.

### Parameters

- coming\_mux\_id** - the identifier of TCP or UDP.
- buffer** - the buffer for storing data.
- buffer\_size** - the length of the buffer.
- timeout** - the time waiting data.

### Returns

the length of data received actually.

Definition at line [412](#) of file [ESP8266.cpp](#).

```
bool ESP8266::registerUDP ( String    addr,  
                            uint32_t  port  
                        )
```

---

Register UDP port number in single mode.

### Parameters

**addr** - the IP or domain name of the target host.  
**port** - the port number of the target host.

### Return values

**true** - success.  
**false** - failure.

### Examples:

[UDPClientMultiple.ino](#), and [UDPClientSingle.ino](#).

Definition at line [308](#) of file [ESP8266.cpp](#).

```
bool ESP8266::registerUDP ( uint8_t  mux_id,  
                            String    addr,  
                            uint32_t  port  
                        )
```

---

Register UDP port number in multiple mode.

### Parameters

**mux\_id** - the identifier of this TCP(available value: 0 - 4).  
**addr** - the IP or domain name of the target host.  
**port** - the port number of the target host.

### Return values

**true** - success.  
**false** - failure.

Definition at line [328](#) of file [ESP8266.cpp](#).

## **bool ESP8266::releaseTCP ( void )**

---

Release TCP connection in single mode.

### **Return values**

**true** - success.

**false** - failure.

### **Examples:**

[HTTPGET.ino](#), [TCPClientMultiple.ino](#), [TCPClientSingle.ino](#),  
[TCPClientSingleUNO.ino](#), and [TCPServer.ino](#).

Definition at line [303](#) of file [ESP8266.cpp](#).

## **bool ESP8266::releaseTCP ( uint8\_t mux\_id )**

---

Release TCP connection in multiple mode.

### **Parameters**

**mux\_id** - the identifier of this TCP(available value: 0 - 4).

### **Return values**

**true** - success.

**false** - failure.

Definition at line [323](#) of file [ESP8266.cpp](#).

## **bool ESP8266::restart ( void )**

---

Restart [ESP8266](#) by "AT+RST".

This method will take 3 seconds or more.

### **Return values**

**true** - success.  
**false** - failure.

Definition at line [63](#) of file [ESP8266.cpp](#).

---

### **bool ESP8266::restore ( void )**

Restore factory.

#### **Return values**

**true** - success.  
**false** - failure.

#### **Note**

The operation can lead to restart the machine.

Definition at line [92](#) of file [ESP8266.cpp](#).

---

### **bool ESP8266::saveTransLink ( uint8\_t mode,                               String ip,                               uint32\_t port                               )**

Save the passthrough links.

#### **Return values**

**true** - success.  
**false** - failure.

#### **Examples:**

[test.ino](#).

Definition at line [363](#) of file [ESP8266.cpp](#).

---

### **bool ESP8266::send ( const uint8\_t \* buffer,**

```
    uint32_t      len  
)  
}
```

Send data based on TCP or UDP builded already in single mode.

### Parameters

**buffer** - the buffer of data to send.  
**len** - the length of data to send.

### Return values

**true** - success.  
**false** - failure.

### Examples:

[HTTPGET.ino](#), [TCPClientMultiple.ino](#), [TCPClientSingle.ino](#),  
[TCPClientSingleUNO.ino](#), [TCPServer.ino](#),  
[UDPClientMultiple.ino](#), and [UDPClientSingle.ino](#).

Definition at line [386](#) of file [ESP8266.cpp](#).

```
bool ESP8266::send ( uint8_t      mux_id,  
                      const uint8_t * buffer,  
                      uint32_t      len  
)
```

Send data based on one of TCP or UDP builded already in multiple mode.

### Parameters

**mux\_id** - the identifier of this TCP(available value: 0 - 4).  
**buffer** - the buffer of data to send.  
**len** - the length of data to send.

### Return values

**true** - success.  
**false** - failure.

Definition at line [391](#) of file [ESP8266.cpp](#).

```
bool ESP8266::setAPIp ( String ip,
                         uint8_t pattern = 3
                       )
```

---

Set the AP IP.

### Parameters

**pattern** -1 send "AT+CIPAP\_DEF=" -2 send  
"AT+CIPAP\_CUR=" -3 send "AT+CIPAP=".  
**ip** - the ip of AP.

### Return values

**true** - success.  
**false** - failure.

### Note

This method should not be called when station mode.

### Examples:

[test.ino](#).

Definition at line [256](#) of file [ESP8266.cpp](#).

---

```
bool ESP8266::setAutoConnect ( uint8_t en )
```

---

make boot automatically connected.

### Parameters

**en** -1 enable -0 disable.

### Return values

**true** - success.  
**false** - failure.

**Examples:**

[test.ino](#).

Definition at line [220](#) of file [ESP8266.cpp](#).

---

**bool ESP8266::setCIPMODE ( uint8\_t mode )**

Set the module transfer mode.

**Return values**

**true** - success.

**false** - failure.

Definition at line [358](#) of file [ESP8266.cpp](#).

---

**bool ESP8266::setDHCP ( uint8\_t mode,  
                          uint8\_t en,  
                          uint8\_t pattern = 3  
                          )**

Set the state of DHCP.

**Parameters**

**pattern** -1 send "AT+CWDHCP\_DEF=" -2 send  
                  "AT+CWDHCP\_CUR=" -3 send "AT+CWDHCP=".

**mode** - set ap or set station or set ap + station.

**en** - 0 disable DHCP - 1 enable DHCP.

**Return values**

**true** - success.

**false** - failure.

**Examples:**

[test.ino](#).

Definition at line [215](#) of file [ESP8266.cpp](#).

---

## **bool ESP8266::setEcho ( uint8\_t mode )**

---

Switch the echo function.

### **Parameters**

**mode** - 1 start echo -0 stop echo

### **Return values**

**true** - success.

**false** - failure.

Definition at line [87](#) of file [ESP8266.cpp](#).

---

## **bool ESP8266::setOprToSoftAP ( uint8\_t pattern1 = 3,                                 uint8\_t pattern2 = 3                                 )**

---

Set operation mode to softap.

### **Parameters**

**pattern1** -1, send "AT+CWMODE\_DEF?", -2, send  
"AT+CWMODE\_CUR?", -3, send "AT+CWMODE?".

**pattern2** -1, send "AT+CWMODE\_DEF=", -2, send  
"AT+CWMODE\_CUR=", -3, send "AT+CWMODE=".

### **Return values**

**true** - success.

**false** - failure.

### **Examples:**

[test.ino](#).

Definition at line [129](#) of file [ESP8266.cpp](#).

```
bool ESP8266::setOprToStation ( uint8_t pattern1 = 3,  
                                uint8_t pattern2 = 3  
)
```

Set operation mode to station.

### Parameters

**pattern1** -1, send "AT+CWMODE\_DEF?", -2, send  
"AT+CWMODE\_CUR?", -3, send "AT+CWMODE?".  
**pattern2** -1, send "AT+CWMODE\_DEF=", -2, send  
"AT+CWMODE\_CUR=", -3, send "AT+CWMODE=".

### Return values

**true** - success.  
**false** - failure.

### Examples:

[ConnectWiFi.ino](#), and [test.ino](#).

Definition at line [107](#) of file [ESP8266.cpp](#).

```
bool ESP8266::setOprToStationSoftAP ( uint8_t pattern1 = 3,  
                                       uint8_t pattern2 = 3  
)
```

Set operation mode to station + softap.

### Parameters

**pattern1** -1, send "AT+CWMODE\_DEF?", -2, send  
"AT+CWMODE\_CUR?", -3, send "AT+CWMODE?".  
**pattern2** -1, send "AT+CWMODE\_DEF=", -2, send  
"AT+CWMODE\_CUR=", -3, send "AT+CWMODE=".

### Return values

**true** - success.  
**false** - failure.

## Examples:

[HTTPGET.ino](#), [TCPClientMultiple.ino](#), [TCPClientSingle.ino](#),  
[TCPClientSingleUNO.ino](#), [TCPServer.ino](#), [test.ino](#),  
[UDPClientMultiple.ino](#), and [UDPClientSingle.ino](#).

Definition at line [146](#) of file [ESP8266.cpp](#).

---

### **bool ESP8266::setPing ( String ip )**

---

PING COMMAND.

#### Return values

**true** - success.

**false** - failure.

Definition at line [368](#) of file [ESP8266.cpp](#).

---

### **bool ESP8266::setSoftAPParam ( String ssid,                                   String pwd,                                   uint8\_t chl = 7,                                   uint8\_t ecn = 4,                                   uint8\_t pattern = 3                                  )**

---

Set SoftAP parameters.

#### Parameters

**pattern** -1 send "AT+CWSAP\_DEF=" -2 send  
"AT+CWSAP\_CUR=" -3 send "AT+CWSAP=".

**ssid** - SSID of SoftAP.

**pwd** - PASSWORD of SoftAP.

**chl** - the channel (1 - 13, default: 7).

**ecn** - the way of encryption (0 - OPEN, 1 - WEP, 2 -  
WPA\_PSK, 3 - WPA2\_PSK, 4 - WPA\_WPA2\_PSK,  
default: 4).

## Return values

**true** - success.  
**false** - failure.

### Note

This method should not be called when station mode.

### Examples:

[test.ino](#).

Definition at line [197](#) of file [ESP8266.cpp](#).

```
bool ESP8266::setStationIp ( String ip,  
                           String gateway,  
                           String netmask,  
                           uint8_t pattern = 3  
 )
```

---

Set the station's IP.

### Parameters

**pattern** -1 send "AT+CIPSTA\_DEF=" -2 send  
"AT+CIPSTA\_CUR=" -3 send "AT+CIPSTA=".  
**ip** - the ip of station.  
**gateway** -the gateway of station.  
**netmask** -the netmask of station.

## Return values

**true** - success.  
**false** - failure.

### Note

This method should not be called when ap mode.

### Examples:

[test.ino](#).

Definition at line [244](#) of file [ESP8266.cpp](#).

```
bool ESP8266::setStationMac ( String mac,  
                           uint8_t pattern = 3  
                         )
```

---

Set the station's MAC address.

### Parameters

**pattern** -1 send "AT+CIPSTAMAC\_DEF=" -2 send  
"AT+CIPSTAMAC\_CUR=" -3 send  
"AT+CIPSTAMAC=".  
**mac** - the mac address of station.

### Return values

**true** - success.  
**false** - failure.

### Examples:

[test.ino](#).

Definition at line [232](#) of file [ESP8266.cpp](#).

---

```
bool ESP8266::setTCPServerTimeout ( uint32_t timeout = 180 )
```

---

Set the timeout of TCP Server.

### Parameters

**timeout** - the duration for timeout by second(0 ~ 28800,  
default:180).

### Return values

**true** - success.  
**false** - failure.

### Examples:

## [TCPServer.ino](#).

Definition at line [338](#) of file [ESP8266.cpp](#).

---

```
bool ESP8266::setUart ( uint32_t baudrate,
                         uint8_t  pattern
                       )
```

---

Set up a serial port configuration.

### Parameters

**pattern** -1 send "AT+UART=", -2 send "AT+UART\_CUR=", -3 send "AT+UART\_DEF=".

**baudrate** - the uart baudrate.

### Return values

**true** - success.

**false** - failure.

### Note

Only allows baud rate design, for the other parameters: databits-8,stopbits -1,parity -0,flow control -0 .

### Examples:

[test.ino](#).

Definition at line [96](#) of file [ESP8266.cpp](#).

---

```
bool ESP8266::startServer ( uint32_t port = 333 )
```

---

Start Server(Only in multiple mode).

### Parameters

**port** - the port number to listen(default: 333).

### Return values

**true** - success.  
**false** - failure.

## See also

[String getIPStatus\(void\);](#)  
[uint32\\_t recv\(uint8\\_t \\*coming\\_mux\\_id, uint8\\_t \\*buffer, uint32\\_t len, uint32\\_t timeout\);](#)

Definition at line [376](#) of file [ESP8266.cpp](#).

## bool ESP8266::startSmartConfig ( uint8\_t type )

---

start smartconfig.

### Parameters

**type** -1:ESP\_TOUCH -2:AirKiss.

### Return values

**true** - success.  
**false** - failure.

### Examples:

[test.ino](#).

Definition at line [261](#) of file [ESP8266.cpp](#).

## bool ESP8266::startTCPServer ( uint32\_t port = 333 )

---

Start TCP Server(Only in multiple mode).

After started, user should call method: getIPStatus to know the status of TCP connections. The methods of receiving data can be called for user's any purpose. After communication, release the TCP connection is needed by calling method: releaseTCP with mux\_id.

### Parameters

**port** - the port number to listen(default: 333).

## Return values

- true** - success.
- false** - failure.

## See also

[String getIPStatus\(void\);](#)  
[uint32\\_t recv\(uint8\\_t \\*coming\\_mux\\_id, uint8\\_t \\*buffer, uint32\\_t len, uint32\\_t timeout\);](#)  
[bool releaseTCP\(uint8\\_t mux\\_id\);](#)

## Examples:

[TCPserver.ino](#).

Definition at line [343](#) of file [ESP8266.cpp](#).

---

## bool ESP8266::stopServer ( void )

Stop Server(Only in multiple mode).

## Return values

- true** - success.
- false** - failure.

Definition at line [381](#) of file [ESP8266.cpp](#).

---

## bool ESP8266::stopSmartConfig ( void )

stop smartconfig.

## Return values

- true** - success.
- false** - failure.

## Examples:

[test.ino](#).

Definition at line [266](#) of file [ESP8266.cpp](#).

---

### **bool ESP8266::stopTCPServer ( void )**

---

Stop TCP Server(Only in multiple mode).

#### **Return values**

**true** - success.

**false** - failure.

Definition at line [351](#) of file [ESP8266.cpp](#).

---

### **bool ESP8266::unregisterUDP ( void )**

---

Unregister UDP port number in single mode.

#### **Return values**

**true** - success.

**false** - failure.

#### **Examples:**

[UDPClientMultiple.ino](#), and [UDPClientSingle.ino](#).

Definition at line [313](#) of file [ESP8266.cpp](#).

---

### **bool ESP8266::unregisterUDP ( uint8\_t mux\_id )**

---

Unregister UDP port number in multiple mode.

#### **Parameters**

**mux\_id** - the identifier of this TCP(available value: 0 - 4).

#### **Return values**

**true** - success.

**false** - failure.

Definition at line [333](#) of file [ESP8266.cpp](#).

---

The documentation for this class was generated from the following files:

- [ESP8266.h](#)
  - [ESP8266.cpp](#)
- 

Generated on Thu Apr 9 2015 13:57:59 for API by [doxygen](#) 1.8.7



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)[Class List](#)[Class Index](#)[Class Members](#)

## Class Index

E

E

[ESP8266](#)

E

Generated on Thu Apr 9 2015 13:57:59 for API by [doxygen](#) 1.8.7



# API

## For Arduino developers

Main Page	Related Pages	Classes	Files	Examples		
Class List	Class Index	Class Members				
All	Functions					
c	d	e	g	j	k	l
r	s	u				

Here is a list of all documented class members with links to the class documentation for each member:

### - c -

- `createTCP()` : [ESP8266](#)

### - d -

- `deepSleep()` : [ESP8266](#)
- `disableMUX()` : [ESP8266](#)

### - e -

- `enableMUX()` : [ESP8266](#)

### - g -

- `getAPIp()` : [ESP8266](#)
- `getAPList()` : [ESP8266](#)
- `getDHCP()` : [ESP8266](#)
- `getIPStatus()` : [ESP8266](#)
- `getJoinedDeviceIP()` : [ESP8266](#)
- `getLocalIP()` : [ESP8266](#)
- `getNowConecAp()` : [ESP8266](#)
- `getSoftAPPParam()` : [ESP8266](#)
- `getStationIp()` : [ESP8266](#)
- `getStationMac()` : [ESP8266](#)

- `getVersion()` : **ESP8266**
- `getWifiModeList()` : **ESP8266**

- j -

- `joinAP()` : **ESP8266**

- k -

- `kick()` : **ESP8266**

- l -

- `leaveAP()` : **ESP8266**

- r -

- `recv()` : **ESP8266**
- `registerUDP()` : **ESP8266**
- `releaseTCP()` : **ESP8266**
- `restart()` : **ESP8266**
- `restore()` : **ESP8266**

- s -

- `saveTransLink()` : **ESP8266**
- `send()` : **ESP8266**
- `setAPIp()` : **ESP8266**
- `setAutoConnect()` : **ESP8266**
- `setCIPMODE()` : **ESP8266**
- `setDHCP()` : **ESP8266**
- `setEcho()` : **ESP8266**
- `setOprToSoftAP()` : **ESP8266**
- `setOprToStation()` : **ESP8266**
- `setOprToStationSoftAP()` : **ESP8266**
- `setPing()` : **ESP8266**
- `setSoftAPPParam()` : **ESP8266**
- `setStationIp()` : **ESP8266**
- `setStationMac()` : **ESP8266**

- setTCPServerTimeout() : **ESP8266**
- setUart() : **ESP8266**
- startServer() : **ESP8266**
- startSmartConfig() : **ESP8266**
- startTCPServer() : **ESP8266**
- stopServer() : **ESP8266**
- stopSmartConfig() : **ESP8266**
- stopTCPServer() : **ESP8266**

- **U** -

- unregisterUDP() : **ESP8266**



# API

## For Arduino developers

Main Page	Related Pages	Classes	Files	Examples		
Class List	Class Index	Class Members				
All	Functions					
c	d	e	g	j	k	l
r	s	u				

### - C -

- createTCP() : [ESP8266](#)

### - d -

- deepSleep() : [ESP8266](#)
- disableMUX() : [ESP8266](#)

### - e -

- enableMUX() : [ESP8266](#)

### - g -

- getAPIp() : [ESP8266](#)
- getAPList() : [ESP8266](#)
- getDHCP() : [ESP8266](#)
- getIPStatus() : [ESP8266](#)
- getJoinedDeviceIP() : [ESP8266](#)
- getLocalIP() : [ESP8266](#)
- getNowConecAp() : [ESP8266](#)
- getSoftAPPParam() : [ESP8266](#)
- getStationIp() : [ESP8266](#)
- getStationMac() : [ESP8266](#)
- getVersion() : [ESP8266](#)

- getWifiModeList() : **ESP8266**

- j -

- joinAP() : **ESP8266**

- k -

- kick() : **ESP8266**

- l -

- leaveAP() : **ESP8266**

- r -

- recv() : **ESP8266**
- registerUDP() : **ESP8266**
- releaseTCP() : **ESP8266**
- restart() : **ESP8266**
- restore() : **ESP8266**

- s -

- saveTransLink() : **ESP8266**
- send() : **ESP8266**
- setAPIp() : **ESP8266**
- setAutoConnect() : **ESP8266**
- setCIPMODE() : **ESP8266**
- setDHCP() : **ESP8266**
- setEcho() : **ESP8266**
- setOprToSoftAP() : **ESP8266**
- setOprToStation() : **ESP8266**
- setOprToStationSoftAP() : **ESP8266**
- setPing() : **ESP8266**
- setSoftAPPParam() : **ESP8266**
- setStationIp() : **ESP8266**
- setStationMac() : **ESP8266**
- setTCPServerTimeout() : **ESP8266**

- setUart() : [ESP8266](#)
- startServer() : [ESP8266](#)
- startSmartConfig() : [ESP8266](#)
- startTCPServer() : [ESP8266](#)
- stopServer() : [ESP8266](#)
- stopSmartConfig() : [ESP8266](#)
- stopTCPServer() : [ESP8266](#)

- **U** -

- unregisterUDP() : [ESP8266](#)



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)[File List](#)[File Members](#)

## File List

Here is a list of all documented files with brief descriptions:

[detail level [1](#) [2](#) [3](#)]

<a href="#">examples</a>	
<a href="#">ConnectWiFi</a>	
<a href="#">ConnectWiFi.ino</a>	
<a href="#">HTTPGET</a>	
<a href="#">HTTPGET.ino</a>	
<a href="#">TCPClientMultiple</a>	
<a href="#">TCPClientMultiple.ino</a>	
<a href="#">TCPClientSingle</a>	
<a href="#">TCPClientSingle.ino</a>	
<a href="#">TCPClientSingleUNO</a>	
<a href="#">TCPClientSingleUNO.ino</a>	
<a href="#">TCPServer</a>	
<a href="#">TCPServer.ino</a>	
<a href="#">test</a>	
<a href="#">test.ino</a>	
<a href="#">UDPClientMultiple</a>	
<a href="#">UDPClientMultiple.ino</a>	
<a href="#">UDPClientSingle</a>	
<a href="#">UDPClientSingle.ino</a>	
<a href="#">doxygen.h</a>	Define modules in API doc
<a href="#">ESP8266.cpp</a>	The implementation of class





# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)

examples &gt;

## examples Directory Reference

## Directories

---

directory [\*\*ConnectWiFi\*\*](#)

directory [\*\*HTTPGET\*\*](#)

directory [\*\*TCPClientMultiple\*\*](#)

directory [\*\*TCPClientSingle\*\*](#)

directory [\*\*TCPClientSingleUNO\*\*](#)

directory [\*\*TCPServer\*\*](#)

directory [\*\*test\*\*](#)

directory [\*\*UDPClientMultiple\*\*](#)

directory [\*\*UDPClientSingle\*\*](#)

---



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)

examples

ConnectWiFi

## ConnectWiFi Directory Reference

## Files

---

file **ConnectWiFi.ino** [\[code\]](#)

---

Generated on Thu Apr 9 2015 13:57:59 for API by  doxygen 1.8.7



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)[File List](#)[File Members](#)

examples

ConnectWiFi

## ConnectWiFi.ino

```
1
21 #include "ESP8266.h"
22
23 #define SSID          "ITEAD"
24 #define PASSWORD      "12345678"
25
26 ESP8266 wifi(Serial1);
27
28 void setup(void)
29 {
30     Serial.begin(9600);
31     Serial.print("setup begin\r\n");
32
33     Serial.print("FW Version: ");
34
35     Serial.println(wifi.getVersion().c_str());
36
37     if (wifi.setOprToStation()) {
38         Serial.print("to station ok\r\n");
39     } else {
40         Serial.print("to station err\r\n");
41     }
42
43     if (wifi.joinAP(SSID, PASSWORD)) {
```

```
44     Serial.print("Join AP success\r\n");
45     Serial.print("IP: ");
46
47     Serial.println(wifi.getLocalIP().c_str());
48 } else {
49     Serial.print("Join AP failure\r\n");
50
51     Serial.print("setup end\r\n");
52 }
53
54 void loop(void)
55 {
56 }
```



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)

examples

HTTPGET

## HTTPGET Directory Reference

## Files

---

file **HTTPGET.ino** [code]

---

Generated on Thu Apr 9 2015 13:57:59 for API by  doxygen 1.8.7



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)[File List](#)[File Members](#)

examples

HTTPGET

## HTTPGET.ino

```
1
22 #include "ESP8266.h"
23
24 #define SSID          "ITEAD"
25 #define PASSWORD      "12345678"
26 #define HOST_NAME     "www.baidu.com"
27 #define HOST_PORT      (80)
28
29 ESP8266 wifi(Serial1);
30
31 void setup(void)
32 {
33     Serial.begin(9600);
34     Serial.print("setup begin\r\n");
35
36     Serial.print("FW Version:");
37
38     Serial.println(wifi.getVersion().c_str());
39
40     if (wifi.setOprToStationSoftAP()) {
41         Serial.print("to station + softap
42         ok\r\n");
43     } else {
44         Serial.print("to station + softap
45         err\r\n");
46 }
```

```
43     }
44
45     if (wifi.joinAP(SSID, PASSWORD)) {
46         Serial.print("Join AP success\r\n");
47
48         Serial.print("IP:");
49         Serial.println(
50             wifi.getLocalIP().c_str());
51     } else {
52         Serial.print("Join AP failure\r\n");
53     }
54
55     if (wifi.disableMUX()) {
56         Serial.print("single ok\r\n");
57     } else {
58         Serial.print("single err\r\n");
59     }
60
61     Serial.print("setup end\r\n");
62 }
63 void loop(void)
64 {
65     uint8_t buffer[1024] = {0};
66
67     if (wifi.createTCP(HOST_NAME,
68 HOST_PORT)) {
69         Serial.print("create tcp ok\r\n");
70     } else {
71         Serial.print("create tcp err\r\n");
72     }
73
74     char *hello = "GET / HTTP/1.1\r\nHost:
75         www.baidu.com\r\nConnection: close\r\n\r\n";
76
77     wifi.send((const uint8_t*)hello,
78     strlen(hello));
79 }
```

```
76     uint32_t len = wifi.recv(buffer,
77     sizeof(buffer), 10000);
78     if (len > 0) {
79         Serial.print("Received:[");
80         for(uint32_t i = 0; i < len; i++) {
81             Serial.print((char)buffer[i]);
82         }
83         Serial.print("]\r\n");
84     }
85     if (wifi.releaseTCP()) {
86         Serial.print("release tcp ok\r\n");
87     } else {
88         Serial.print("release tcp err\r\n");
89     }
90
91     while(1);
92
93 }
94 }
```



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)

examples

TCPClientMultiple

## TCPClientMultiple Directory Reference

## Files

---

file **TCPClientMultiple.ino** [\[code\]](#)

---

Generated on Thu Apr 9 2015 13:57:59 for API by  doxygen 1.8.7



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)[File List](#)[File Members](#)

examples

TCPClientMultiple

## TCPClientMultiple.ino

```
1
21 #include "ESP8266.h"
22
23 #define SSID          "ITEAD"
24 #define PASSWORD      "12345678"
25 #define HOST_NAME     "192.168.1.1"
26 #define HOST_PORT     (8090)
27
28 ESP8266 wifi(Serial1);
29
30 void setup(void)
31 {
32     Serial.begin(9600);
33     Serial.print("setup begin\r\n");
34
35     Serial.print("FW Version: ");
36
37     Serial.println(wifi.getVersion().c_str());
38
39     if (wifi.setOprToStationSoftAP()) {
40         Serial.print("to station + softap
ok\r\n");
41     } else {
42         Serial.print("to station + softap
```

```
    err\r\n");
43    }
44
45    if (wifi.joinAP(SSID, PASSWORD)) {
46        Serial.print("Join AP success\r\n");
47        Serial.print("IP: ");
48
49        Serial.println(wifi.getLocalIP().c_str());
50    } else {
51        Serial.print("Join AP failure\r\n");
52    }
53
54    if (wifi.enableMUX()) {
55        Serial.print("multiple ok\r\n");
56    } else {
57        Serial.print("multiple err\r\n");
58    }
59
60    Serial.print("setup end\r\n");
61}
62void loop(void)
63{
64    uint8_t buffer[128] = {0};
65    static uint8_t mux_id = 0;
66
67    if (wifi.createTCP(mux_id, HOST_NAME,
HOST_PORT)) {
68        Serial.print("create tcp ");
69        Serial.print(mux_id);
70        Serial.println(" ok");
71    } else {
72        Serial.print("create tcp ");
73        Serial.print(mux_id);
74        Serial.println(" err");
75    }
76}
```

```
77
78     char *hello = "Hello, this is client!";
79     if (wifi.send(mux_id, (const
80         uint8_t*)hello, strlen(hello))) {
81         Serial.println("send ok");
82     } else {
83         Serial.println("send err");
84     }
85     uint32_t len = wifi.recv(mux_id, buffer,
86     sizeof(buffer), 10000);
87     if (len > 0) {
88         Serial.print("Received:[");
89         for(uint32_t i = 0; i < len; i++) {
90             Serial.print((char)buffer[i]);
91         }
92         Serial.print("]\r\n");
93     }
94     if (wifi.releaseTCP(mux_id)) {
95         Serial.print("release tcp ");
96         Serial.print(mux_id);
97         Serial.println(" ok");
98     } else {
99         Serial.print("release tcp ");
100        Serial.print(mux_id);
101        Serial.println(" err");
102    }
103
104    delay(3000);
105    mux_id++;
106    if (mux_id >= 5) {
107        mux_id = 0;
108    }
109}
110
```

Generated on Thu Apr 9 2015 13:57:59 for API by [doxygen](#) 1.8.7



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)

examples

TCPClientSingle

## TCPClientSingle Directory Reference

## Files

---

file **TCPClientSingle.ino** [\[code\]](#)

---

Generated on Thu Apr 9 2015 13:57:59 for API by  doxygen 1.8.7



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)[File List](#)[File Members](#)

examples

TCPClientSingle

## TCPClientSingle.ino

```
1
21 #include "ESP8266.h"
22
23 #define SSID          "ITEAD"
24 #define PASSWORD      "12345678"
25 #define HOST_NAME     "172.16.5.12"
26 #define HOST_PORT      (8090)
27
28 ESP8266 wifi(Serial1);
29
30 void setup(void)
31 {
32     Serial.begin(9600);
33     Serial.print("setup begin\r\n");
34
35     Serial.print("FW Version:");
36
37     Serial.println(wifi.getVersion().c_str());
38
39     if (wifi.setOprToStationSoftAP()) {
40         Serial.print("to station + softap
41 ok\r\n");
42     } else {
43         Serial.print("to station + softap
44 err\r\n");
45 }
```

```
42     }
43
44     if (wifi.joinAP(SSID, PASSWORD)) {
45         Serial.print("Join AP success\r\n");
46         Serial.print("IP:");
47         Serial.println(
48             wifi.getLocalIP().c_str());
49     } else {
50         Serial.print("Join AP failure\r\n");
51     }
52
53     if (wifi.disableMUX()) {
54         Serial.print("single ok\r\n");
55     } else {
56         Serial.print("single err\r\n");
57     }
58
59     Serial.print("setup end\r\n");
60 }
61 void loop(void)
62 {
63     uint8_t buffer[128] = {0};
64
65     if (wifi.createTCP(HOST_NAME,
HOST_PORT)) {
66         Serial.print("create tcp ok\r\n");
67     } else {
68         Serial.print("create tcp err\r\n");
69     }
70
71     char *hello = "Hello, this is client!";
72     wifi.send((const uint8_t*)hello,
    strlen(hello));
73
74     uint32_t len = wifi.recv(buffer,
sizeof(buffer), 10000);
```

```
75  if (len > 0) {
76      Serial.print("Received:[");
77      for(uint32_t i = 0; i < len; i++) {
78          Serial.print((char)buffer[i]);
79      }
80      Serial.print("]\r\n");
81  }
82
83  if (wifi.releaseTCP()) {
84      Serial.print("release tcp ok\r\n");
85  } else {
86      Serial.print("release tcp err\r\n");
87  }
88  delay(5000);
89
90 }
```



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)

examples

&gt; TCPClientSingleUNO

## TCPClientSingleUNO Directory Reference

## Files

---

file **TCPClientSingleUNO.ino** [\[code\]](#)

---

Generated on Thu Apr 9 2015 13:57:59 for API by  doxygen 1.8.7



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)[File List](#)[File Members](#)

examples

TCPClientSingleUNO

## TCPClientSingleUNO.ino

```
1
21 #include "ESP8266.h"
22 #include <SoftwareSerial.h>
23
24 #define SSID          "ITEAD"
25 #define PASSWORD      "12345678"
26 #define HOST_NAME     "172.16.5.12"
27 #define HOST_PORT     (8090)
28
29 SoftwareSerial mySerial(3, 2); /* RX:D3,
   TX:D2 */
30 ESP8266 wifi(mySerial);
31
32 void setup(void)
33 {
34     Serial.begin(9600);
35     Serial.print("setup begin\r\n");
36
37     Serial.print("FW Version:");
38
39     Serial.println(wifi.getVersion().c_str());
40
41     if (wifi.setOprToStationSoftAP()) {
42         Serial.print("to station + softap
ok\r\n");
43 }
```

```
42     } else {
43         Serial.print("to station + softap
44         err\r\n");
45     }
46
47     if (wifi.joinAP(SSID, PASSWORD)) {
48         Serial.print("Join AP success\r\n");
49         Serial.print("IP:");
50         Serial.println(
51             wifi.getLocalIP().c_str());
52     } else {
53         Serial.print("Join AP failure\r\n");
54     }
55
56     if (wifi.disableMUX()) {
57         Serial.print("single ok\r\n");
58     } else {
59         Serial.print("single err\r\n");
60     }
61
62     Serial.print("setup end\r\n");
63 }
64
65 void loop(void)
66 {
67     uint8_t buffer[128] = {0};
68
69     if (wifi.createTCP(HOST_NAME,
70 HOST_PORT)) {
71         Serial.print("create tcp ok\r\n");
72     } else {
73         Serial.print("create tcp err\r\n");
74     }
75
76     char *hello = "Hello, this is client!";
77     wifi.send((const uint8_t*)hello,
78     strlen(hello));
```

```
75
76     uint32_t len = wifi.recv(buffer,
77     sizeof(buffer), 10000);
78     if (len > 0) {
79         Serial.print("Received:[");
80         for(uint32_t i = 0; i < len; i++) {
81             Serial.print((char)buffer[i]);
82         }
83         Serial.print("]\r\n");
84     }
85     if (wifi.releaseTCP()) {
86         Serial.print("release tcp ok\r\n");
87     } else {
88         Serial.print("release tcp err\r\n");
89     }
90     delay(5000);
91 }
92 }
```



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)

examples

TCPserver

## TCPserver Directory Reference

## Files

---

file **TCPServer.ino** [\[code\]](#)

---

Generated on Thu Apr 9 2015 13:57:59 for API by  doxygen 1.8.7



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)[File List](#)[File Members](#)

examples

TCPserver

## TCPServer.ino

```
1
22 #include "ESP8266.h"
23
24 #define SSID          "ITEAD"
25 #define PASSWORD      "12345678"
26
27 ESP8266 wifi(Serial1);
28
29 void setup(void)
30 {
31     Serial.begin(9600);
32     Serial.print("setup begin\r\n");
33
34     Serial.print("FW Version:");
35
36     Serial.println(wifi.getVersion().c_str());
37
38     if (wifi.setOprToStationSoftAP()) {
39         Serial.print("to station + softap
40         ok\r\n");
41     } else {
42         Serial.print("to station + softap
43         err\r\n");
44     }
45 }
```

```
43     if (wifi.joinAP(SSID, PASSWORD)) {
44         Serial.print("Join AP success\r\n");
45         Serial.print("IP: ");
46
47         Serial.println(wifi.getLocalIP().c_str());
48     } else {
49         Serial.print("Join AP failure\r\n");
50     }
51
52     if (wifi.enableMUX()) {
53         Serial.print("multiple ok\r\n");
54     } else {
55         Serial.print("multiple err\r\n");
56     }
57
58     if (wifi.startTCPServer(8090)) {
59         Serial.print("start tcp server
ok\r\n");
60     } else {
61         Serial.print("start tcp server
err\r\n");
62     }
63
64     if (wifi.setTCPServerTimeout(10)) {
65         Serial.print("set tcp server timeout
10 seconds\r\n");
66     } else {
67         Serial.print("set tcp server timeout
err\r\n");
68     }
69
70     Serial.print("setup end\r\n");
71 }
72 void loop(void)
73 {
74     uint8_t buffer[128] = {0};
```

```
75     uint8_t mux_id;
76     uint32_t len = wifi.recv(&mux_id,
77     buffer, sizeof(buffer), 100);
78     if (len > 0) {
79         Serial.print("Status:[");
80         Serial.print(wifi.getIPStatus().c_str());
81         Serial.println("]");
82         Serial.print("Received from :");
83         Serial.print(mux_id);
84         Serial.print("[");
85         for(uint32_t i = 0; i < len; i++) {
86             Serial.print((char)buffer[i]);
87         }
88         Serial.print("]\r\n");
89
90         if(wifi.send(mux_id, buffer, len)) {
91             Serial.print("send back
ok\r\n");
92         } else {
93             Serial.print("send back
err\r\n");
94         }
95
96         if (wifi.releaseTCP(mux_id)) {
97             Serial.print("release tcp ");
98             Serial.print(mux_id);
99             Serial.println(" ok");
100        } else {
101            Serial.print("release tcp");
102            Serial.print(mux_id);
103            Serial.println(" err");
104        }
105
106        Serial.print("Status:[");
107
```

```
    Serial.print(wifi.getIPStatus().c_str());
108     Serial.println("]");
109 }
110 }
111 }
```

---

Generated on Thu Apr 9 2015 13:57:59 for API by [doxygen](#) 1.8.7



# API

## For Arduino developers

[Main Page](#)

[Related Pages](#)

[Classes](#)

[Files](#)

[Examples](#)

[examples](#) › [test](#) ›

## test Directory Reference

## Files

---

file **test.ino** [code]

---

Generated on Thu Apr 9 2015 13:57:59 for API by  doxygen 1.8.7



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)[File List](#)[File Members](#)

examples &gt; test &gt;

### test.ino

```
1
23 #include "ESP8266.h"
24
25 #define SSID          "ITEAD"
26 #define PASSWORD      "12345678"
27
28 ESP8266 wifi(Serial1,9600);
29
30 void setup(void)
31 {
32     Serial.begin(9600);
33     Serial.print("setup begin\r\n");
34     if(wifi.setUart(9600,2)){
35         Serial.println("set uart is ok ");
36     }
37     else{
38         Serial.println("set uart is error");
39     }
40
41     Serial.println(wifi.getWifiModeList().c_str())
42     ;
43     if(wifi.setOprToSoftAP(2,2)){
44         Serial.println("it is STA");
45     }
46     if(wifi.setOprToStation(2,2)){
```

```
45         Serial.println("it is AP");
46     }
47     if(wifi.setOprToStationSoftAP(2,2)){
48         Serial.println("it is AP+SoftAP");
49     }
50
51     Serial.println(wifi.getAPList().c_str());
52     wifi.joinAP(SSID,PASSWORD);
53
54     Serial.println(wifi.getNowConecAp(1).c_str());
55     if(wifi.leaveAP())
56     {
57         Serial.println("it is leave");
58     }
59     Serial.println(wifi.getNowConecAp(1).c_str());
60
61     if(wifi.setSoftAPPParam("aaa","12345678"))
62     {
63         Serial.println("it is set param
ok");
64     }
65     Serial.println(wifi.getSoftAPPParam());
66
67     Serial.println(wifi.getJoinedDeviceIP().c_str());
68
69     Serial.print("the state of DHCP:");
70     Serial.println(wifi.getDHCP().c_str());
71     if(wifi.setDHCP(2,1)){
72         Serial.println("it is set DHCP OK");
73     }
74     Serial.println(wifi.getDHCP().c_str());
75     if(wifi.setAutoConnect(0)){
76         Serial.println("take off auto connect
ok");
77     }
78
79     Serial.print("get the station mac: ");
```

```
74     Serial.println(wifi.getStationMac().c_str());
75
76     if(wifi.setStationMac("18:fe:35:98:d3:7b")){
77         Serial.println("set station mac is ok");
78     }
79     else {
80         Serial.println("it is error");
81     }
82     Serial.print("get the station mac: ");
83
84     Serial.println(wifi.getStationMac().c_str());
85
86
87     if(wifi.setStationIp("192.168.1.6", "192.168.1.
88     1", "255.255.255.0")){
89         Serial.println("set station's ip is
90         ok");
91     }
92     else{
93         Serial.println("set station's ip is
94         error");
95     }
96     Serial.print("get the station's ip");
97
98     Serial.println(wifi.getStationIp().c_str());
99
100    if(wifi.setAPIp("192.168.1.1")){
101        Serial.println("set ap's ip is ok");
102    }
```

```
101      else
102      {
103          Serial.println("set ap's is is
104              error");
105          }
106          Serial.print("get the ap's ap");
107          Serial.println(wifi.getAPIp().c_str());
108          if(wifi.startSmartConfig(1)){
109              Serial.println("start smartconfig is
110                  ok");
111          }
112          else{
113              Serial.println("start smartconfig is
114                  error");
115          }
116          if(wifi.stopSmartConfig()){
117              Serial.println("stop smartconfig is
118                  ok");
119          }
120          Serial.print("get the current status of
121              connection:");
122          Serial.println(wifi.getIPStatus().c_str());
123          if(wifi.saveTransLink(1, "192.168.1.18", 1006)){
124              Serial.println("save trans link is
125                  ok");
126          }
127          else{
128              Serial.println("save trans link is
129                  error");
130          }
```

```
128     Serial.println("setup end\r\n");
129 }
130
131 void loop(void)
132 {
133 }
134
```

---

Generated on Thu Apr 9 2015 13:57:59 for API by [doxygen](#) 1.8.7



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)

examples

UDPClientMultiple

## UDPClientMultiple Directory Reference

## Files

---

file **UDPClientMultiple.ino** [\[code\]](#)

---

Generated on Thu Apr 9 2015 13:57:59 for API by  doxygen 1.8.7



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)[File List](#)[File Members](#)

examples

UDPClientMultiple

## UDPClientMultiple.ino

```
1
22 #include "ESP8266.h"
23
24 #define SSID          "ITEAD"
25 #define PASSWORD      "12345678"
26 #define HOST_NAME     "172.16.5.12"
27 #define HOST_PORT     (5416)
28
29 ESP8266 wifi(Serial1);
30
31 void setup(void)
32 {
33     Serial.begin(9600);
34     Serial.print("setup begin\r\n");
35
36     Serial.print("FW Version:");
37
38     Serial.println(wifi.getVersion().c_str());
39
40     if (wifi.setOprToStationSoftAP()) {
41         Serial.print("to station + softap
42         ok\r\n");
43     } else {
44         Serial.print("to station + softap
45         err\r\n");
46 }
```

```
43     }
44
45     if (wifi.joinAP(SSID, PASSWORD)) {
46         Serial.print("Join AP success\r\n");
47         Serial.print("IP: ");
48
49         Serial.println(wifi.getLocalIP().c_str());
50     } else {
51         Serial.print("Join AP failure\r\n");
52     }
53
54     if (wifi.enableMUX()) {
55         Serial.print("multiple ok\r\n");
56     } else {
57         Serial.print("multiple err\r\n");
58     }
59
60     Serial.print("setup end\r\n");
61 }
62 void loop(void)
63 {
64     uint8_t buffer[128] = {0};
65     static uint8_t mux_id = 0;
66
67     if (wifi.registerUDP(mux_id, HOST_NAME,
68     HOST_PORT)) {
69         Serial.print("register udp ");
70         Serial.print(mux_id);
71         Serial.println(" ok");
72     } else {
73         Serial.print("register udp ");
74         Serial.print(mux_id);
75         Serial.println(" err");
76     }
77
    char *hello = "Hello, this is client!";
```

```
78     wifi.send(mux_id, (const uint8_t*)hello,
    strlen(hello));
79
80     uint32_t len = wifi.recv(mux_id, buffer,
    sizeof(buffer), 10000);
81     if (len > 0) {
82         Serial.print("Received:[");
83         for(uint32_t i = 0; i < len; i++) {
84             Serial.print((char)buffer[i]);
85         }
86         Serial.print("]\r\n");
87     }
88
89     if (wifi.unregisterUDP(mux_id)) {
90         Serial.print("unregister udp ");
91         Serial.print(mux_id);
92         Serial.println(" ok");
93     } else {
94         Serial.print("unregister udp ");
95         Serial.print(mux_id);
96         Serial.println(" err");
97     }
98     delay(5000);
99     mux_id++;
100    if (mux_id >= 5) {
101        mux_id = 0;
102    }
103}
104
```



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)

examples

UDPClientSingle

## UDPClientSingle Directory Reference

## Files

---

file **UDPClientSingle.ino** [\[code\]](#)

---

Generated on Thu Apr 9 2015 13:57:59 for API by  doxygen 1.8.7



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)[File List](#)[File Members](#)

examples

UDPClientSingle

## UDPClientSingle.ino

```
1
22 #include "ESP8266.h"
23
24 #define SSID          "ITEAD"
25 #define PASSWORD      "12345678"
26 #define HOST_NAME     "172.16.5.12"
27 #define HOST_PORT     (5416)
28
29 ESP8266 wifi(Serial1);
30
31 void setup(void)
32 {
33     Serial.begin(9600);
34     Serial.print("setup begin\r\n");
35
36     Serial.print("FW Version:");
37
38     Serial.println(wifi.getVersion().c_str());
39
40     if (wifi.setOprToStationSoftAP()) {
41         Serial.print("to station + softap
42         ok\r\n");
43     } else {
44         Serial.print("to station + softap
45         err\r\n");
46 }
```

```
43     }
44
45     if (wifi.joinAP(SSID, PASSWORD)) {
46         Serial.print("Join AP success\r\n");
47         Serial.print("IP: ");
48
49         Serial.println(wifi.getLocalIP().c_str());
50     } else {
51         Serial.print("Join AP failure\r\n");
52     }
53
54     if (wifi.disableMUX()) {
55         Serial.print("single ok\r\n");
56     } else {
57         Serial.print("single err\r\n");
58     }
59
60     Serial.print("setup end\r\n");
61 }
62 void loop(void)
63 {
64     uint8_t buffer[128] = {0};
65
66     if (wifi.registerUDP(HOST_NAME,
HOST_PORT)) {
67         Serial.print("register udp ok\r\n");
68     } else {
69         Serial.print("register udp
err\r\n");
70     }
71
72     char *hello = "Hello, this is client!";
73     wifi.send((const uint8_t*)hello,
strlen(hello));
74
75     uint32_t len = wifi.recv(buffer,
```

```
    sizeof(buffer), 10000);
76    if (len > 0) {
77        Serial.print("Received:[");
78        for(uint32_t i = 0; i < len; i++) {
79            Serial.print((char)buffer[i]);
80        }
81        Serial.print("]\r\n");
82    }
83
84    if (wifi.unregisterUDP()) {
85        Serial.print("unregister udp
ok\r\n");
86    } else {
87        Serial.print("unregister udp
err\r\n");
88    }
89    delay(5000);
90}
91
```



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)[File List](#)[File Members](#)

## doxygen.h File Reference

Define modules in API doc. More...

[Go to the source code of this file.](#)

## Detailed Description

---

Define modules in API doc.

### **Author**

Wu Pengfei (email:[pengfei.wu@itead.cc](mailto:pengfei.wu@itead.cc))

### **Date**

2014/11/20

### **Copyright**

Copyright (C) 2013-2014 ITEAD Intelligent Systems Co., Ltd.  
This program is free software; you can redistribute it and/or modify  
it under the terms of the GNU General Public License as published  
by the Free Software Foundation; either version 2 of the License,  
or (at your option) any later version.

Definition in file [doxygen.h](#).



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)[File List](#)[File Members](#)

## ESP8266.cpp File Reference

The implementation of class [ESP8266](#). More...

```
#include "ESP8266.h"
```

[Go to the source code of this file.](#)

# Detailed Description

---

The implementation of class **ESP8266**.

**Author**

Wu Pengfei [pengfei.wu@itead.cc](mailto:pengfei.wu@itead.cc)

**Date**

2015.02

**Copyright:**

Copyright (c) 2015 ITEAD Intelligent Systems Co., Ltd.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Definition in file **ESP8266.cpp**.



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)[File List](#)[File Members](#)[Classes](#) | [Macros](#)

## ESP8266.h File Reference

The definition of class [ESP8266](#). More...

```
#include "Arduino.h"
```

[Go to the source code of this file.](#)

## Classes

---

class **ESP8266**

Provide an easy-to-use way to manipulate **ESP8266**. More...

---

## Macros

---

```
#define USER_SEL_VERSION VERSION_18
```

You can modify the macro to choose a different version.

---

## Detailed Description

---

The definition of class **ESP8266**.

**Author**

Wu Pengfei [pengfei.wu@itead.cc](mailto:pengfei.wu@itead.cc)

**Date**

2015.02

**Copyright:**

Copyright (c) 2015 ITEAD Intelligent Systems Co., Ltd.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Definition in file **ESP8266.h**.



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)[File List](#)[File Members](#)[All](#)[Macros](#)

Here is a list of all documented file members with links to the documentation:

- USER\_SEL\_VERSION : [ESP8266.h](#)

---

Generated on Thu Apr 9 2015 13:57:59 for API by [doxygen](#) 1.8.7



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)[File List](#)[File Members](#)[All](#)[Macros](#)

- USER\_SEL\_VERSION : [ESP8266.h](#)

---

Generated on Thu Apr 9 2015 13:57:59 for API by [doxygen](#) 1.8.7



# API

## For Arduino developers

Main Page

Related Pages

Classes

Files

Examples

## Examples

Here is a list of all examples:

- [ConnectWiFi.ino](#)
- [HTTPGET.ino](#)
- [TCPClientMultiple.ino](#)
- [TCPClientSingle.ino](#)
- [TCPClientSingleUNO.ino](#)
- [TCPServer.ino](#)
- [test.ino](#)
- [UDPClientMultiple.ino](#)
- [UDPClientSingle.ino](#)



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)

## ConnectWiFi.ino

The ConnectWiFi demo of library WeeESP8266.

**Author**

Wu Pengfei [pengfei.wu@itead.cc](mailto:pengfei.wu@itead.cc)

**Date**

2015.03

**Copyright:**

Copyright (c) 2015 ITEAD Intelligent Systems Co., Ltd.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

```
#include "ESP8266.h"  
  
#define SSID "ITEAD"
```

```
#define PASSWORD      "12345678"

ESP8266 wifi(Serial1);

void setup(void)
{
    Serial.begin(9600);
    Serial.print("setup begin\r\n");
    Serial.print("FW Version: ");
    Serial.println(wifi.getVersion().c_str());

    if (wifi.setOprToStation()) {
        Serial.print("to station ok\r\n");
    } else {
        Serial.print("to station err\r\n");
    }

    if (wifi.joinAP(SSID, PASSWORD)) {
        Serial.print("Join AP success\r\n");
        Serial.print("IP: ");
        Serial.println(wifi.getLocalIP().c_str());
    } else {
        Serial.print("Join AP failure\r\n");
    }

    Serial.print("setup end\r\n");
}

void loop(void)
{
```



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)

## HTTPGET.ino

The HTTPGET demo of library WeeESP8266.

**Author**

Wu Pengfei [pengfei.wu@itead.cc](mailto:pengfei.wu@itead.cc)

**Date**

2015.03

**Copyright:**

Copyright (c) 2015 ITEAD Intelligent Systems Co., Ltd.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

```
#include "ESP8266.h"  
  
#define SSID "ITEAD"
```

```
#define PASSWORD      "12345678"
#define HOST_NAME     "www.baidu.com"
#define HOST_PORT      (80)

ESP8266 wifi(Serial1);

void setup(void)
{
    Serial.begin(9600);
    Serial.print("setup begin\r\n");

    Serial.print("FW Version:");
    Serial.println(wifi.getVersion().c_str());

    if (wifi.setOprToStationSoftAP()) {
        Serial.print("to station + softap
ok\r\n");
    } else {
        Serial.print("to station + softap
err\r\n");
    }

    if (wifi.joinAP(SSID, PASSWORD)) {
        Serial.print("Join AP success\r\n");

        Serial.print("IP:");
        Serial.println(
wifi.getLocalIP().c_str());
    } else {
        Serial.print("Join AP failure\r\n");
    }

    if (wifi.disableMUX()) {
        Serial.print("single ok\r\n");
    } else {
        Serial.print("single err\r\n");
    }

    Serial.print("setup end\r\n");
}
```

```

void loop(void)
{
    uint8_t buffer[1024] = {0};

    if (wifi.createTCP(HOST_NAME, HOST_PORT)) {
        Serial.print("create tcp ok\r\n");
    } else {
        Serial.print("create tcp err\r\n");
    }

    char *hello = "GET / HTTP/1.1\r\nHost:
www.baidu.com\r\nConnection: close\r\n\r\n";
    wifi.send((const uint8_t*)hello,
    strlen(hello));

    uint32_t len = wifi.recv(buffer,
    sizeof(buffer), 10000);
    if (len > 0) {
        Serial.print("Received:[");
        for(uint32_t i = 0; i < len; i++) {
            Serial.print((char)buffer[i]);
        }
        Serial.print("]\r\n");
    }

    if (wifi.releaseTCP()) {
        Serial.print("release tcp ok\r\n");
    } else {
        Serial.print("release tcp err\r\n");
    }

    while(1);
}

```



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)

## TCPCClientMultiple.ino

The TCPCClientMultiple demo of library WeeESP8266.

**Author**

Wu Pengfei [pengfei.wu@itead.cc](mailto:pengfei.wu@itead.cc)

**Date**

2015.02

**Copyright:**

Copyright (c) 2015 ITEAD Intelligent Systems Co., Ltd.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

```
#include "ESP8266.h"  
  
#define SSID "ITEAD"
```

```
#define PASSWORD      "12345678"
#define HOST_NAME     "192.168.1.1"
#define HOST_PORT      (8090)

ESP8266 wifi(Serial1);

void setup(void)
{
    Serial.begin(9600);
    Serial.print("setup begin\r\n");

    Serial.print("FW Version: ");
    Serial.println(wifi.getVersion().c_str());

    if (wifi.setOprToStationSoftAP()) {
        Serial.print("to station + softap
ok\r\n");
    } else {
        Serial.print("to station + softap
err\r\n");
    }

    if (wifi.joinAP(SSID, PASSWORD)) {
        Serial.print("Join AP success\r\n");
        Serial.print("IP: ");
        Serial.println(wifi.getLocalIP().c_str());
    } else {
        Serial.print("Join AP failure\r\n");
    }

    if (wifi.enableMUX()) {
        Serial.print("multiple ok\r\n");
    } else {
        Serial.print("multiple err\r\n");
    }

    Serial.print("setup end\r\n");
}
```

```
void loop(void)
{
    uint8_t buffer[128] = {0};
    static uint8_t mux_id = 0;

    if (wifi.createTCP(mux_id, HOST_NAME, HOST_PORT))
    {
        Serial.print("create tcp ");
        Serial.print(mux_id);
        Serial.println(" ok");
    } else {
        Serial.print("create tcp ");
        Serial.print(mux_id);
        Serial.println(" err");
    }

    char *hello = "Hello, this is client!";
    if (wifi.send(mux_id, (const uint8_t*)hello,
        strlen(hello))) {
        Serial.println("send ok");
    } else {
        Serial.println("send err");
    }

    uint32_t len = wifi.recv(mux_id, buffer,
        sizeof(buffer), 10000);
    if (len > 0) {
        Serial.print("Received:[");
        for(uint32_t i = 0; i < len; i++) {
            Serial.print((char)buffer[i]);
        }
        Serial.print("]\r\n");
    }

    if (wifi.releaseTCP(mux_id)) {
        Serial.print("release tcp ");
    }
}
```

```
    Serial.print(mux_id);
    Serial.println(" ok");
} else {
    Serial.print("release tcp ");
    Serial.print(mux_id);
    Serial.println(" err");
}

delay(3000);
mux_id++;
if (mux_id >= 5) {
    mux_id = 0;
}
}
```



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)

## TCPClientSingle.ino

The TCPClientSingle demo of library WeeESP8266.

**Author**

Wu Pengfei [pengfei.wu@itead.cc](mailto:pengfei.wu@itead.cc)

**Date**

2015.02

**Copyright:**

Copyright (c) 2015 ITEAD Intelligent Systems Co., Ltd.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

```
#include "ESP8266.h"  
  
#define SSID "ITEAD"
```

```
#define PASSWORD      "12345678"
#define HOST_NAME     "172.16.5.12"
#define HOST_PORT      (8090)

ESP8266 wifi(Serial1);

void setup(void)
{
    Serial.begin(9600);
    Serial.print("setup begin\r\n");

    Serial.print("FW Version:");
    Serial.println(wifi.getVersion().c_str());

    if (wifi.setOprToStationSoftAP()) {
        Serial.print("to station + softap
ok\r\n");
    } else {
        Serial.print("to station + softap
err\r\n");
    }

    if (wifi.joinAP(SSID, PASSWORD)) {
        Serial.print("Join AP success\r\n");
        Serial.print("IP:");
        Serial.println(
            wifi.getLocalIP().c_str());
    } else {
        Serial.print("Join AP failure\r\n");
    }

    if (wifi.disableMUX()) {
        Serial.print("single ok\r\n");
    } else {
        Serial.print("single err\r\n");
    }

    Serial.print("setup end\r\n");
}
```

```
}

void loop(void)
{
    uint8_t buffer[128] = {0};

    if (wifi.createTCP(HOST_NAME, HOST_PORT)) {
        Serial.print("create tcp ok\r\n");
    } else {
        Serial.print("create tcp err\r\n");
    }

    char *hello = "Hello, this is client!";
    wifi.send((const uint8_t*)hello,
              strlen(hello));

    uint32_t len = wifi.recv(buffer,
                            sizeof(buffer), 10000);
    if (len > 0) {
        Serial.print("Received:[");
        for(uint32_t i = 0; i < len; i++) {
            Serial.print((char)buffer[i]);
        }
        Serial.print("]\r\n");
    }

    if (wifi.releaseTCP()) {
        Serial.print("release tcp ok\r\n");
    } else {
        Serial.print("release tcp err\r\n");
    }
    delay(5000);
}
```



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)

## TCPClientSingleUNO.ino

The TCPClientSingleUNO demo of library WeeESP8266.

**Author**

Wu Pengfei [pengfei.wu@itead.cc](mailto:pengfei.wu@itead.cc)

**Date**

2015.03

**Copyright:**

Copyright (c) 2015 ITEAD Intelligent Systems Co., Ltd.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

```
#include "ESP8266.h"
#include <SoftwareSerial.h>
```

```
#define SSID          "ITEAD"
#define PASSWORD       "12345678"
#define HOST_NAME      "172.16.5.12"
#define HOST_PORT      (8090)

SoftwareSerial mySerial(3, 2); /* RX:D3, TX:D2 */
ESP8266 wifi(mySerial);

void setup(void)
{
    Serial.begin(9600);
    Serial.print("setup begin\r\n");

    Serial.print("FW Version:");
    Serial.println(wifi.getVersion().c_str());

    if (wifi.setOprToStationSoftAP()) {
        Serial.print("to station + softap
ok\r\n");
    } else {
        Serial.print("to station + softap
err\r\n");
    }

    if (wifi.joinAP(SSID, PASSWORD)) {
        Serial.print("Join AP success\r\n");
        Serial.print("IP:");
        Serial.println(
            wifi.getLocalIP().c_str());
    } else {
        Serial.print("Join AP failure\r\n");
    }

    if (wifi.disableMUX()) {
        Serial.print("single ok\r\n");
    } else {
        Serial.print("single err\r\n");
    }
}
```

```
        Serial.print("setup end\r\n");
    }

void loop(void)
{
    uint8_t buffer[128] = {0};

    if (wifi.createTCP(HOST_NAME, HOST_PORT)) {
        Serial.print("create tcp ok\r\n");
    } else {
        Serial.print("create tcp err\r\n");
    }

    char *hello = "Hello, this is client!";
    wifi.send((const uint8_t*)hello,
              strlen(hello));

    uint32_t len = wifi.recv(buffer,
                            sizeof(buffer), 10000);
    if (len > 0) {
        Serial.print("Received:[");
        for(uint32_t i = 0; i < len; i++) {
            Serial.print((char)buffer[i]);
        }
        Serial.print("]\r\n");
    }

    if (wifi.releaseTCP()) {
        Serial.print("release tcp ok\r\n");
    } else {
        Serial.print("release tcp err\r\n");
    }
    delay(5000);
}
```

Generated on Thu Apr 9 2015 13:57:59 for API by [doxygen](#) 1.8.7



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)

## TCPServer.ino

The TCPServer demo of library WeeESP8266.

**Author**

Wu Pengfei [pengfei.wu@itead.cc](mailto:pengfei.wu@itead.cc)

**Date**

2015.02

**Copyright:**

Copyright (c) 2015 ITEAD Intelligent Systems Co., Ltd.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

```
#include "ESP8266.h"  
  
#define SSID "ITEAD"
```

```
#define PASSWORD      "12345678"

ESP8266 wifi(Serial1);

void setup(void)
{
    Serial.begin(9600);
    Serial.print("setup begin\r\n");

    Serial.print("FW Version:");
    Serial.println(wifi.getVersion().c_str());

    if (wifi.setOprToStationSoftAP()) {
        Serial.print("to station + softap
ok\r\n");
    } else {
        Serial.print("to station + softap
err\r\n");
    }

    if (wifi.joinAP(SSID, PASSWORD)) {
        Serial.print("Join AP success\r\n");
        Serial.print("IP: ");
        Serial.println(wifi.getLocalIP().c_str());
    } else {
        Serial.print("Join AP failure\r\n");
    }

    if (wifi.enableMUX()) {
        Serial.print("multiple ok\r\n");
    } else {
        Serial.print("multiple err\r\n");
    }

    if (wifi.startTCPServer(8090)) {
        Serial.print("start tcp server ok\r\n");
    } else {
        Serial.print("start tcp server err\r\n");
    }
}
```

```
}

if (wifi.setTCPServerTimeout(10)) {
    Serial.print("set tcp server timout 10
seconds\r\n");
} else {
    Serial.print("set tcp server timeout
err\r\n");
}

Serial.print("setup end\r\n");
}

void loop(void)
{
    uint8_t buffer[128] = {0};
    uint8_t mux_id;
    uint32_t len = wifi.recv(&mux_id, buffer,
                           sizeof(buffer), 100);
    if (len > 0) {
        Serial.print("Status:[");
        Serial.print(wifi.getIPStatus().c_str());
        Serial.println("]");

        Serial.print("Received from :");
        Serial.print(mux_id);
        Serial.print("[");
        for(uint32_t i = 0; i < len; i++) {
            Serial.print((char)buffer[i]);
        }
        Serial.print("]\r\n");

        if(wifi.send(mux_id, buffer, len)) {
            Serial.print("send back ok\r\n");
        } else {
            Serial.print("send back err\r\n");
        }
    }
}
```

```
if (wifi.releaseTCP(mux_id)) {
    Serial.print("release tcp ");
    Serial.print(mux_id);
    Serial.println(" ok");
} else {
    Serial.print("release tcp");
    Serial.print(mux_id);
    Serial.println(" err");
}

Serial.print("Status:[");
Serial.print(wifi.getIPStatus().c_str());
Serial.println("]");
}
}
```



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)

## test.ino

The test demo of library WeeESP8266.

**Author**

Wu Pengfei [pengfei.wu@itead.cc](mailto:pengfei.wu@itead.cc)

**Date**

2015.03

**Copyright:**

Copyright (c) 2015 ITEAD Intelligent Systems Co., Ltd.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

```
#include "ESP8266.h"  
  
#define SSID "ITEAD"
```

```
#define PASSWORD      "12345678"

ESP8266 wifi(Serial1,9600);

void setup(void)
{
    Serial.begin(9600);
    Serial.print("setup begin\r\n");
    if(wifi.setUart(9600,2)){
        Serial.println("set uart is ok ");
    }
    else{
        Serial.println("set uart is error");
    }
    Serial.println(wifi.getWifiModeList().c_str());
    if(wifi.setOprToSoftAP(2,2)){
        Serial.println("it is STA");
    }
    if(wifi.setOprToStation(2,2)){
        Serial.println("it is AP");
    }
    if(wifi.setOprToStationSoftAP(2,2)){
        Serial.println("it is AP+SoftAP");
    }
    Serial.println(wifi.getAPList().c_str());
    wifi.joinAP(SSID,PASSWORD);

    Serial.println(wifi.getNowConecAp(1).c_str());
    if(wifi.leaveAP())
    {
        Serial.println("it is leave");
    }

    Serial.println(wifi.getNowConecAp(1).c_str());
    if(wifi.setSoftAPPParam("aaa","12345678"))
    {
        Serial.println("it is set param ok");
    }
```

```
Serial.println(wifi.getSoftAPParam());

Serial.println(wifi.getJoinedDeviceIP().c_str());
    Serial.print("the state of DHCP:");
    Serial.println(wifi.getDHCP().c_str());
if(wifi.setDHCP(2,1)){
    Serial.println("it is set DHCP OK");
}
Serial.println(wifi.getDHCP().c_str());
if(wifi.setAutoConnect(0)){
    Serial.println("take off auto connect ok");
}
Serial.print("get the station mac: ");
Serial.println(wifi.getStationMac().c_str());
if(wifi.setStationMac("18:fe:35:98:d3:7b")){
    Serial.println("set station mac is ok ");
}
else {
    Serial.println("it is error");
}
Serial.print("get the station mac: ");
Serial.println(wifi.getStationMac().c_str());
Serial.print("get the station's ip");
Serial.println(wifi.getStationIp().c_str());

if(wifi.setStationIp("192.168.1.6", "192.168.1.1",
"255.255.255.0")){
    Serial.println("set station's ip is ok");
}
else{
    Serial.println("set station's ip is error");
}
    Serial.print("get the station's ip");
    Serial.println(wifi.getStationIp().c_str());

    Serial.print("get the ap's ap");
```

```
    Serial.println(wifi.getAPIp().c_str());\n\n    if(wifi.setAPIp("192.168.1.1")){\n        Serial.println("set ap's ip is ok");\n    }\n    else{\n        {\n            Serial.println("set ap's is is error");\n        }\n        Serial.print("get the ap's ap");\n        Serial.println(wifi.getAPIp().c_str());\n\n        if(wifi.startSmartConfig(1)){\n            Serial.println("start smartconfig is ok");\n        }\n        else{\n            Serial.println("start smartconfig is\nerror");\n        }\n        if(wifi.stopSmartConfig()){\n            Serial.println("stop smartconfig is ok");\n        }\n        else{\n            Serial.println("stop smartconfig is\nerror");\n        }\n        Serial.print("get the current status of\nconnection:");\n        Serial.println(wifi.getIPStatus().c_str());\n        if(wifi.saveTransLink(1,"192.168.1.18",1006)){\n            Serial.println("save trans link is ok");\n        }\n        else{\n            Serial.println("save trans link is error");\n        }\n        Serial.println("setup end\r\n");\n    }\n}
```

```
void loop(void)
{
}
```

---

Generated on Thu Apr 9 2015 13:57:59 for API by [doxygen](#) 1.8.7



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)

## UDPClientMultiple.ino

The UDPClientMultiple demo of library WeeESP8266.

**Author**

Wu Pengfei [pengfei.wu@itead.cc](mailto:pengfei.wu@itead.cc)

**Date**

2015.02

**Copyright:**

Copyright (c) 2015 ITEAD Intelligent Systems Co., Ltd.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

```
#include "ESP8266.h"  
  
#define SSID "ITEAD"
```

```
#define PASSWORD      "12345678"
#define HOST_NAME     "172.16.5.12"
#define HOST_PORT      (5416)

ESP8266 wifi(Serial1);

void setup(void)
{
    Serial.begin(9600);
    Serial.print("setup begin\r\n");

    Serial.print("FW Version:");
    Serial.println(wifi.getVersion().c_str());

    if (wifi.setOprToStationSoftAP()) {
        Serial.print("to station + softap
ok\r\n");
    } else {
        Serial.print("to station + softap
err\r\n");
    }

    if (wifi.joinAP(SSID, PASSWORD)) {
        Serial.print("Join AP success\r\n");
        Serial.print("IP: ");
        Serial.println(wifi.getLocalIP().c_str());
    } else {
        Serial.print("Join AP failure\r\n");
    }

    if (wifi.enableMUX()) {
        Serial.print("multiple ok\r\n");
    } else {
        Serial.print("multiple err\r\n");
    }

    Serial.print("setup end\r\n");
}
```

```
}

void loop(void)
{
    uint8_t buffer[128] = {0};
    static uint8_t mux_id = 0;

    if (wifi.registerUDP(mux_id, HOST_NAME,
        HOST_PORT)) {
        Serial.print("register udp ");
        Serial.print(mux_id);
        Serial.println(" ok");
    } else {
        Serial.print("register udp ");
        Serial.print(mux_id);
        Serial.println(" err");
    }

    char *hello = "Hello, this is client!";
    wifi.send(mux_id, (const uint8_t*)hello,
        strlen(hello));

    uint32_t len = wifi.recv(mux_id, buffer,
        sizeof(buffer), 10000);
    if (len > 0) {
        Serial.print("Received:[");
        for(uint32_t i = 0; i < len; i++) {
            Serial.print((char)buffer[i]);
        }
        Serial.print("]\r\n");
    }

    if (wifi.unregisterUDP(mux_id)) {
        Serial.print("unregister udp ");
        Serial.print(mux_id);
        Serial.println(" ok");
    } else {
```

```
    Serial.print("unregister udp ");
    Serial.print(mux_id);
    Serial.println(" err");
}
delay(5000);
mux_id++;
if (mux_id >= 5) {
    mux_id = 0;
}
}
```



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)

## UDPClientSingle.ino

The UDPClientSingle demo of library WeeESP8266.

**Author**

Wu Pengfei [pengfei.wu@itead.cc](mailto:pengfei.wu@itead.cc)

**Date**

2015.02

**Copyright:**

Copyright (c) 2015 ITEAD Intelligent Systems Co., Ltd.

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

```
#include "ESP8266.h"  
  
#define SSID "ITEAD"
```

```
#define PASSWORD      "12345678"
#define HOST_NAME     "172.16.5.12"
#define HOST_PORT      (5416)

ESP8266 wifi(Serial1);

void setup(void)
{
    Serial.begin(9600);
    Serial.print("setup begin\r\n");

    Serial.print("FW Version:");
    Serial.println(wifi.getVersion().c_str());

    if (wifi.setOprToStationSoftAP()) {
        Serial.print("to station + softap
ok\r\n");
    } else {
        Serial.print("to station + softap
err\r\n");
    }

    if (wifi.joinAP(SSID, PASSWORD)) {
        Serial.print("Join AP success\r\n");
        Serial.print("IP: ");
        Serial.println(wifi.getLocalIP().c_str());
    } else {
        Serial.print("Join AP failure\r\n");
    }

    if (wifi.disableMUX()) {
        Serial.print("single ok\r\n");
    } else {
        Serial.print("single err\r\n");
    }

    Serial.print("setup end\r\n");
}
```

```

}

void loop(void)
{
    uint8_t buffer[128] = {0};

    if (wifi.registerUDP(HOST_NAME, HOST_PORT)) {
        Serial.print("register udp ok\r\n");
    } else {
        Serial.print("register udp err\r\n");
    }

    char *hello = "Hello, this is client!";
    wifi.send((const uint8_t*)hello,
              strlen(hello));

    uint32_t len = wifi.recv(buffer,
                            sizeof(buffer), 10000);
    if (len > 0) {
        Serial.print("Received:[");
        for(uint32_t i = 0; i < len; i++) {
            Serial.print((char)buffer[i]);
        }
        Serial.print("]\r\n");
    }

    if (wifi.unregisterUDP()) {
        Serial.print("unregister udp ok\r\n");
    } else {
        Serial.print("unregister udp err\r\n");
    }
    delay(5000);
}

```



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)

## Related Pages

Here is a list of all related documentation pages:

[README](#)

Generated on Thu Apr 9 2015 13:57:59 for API by [doxygen](#) 1.8.7



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)[Class List](#)[Class Index](#)[Class Members](#)

## ESP8266 Member List

This is the complete list of members for [ESP8266](#), including all inherited members.

[createTCP\(String addr, uint32\\_t port\)](#)

[createTCP\(uint8\\_t mux\\_id, String addr, uint32\\_t port\)](#)

[deepSleep\(uint32\\_t time\)](#)

[disableMUX\(void\)](#)

[enableMUX\(void\)](#)

[getAPIp\(uint8\\_t pattern=3\)](#)

[getAPList\(void\)](#)

[getDHCP\(uint8\\_t pattern=3\)](#)

[getIPStatus\(void\)](#)

[getJoinedDeviceIP\(void\)](#)

[getLocalIP\(void\)](#)

[getNowConecAp\(uint8\\_t pattern=3\)](#)

[getSoftAPPParam\(uint8\\_t pattern=3\)](#)

[getStationIp\(uint8\\_t pattern=3\)](#)

[getStationMac\(uint8\\_t pattern=3\)](#)

[getVersion\(void\)](#)

[getWifiModeList\(void\)](#)

[joinAP\(String ssid, String pwd, uint8\\_t pattern=3\)](#)

[kick\(void\)](#)

[leaveAP\(void\)](#)

```
recv(uint8_t *buffer, uint32_t buffer_size, uint32_t timeout=1000)
recv(uint8_t mux_id, uint8_t *buffer, uint32_t buffer_size, uint32_t timeout)
recv(uint8_t *coming_mux_id, uint8_t *buffer, uint32_t buffer_size, uint32_t timeout)
registerUDP(String addr, uint32_t port)
registerUDP(uint8_t mux_id, String addr, uint32_t port)
releaseTCP(void)
releaseTCP(uint8_t mux_id)
restart(void)
restore(void)
saveTransLink(uint8_t mode, String ip, uint32_t port)
send(const uint8_t *buffer, uint32_t len)
send(uint8_t mux_id, const uint8_t *buffer, uint32_t len)
setAPIp(String ip, uint8_t pattern=3)
setAutoConnect(uint8_t en)
setCIPMODE(uint8_t mode)
setDHCP(uint8_t mode, uint8_t en, uint8_t pattern=3)
setEcho(uint8_t mode)
setOprToSoftAP(uint8_t pattern1=3, uint8_t pattern2=3)
setOprToStation(uint8_t pattern1=3, uint8_t pattern2=3)
setOprToStationSoftAP(uint8_t pattern1=3, uint8_t pattern2=3)
setPing(String ip)
setSoftAPPParam(String ssid, String pwd, uint8_t chl=7, uint8_t ecn=4, uint8_t auth=2)
setStationIp(String ip, String gateway, String netmask, uint8_t pattern=3)
setStationMac(String mac, uint8_t pattern=3)
setTCPServerTimeout(uint32_t timeout=180)
setUart(uint32_t baudrate, uint8_t pattern)
startServer(uint32_t port=333)
startSmartConfig(uint8_t type)
startTCPServer(uint32_t port=333)
stopServer(void)
stopSmartConfig(void)
```

**stopTCPServer(void)**

**unregisterUDP(void)**

**unregisterUDP(uint8\_t mux\_id)**

---

Generated on Thu Apr 9 2015 13:57:59 for API by [doxygen](#) 1.8.7



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)[File List](#)[File Members](#)

## ESP8266.h

[Go to the documentation of this file.](#)

```
1
21 ifndef __ESP8266_H__
22 define __ESP8266_H__
23
24 #include "Arduino.h"
25
26 #ifdef ESP8266_USE_SOFTWARE_SERIAL
27 #include "SoftwareSerial.h"
28 #endif
29
30 #define VERSION_18 0X18
31 #define VERSION_22 0X22
32
37 #define USER_SEL_VERSION VERSION_18
38
42 class ESP8266 {
43   public:
44
45 #ifdef ESP8266_USE_SOFTWARE_SERIAL
46   /*
47     * Constuctor.
48     *
49     * @param uart - an reference of
SoftwareSerial object.
50     * @param baud - the buad rate to
communicate with ESP8266(default:9600).
```

```
51      *
52      * @warning parameter baud depends on
53      * the AT firmware. 9600 is an common value.
53      */
54 #if (USER_SEL_VERSION == VERSION_22)
55     ESP8266(SoftwareSerial &uart, uint32_t
56     baud = 115200);
56 #elif (USER_SEL_VERSION == VERSION_18)
57     ESP8266(SoftwareSerial &uart, uint32_t
58     baud = 9600);
58 #endif /* #if(USER_SEL_VERSION==VERSION_22)
59 */
60 #else /* HardwareSerial */
61     /*
62     * Constuctor.
63     *
64     * @param uart - an reference of
65     * HardwareSerial object.
65     * @param baud - the buad rate to
66     * communicate with ESP8266(default:9600).
66     *
67     * @warning parameter baud depends on
68     * the AT firmware. 9600 is an common value.
68     */
69 #if (USER_SEL_VERSION == VERSION_22)
70     ESP8266(HardwareSerial &uart, uint32_t
71     baud = 115200);
71 #elif (USER_SEL_VERSION == VERSION_18)
72     ESP8266(HardwareSerial &uart, uint32_t
73     baud = 9600);
73 #endif /* #if(USER_SEL_VERSION ==
74 VERSION_22) */
74
75
76#endif /* #ifdef ESP8266_USE_SOFTWARE_SERIAL
76 */
```

```
77
78
87     bool kick(void);
88
97     bool restart(void);
98
104    String getVersion(void);
105
114    bool deepSleep(uint32_t time);
115
124    bool setEcho(uint8_t mode);
125
132    bool restore(void);
133
143    bool setUart(uint32_t baudrate,uint8_t
pattern);
144
154    bool setOprToStation(uint8_t
pattern1=3,uint8_t pattern2=3);
155
161    String getWifiModeList(void);
162
171    bool setOprToSoftAP(uint8_t
pattern1=3,uint8_t pattern2=3);
172
181    bool setOprToStationSoftAP(uint8_t
pattern1=3,uint8_t pattern2=3);
182
190    String getAPList(void);
191
198    String getNowConecAp(uint8_t pattern=3);
199
210    bool joinAP(String ssid, String
pwd,uint8_t pattern=3);
211
218    bool leaveAP(void);
219
```

```
233     bool setSoftAPPParam(String ssid, String
  pwd, uint8_t chl = 7, uint8_t ecn = 4, uint8_t
  pattern=3);
234
241     String getSoftAPPParam(uint8_t
  pattern=3);
242
249     String getJoinedDeviceIP(void);
250
258     String getDHCP(uint8_t pattern=3);
259
268     bool setDHCP(uint8_t mode, uint8_t en,
  uint8_t pattern=3);
269
276     bool setAutoConnect(uint8_t en);
277
284     String getStationMac(uint8_t
  pattern=3);
285
293     bool setStationMac(String mac, uint8_t
  pattern=3);
294
301     String getStationIp(uint8_t pattern=3);
302
313     bool setStationIp(String ip, String
  gateway, String netmask, uint8_t pattern=3);
314
322     String getAPIp(uint8_t pattern=3);
323
332     bool setAPIp(String ip, uint8_t
  pattern=3);
333
340     bool startSmartConfig(uint8_t type);
341
348     bool stopSmartConfig(void);
349
355     String getIPStatus(void);
```

```
356
362     String getLocalIP(void);
363
373     bool enableMUX(void);
374
383     bool disableMUX(void);
384
393     bool createTCP(String addr, uint32_t
    port);
394
401     bool releaseTCP(void);
402
411     bool registerUDP(String addr, uint32_t
    port);
412
419     bool unregisterUDP(void);
420
430     bool createTCP(uint8_t mux_id, String
    addr, uint32_t port);
431
439     bool releaseTCP(uint8_t mux_id);
440
450     bool registerUDP(uint8_t mux_id, String
    addr, uint32_t port);
451
459     bool unregisterUDP(uint8_t mux_id);
460
468     bool setTCPServerTimeout(uint32_t
    timeout = 180);
469
485     bool startTCPServer(uint32_t port =
    333);
486
493     bool stopTCPServer(void);
494
501     bool setCIPMODE(uint8_t mode);
502
```

```
513     bool startServer(uint32_t port = 333);
514
521     bool stopServer(void);
528     bool saveTransLink (uint8_t mode, String
      ip,uint32_t port);
529
536     bool setPing(String ip);
537
546     bool send(const uint8_t *buffer,
      uint32_t len);
547
557     bool send(uint8_t mux_id, const uint8_t
      *buffer, uint32_t len);
558
567     uint32_t recv(uint8_t *buffer, uint32_t
      buffer_size, uint32_t timeout = 1000);
568
578     uint32_t recv(uint8_t mux_id, uint8_t
      *buffer, uint32_t buffer_size, uint32_t
      timeout = 1000);
579
592     uint32_t recv(uint8_t *coming_mux_id,
      uint8_t *buffer, uint32_t buffer_size,
      uint32_t timeout = 1000);
593
594     private:
595
596     /*
597     * Empty the buffer or UART RX.
598     */
599     void rx_empty(void);
600
601     /*
602     * Recvive data from uart. Return all
      received data if target found or timeout.
603     */
604     String recvString(String target,
```

```
        uint32_t timeout = 1000);
605    /*
606     * Recvive data from uart. Return all
607     received data if one of target1 and target2
608     found or timeout.
609     */
610
611    /*
612     * Recvive data from uart. Return all
613     received data if one of target1, target2 and
614     target3 found or timeout.
615     */
616    /*
617     * Recvive data from uart and search
618     first target. Return true if target found,
619     false for timeout.
620     */
621    /*
622     * Recvive data from uart and search
623     first target and cut out the substring between
624     begin and end(excluding begin and end self).
625     * Return true if target found, false
626     for timeout.
627     */
628    bool recvFind(String target, uint32_t
629      timeout = 1000);
630
631    /*
632     * Recvive data from uart and search
633     first target and cut out the substring between
634     begin and end(excluding begin and end self).
635     * Return true if target found, false
636     for timeout.
637     */
638    bool recvFindAndFilter(String target,
639      String begin, String end, String &data,
640      uint32_t timeout = 1000);
```

```
626
627     /*
628      * Receive a package from uart.
629      *
630      * @param buffer - the buffer storing
631      * @param buffer_size - guess what!
632      * @param data_len - the length of data
633      * actually received(maybe more than buffer_size,
634      * the remained data will be abandoned).
635      * @param timeout - the duration
636      * waiting data comming.
637      * @param coming_mux_id - in single
638      * connection mode, should be NULL and not NULL
639      * in multiple.
640
641      */
642
643     uint32_t recvPkg(uint8_t *buffer,
644                      uint32_t buffer_size, uint32_t *data_len,
645                      uint32_t timeout, uint8_t *coming_mux_id);
646
647
648     bool eAT(void);
649     bool eATRST(void);
650     bool eATGMR(String &version);
651     bool eATGSLP(uint32_t time);
652     bool eATE(uint8_t mode);
653     bool eATRESTORE(void);
654     bool eATSETUART(uint32_t
655                      baudrate,uint8_t pattern);
656
657     bool qATCWMODE(uint8_t *mode,uint8_t
658                    pattern=3);
659     bool eATCWMODE(String &list) ;
660     bool SATCWMODE(uint8_t mode,uint8_t
661                    pattern=3);
662     bool qATCWJAP(String &ssid,uint8_t
663                    pattern=3) ;
```

```
651     bool sATCWJAP(String ssid, String
652         pwd,uint8_t pattern=3);
653     bool eATCWLAP(String &list);
654     bool eATCWQAP(void);
655     bool sATCWSAP(String ssid, String pwd,
656         uint8_t chl, uint8_t ecn,uint8_t pattern=3);
657     bool eATCWLIF(String &list);
658     bool qATCWDHCP(String &List,uint8_t
659         pattern=3);
660     bool sATCWDHCP(uint8_t mode, uint8_t en,
661         uint8_t pattern=3);
662     bool eATCWAUTOCONN(uint8_t en);
663     bool qATCIPSTAMAC(String &mac,uint8_t
664         pattern=3);
665     bool eATCIPSTAMAC(String mac,uint8_t
666         pattern=3);
667     bool qATCIPSTAIP(String &ip,uint8_t
668         pattern=3);
669     bool eATCIPSTAIP(String ip,String
670         gateway,String netmask,uint8_t pattern=3);
671     bool qATCIPAP(String &ip,uint8_t
672         pattern=3);
673     bool eATCIPAP(String ip,uint8_t
674         pattern=3);
675     bool eCWSTARTSMART(uint8_t type);
676     bool eCWSTOPSMART(void);
677
678
679     bool eATCIPSTATUS(String &list);
680     bool sATCIPSTARTSingle(String type,
681         String addr, uint32_t port);
682     bool sATCIPSTARTMultiple(uint8_t mux_id,
683         String type, String addr, uint32_t port);
684     bool sATCIPSENDSingle(const uint8_t
685         *buffer, uint32_t len);
```

```

674     bool sATCIPSENDMultiple(uint8_t mux_id,
675         const uint8_t *buffer, uint32_t len);
676     bool sATCIPCLOSEMultiple(uint8_t
677         mux_id);
678     bool eATCIPCLOSESingle(void);
679     bool eATCIFSR(String &list);
680     bool sATCIPMUX(uint8_t mode);
681     bool sATCIPSERVER(uint8_t mode, uint32_t
682         port = 333);
683     bool sATCIPMODE(uint8_t mode);
684     bool eATSAVETRANSLINK(uint8_t
685         mode, String ip, uint32_t port);
686     bool eATPING(String ip);
687     bool sATCIPST0(uint32_t timeout);
688
689 /*
690  * +IPD,len:data
691  * +IPD,id,len:data
692  */
693
694 #ifdef ESP8266_USE_SOFTWARE_SERIAL
695     SoftwareSerial *m_puart; /* The UART to
696     communicate with ESP8266 */
697 #else
698     HardwareSerial *m_puart; /* The UART to
699     communicate with ESP8266 */
700 #endif
701 };
702
703#endif /* #ifndef __ESP8266_H__ */

```



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)[File List](#)[File Members](#)

## ESP8266.cpp

[Go to the documentation of this file.](#)

```
1
21 #include "ESP8266.h"
22
23 #define LOG_OUTPUT_DEBUG (1)
24 #define LOG_OUTPUT_DEBUG_PREFIX (1)
25
26 #define logDebug(arg) \
27     do { \
28         if (LOG_OUTPUT_DEBUG) \
29         { \
30             if (LOG_OUTPUT_DEBUG_PREFIX) \
31             { \
32                 Serial.print("[LOG Debug:"); \
33                 Serial.print((const \
34                     char*)__FILE__); \
35                 Serial.print(","); \
36                 Serial.print((unsigned \
37                     int)__LINE__); \
38                 Serial.print(","); \
39                 Serial.print((const \
40                     char*)__FUNCTION__); \
41                 Serial.print("] "); \
42             } \
43             Serial.print(arg); \
44         } \
45     } \
46 }
```

```
42     } while(0)
43
44 #ifdef ESP8266_USE_SOFTWARE_SERIAL
45 ESP8266::ESP8266(SoftwareSerial &uart,
46   uint32_t baud): m_puart(&uart)
47 {
48     m_puart->begin(baud);
49     rx_empty();
50 }
51 #else
52 ESP8266::ESP8266(HardwareSerial &uart,
53   uint32_t baud): m_puart(&uart)
54 {
55     m_puart->begin(baud);
56     rx_empty();
57 }
58 #endif
59
60 bool ESP8266::kick(void)
61 {
62     return eAT();
63 }
64
65 bool ESP8266::restart(void)
66 {
67     unsigned long start;
68     if (eATRST()) {
69         delay(2000);
70         start = millis();
71         while (millis() - start < 3000) {
72             if (eAT()) {
73                 delay(1500); /* Waiting for
74 stable */
75             }
76         }
77         return true;
78     }
79     delay(100);
80 }
```

```
76    }
77    return false;
78 }
79
80 String ESP8266::getVersion(void)
81 {
82     String version;
83     eATGMR(version);
84     return version;
85 }
86
87 bool ESP8266::setEcho(uint8_t mode)
88 {
89     return eATE(mode);
90 }
91
92 bool ESP8266::restore(void)
93 {
94     return eATRESTORE();
95 }
96 bool ESP8266::setUart(uint32_t
baudrate,uint8_t pattern)
97 {
98     return eATSETUART(baudrate,pattern);
99 }
100
101 bool ESP8266::deepSleep(uint32_t time)
102 {
103     return eATGSLP(time);
104 }
105
106
107 bool ESP8266::setOprToStation(uint8_t
pattern1,uint8_t pattern2)
108 {
109     uint8_t mode;
110     if (!qATCWMODE(&mode,pattern1)) {
```

```
111         return false;
112     }
113     if (mode == 1) {
114         return true;
115     } else {
116         if (SATCWMODE(1,pattern2)){
117             return true;
118         } else {
119             return false;
120         }
121     }
122 }
123 String ESP8266::getWifiModeList(void)
124 {
125     String list;
126     eATCWMODE(list);
127     return list;
128 }
129 bool ESP8266::setOprToSoftAP(uint8_t
pattern1,uint8_t pattern2)
130 {
131     uint8_t mode;
132     if (!qATCWMODE(&mode,pattern1)) {
133         return false;
134     }
135     if (mode == 2) {
136         return true;
137     } else {
138         if (SATCWMODE(2,pattern2) ){
139             return true;
140         } else {
141             return false;
142         }
143     }
144 }
145
146 bool ESP8266::setOprToStationSoftAP(uint8_t
```

```
    pattern1,uint8_t pattern2)
147 {           uint8_t mode;
148     if (!qATCWMODE(&mode,pattern1)) {
149         return false;
150     }
151     if (mode == 3) {
152         return true;
153     } else {
154         if (SATCWMODE(3,pattern2) ){
155             return true;
156         } else {
157             return false;
158         }
159     }
160 }
161 }
162
163 String ESP8266::getNowConecAp(uint8_t
pattern)
164 {
165     String ssid;
166     qATCWJAP(ssid,pattern);
167     return ssid;
168 }
169
170
171 String ESP8266::getAPList(void)
172 {
173     String list;
174     eATCW LAP(list);
175     return list;
176 }
177
178 bool ESP8266::joinAP(String ssid, String
pwd,uint8_t pattern)
179 {
180     return sATCWJAP(ssid, pwd,pattern);
```

```
181 }
182
183 bool ESP8266::leaveAP(void)
184 {
185     return eATCWQAP();
186 }
187
188 String ESP8266::getSoftAPPParam(uint8_t
189 pattern)
190 {
191     String list;
192     qATCWSAP(list,pattern);
193     return list;
194 }
195
196
197 bool ESP8266::setSoftAPPParam(String ssid,
198     String pwd, uint8_t chl, uint8_t ecn,uint8_t
199 pattern)
200 {
201     return sATCWSAP(ssid, pwd, chl,
202         ecn,pattern);
203 }
204
205 String ESP8266::getJoinedDeviceIP(void)
206 {
207     String list;
208     eATCWLIF(list);
209     return list;
210 }
211
212 String ESP8266::getDHCP(uint8_t pattern)
213 {
214     String dhcp;
215     qATCWDHCP(dhcp,pattern);
216     return dhcp;
```

```
214 }
215 bool ESP8266::setDHCP(uint8_t mode, uint8_t
en, uint8_t pattern)
216 {
217     return sATCWDHCP(mode, en, pattern);
218 }
219
220 bool ESP8266::setAutoConnect(uint8_t en)
221 {
222     return eATCWAUTOCONN(en);
223 }
224 String ESP8266::getStationMac(uint8_t
pattern)
225 {
226     String mac;
227     qATCIPSTAMAC(mac, pattern);
228     return mac;
229 }
230
231
232 bool ESP8266::setStationMac(String
mac, uint8_t pattern)
233 {
234     return eATCIPSTAMAC(mac, pattern);
235 }
236
237 String ESP8266::getStationIp(uint8_t
pattern)
238 {
239     String ip;
240     qATCIPSTAIP(ip, pattern);
241     return ip;
242 }
243
244 bool ESP8266::setStationIp(String ip, String
gateway, String netmask, uint8_t pattern)
245 {
```

```
246     return
247     eATCIPSTAIP(ip,gateway,netmask,pattern);
248
249 String ESP8266::getAPIp(uint8_t pattern)
250 {
251     String ip;
252     qATCIPAP(ip,pattern);
253     return ip;
254 }
255
256 bool ESP8266::setAPIp(String ip,uint8_t
pattern)
257 {
258     return eATCIPAP(ip,pattern);
259 }
260
261 bool ESP8266::startSmartConfig(uint8_t type)
262 {
263     return eCWSTARTSMART(type);
264 }
265
266 bool ESP8266::stopSmartConfig(void)
267 {
268     return eCWSTOPSMART();
269 }
270
271
272
273
274 String ESP8266::getIPStatus(void)
275 {
276     String list;
277     eATCIPSTATUS(list);
278     return list;
279 }
280
```

```
281 String ESP8266::getLocalIP(void)
282 {
283     String list;
284     eATCIFSR(list);
285     return list;
286 }
287
288 bool ESP8266::enableMUX(void)
289 {
290     return sATCIPMUX(1);
291 }
292
293 bool ESP8266::disableMUX(void)
294 {
295     return sATCIPMUX(0);
296 }
297
298 bool ESP8266::createTCP(String addr,
299   uint32_t port)
300 {
301     return sATCIPSTARTSingle("TCP", addr,
302       port);
303 }
304
305 bool ESP8266::releaseTCP(void)
306 {
307     return eATCIPCLOSESingle();
308 }
309
310 bool ESP8266::registerUDP(String addr,
311   uint32_t port)
312 {
313     return sATCIPSTARTSingle("UDP", addr,
314       port);
315 }
316
317 bool ESP8266::unregisterUDP(void)
```

```
314 {
315     return eATCIPCLOSESingle();
316 }
317
318 bool ESP8266::createTCP(uint8_t mux_id,
319 String addr, uint32_t port)
320 {
321     return sATCIPSTARTMultiple(mux_id,
322 "TCP", addr, port);
323 }
324
325 bool ESP8266::releaseTCP(uint8_t mux_id)
326 {
327     return sATCIPCLOSEMulitple(mux_id);
328 }
329
330 bool ESP8266::registerUDP(uint8_t mux_id,
331 String addr, uint32_t port)
332 {
333     return sATCIPSTARTMultiple(mux_id,
334 "UDP", addr, port);
335 }
336
337
338 bool ESP8266::unregisterUDP(uint8_t mux_id)
339 {
340     return sATCIPCLOSEMulitple(mux_id);
341 }
342
343 bool ESP8266::setTCPServerTimeout(uint32_t
344 timeout)
345 {
346     return sATCIPSTO(timeout);
347 }
348
349 bool ESP8266::startTCPServer(uint32_t port)
350 {
351     if (sATCIPSERVER(1, port)) {
```

```
346         return true;
347     }
348     return false;
349 }
350
351 bool ESP8266::stopTCPServer(void)
352 {
353     sATCIPSERVER(0);
354     restart();
355     return false;
356 }
357
358 bool ESP8266::setCIPMODE(uint8_t mode)
359 {
360     return sATCIPMODE(mode);
361 }
362
363 bool ESP8266::saveTransLink (uint8_t
mode, String ip,uint32_t port)
364 {
365     return eATSAVETRANSLINK(mode,ip,port);
366 }
367
368 bool ESP8266::setPing(String ip)
369 {
370     return eATPING(ip);
371 }
372
373
374
375
376 bool ESP8266::startServer(uint32_t port)
377 {
378     return startTCPServer(port);
379 }
380
381 bool ESP8266::stopServer(void)
```

```
382 {
383     return stopTCPServer();
384 }
385
386 bool ESP8266::send(const uint8_t *buffer,
387     uint32_t len)
388 {
389     return sATCIPSENDSingle(buffer, len);
390 }
391 bool ESP8266::send(uint8_t mux_id, const
392     uint8_t *buffer, uint32_t len)
393 {
394     return sATCIPSENDMultiple(mux_id,
395         buffer, len);
396 }
397 uint32_t ESP8266::recv(uint8_t *buffer,
398     uint32_t buffer_size, uint32_t timeout)
399 {
400     return recvPkg(buffer, buffer_size,
401         NULL, timeout, NULL);
402 }
403 uint32_t ESP8266::recv(uint8_t mux_id,
404     uint8_t *buffer, uint32_t buffer_size,
405     uint32_t timeout)
406 {
407     uint8_t id;
408     uint32_t ret;
409     ret = recvPkg(buffer, buffer_size, NULL,
410         timeout, &id);
411     if (ret > 0 && id == mux_id) {
412         return ret;
413     }
414     return 0;
415 }
```

```
411
412 uint32_t ESP8266::recv(uint8_t
413     *coming_mux_id, uint8_t *buffer, uint32_t
414     buffer_size, uint32_t timeout)
415 {
416     return recvPkg(buffer, buffer_size,
417         NULL, timeout, coming_mux_id);
418 }
419 /* +IPD,<id>,<len>:<data> */
420 /* +IPD,<len>:<data> */
421 uint32_t ESP8266::recvPkg(uint8_t *buffer,
422     uint32_t buffer_size, uint32_t *data_len,
423     uint32_t timeout, uint8_t *coming_mux_id)
424 {
425     String data;
426     char a;
427     int32_t index_PIPDcomma = -1;
428     int32_t index_colon = -1; /* : */
429     int32_t index_comma = -1; /* , */
430     int32_t len = -1;
431     int8_t id = -1;
432     bool has_data = false;
433     uint32_t ret;
434     unsigned long start;
435     uint32_t i;
436
437     if (buffer == NULL) {
438         return 0;
439     }
440
441     start = millis();
442     while (millis() - start < timeout) {
443         if(m_puart->available() > 0) {
```

```
442         a = m_puart->read();
443         data += a;
444     }
445
446     index_PIPDcomma =
447     data.indexOf("+IPD,");
448     if (index_PIPDcomma != -1) {
449         index_colon = data.indexOf(':', index_PIPDcomma + 5);
450         if (index_colon != -1) {
451             index_comma =
452             data.indexOf(',', index_PIPDcomma + 5);
453             /* +IPD,id,len:data */
454             if (index_comma != -1 &&
455                 index_comma < index_colon) {
456                 id =
457                 data.substring(index_PIPDcomma + 5,
458                 index_comma).toInt();
459                 if (id < 0 || id > 4) {
460                     return 0;
461                 }
462                 len =
463                 data.substring(index_comma + 1,
464                 index_colon).toInt();
465                 if (len <= 0) {
466                     return 0;
467                 }
468                 has_data = true;
break;
```

```
469         }
470     }
471 }
472
473 if (has_data) {
474     i = 0;
475     ret = len > buffer_size ?
476         buffer_size : len;
477     start = millis();
478     while (millis() - start < 3000) {
479         while(m_puart->available() > 0
480             && i < ret) {
481             a = m_puart->read();
482             buffer[i++] = a;
483         }
484         if (i == ret) {
485             rx_empty();
486             if (data_len) {
487                 *data_len = len;
488             }
489             if (index_comma != -1 &&
490                 coming_mux_id) {
491                 *coming_mux_id = id;
492             }
493         }
494     }
495 }
496
497 void ESP8266::rx_empty(void)
498 {
499     while(m_puart->available() > 0) {
500         m_puart->read();
501     }
502 }
```

```
503
504 String ESP8266::recvString(String target,
505   uint32_t timeout)
506 {
507     String data;
508     char a;
509     unsigned long start = millis();
510     while (millis() - start < timeout) {
511         while(m_puart->available() > 0) {
512             a = m_puart->read();
513             if(a == '\0') continue;
514             data += a;
515         }
516         if (data.indexOf(target) != -1) {
517             break;
518         }
519     }
520     return data;
521 }
522
523 String ESP8266::recvString(String target1,
524   String target2, uint32_t timeout)
525 {
526     String data;
527     char a;
528     unsigned long start = millis();
529     while (millis() - start < timeout) {
530         while(m_puart->available() > 0) {
531             a = m_puart->read();
532             if(a == '\0') continue;
533             data += a;
534         }
535         if (data.indexOf(target1) != -1) {
536             break;
537         } else if (data.indexOf(target2) !=
538 -1) {
```

```
537         break;
538     }
539 }
540 return data;
541 }
542
543 String ESP8266::recvString(String target1,
544     String target2, String target3, uint32_t
545     timeout)
546 {
547     String data;
548     char a;
549     unsigned long start = millis();
550     while (millis() - start < timeout) {
551         while(m_puart->available() > 0) {
552             a = m_puart->read();
553             if(a == '\0') continue;
554             data += a;
555         }
556         if (data.indexOf(target1) != -1) {
557             break;
558         } else if (data.indexOf(target2) !=
559         -1) {
560             break;
561         }
562     }
563 }
564
565 bool ESP8266::recvFind(String target,
566     uint32_t timeout)
567 {
568     String data_tmp;
569     data_tmp = recvString(target, timeout);
```

```
569     if (data_tmp.indexOf(target) != -1) {
570         return true;
571     }
572     return false;
573 }
574
575 bool ESP8266::recvFindAndFilter(String
576     target, String begin, String end, String
577     &data, uint32_t timeout)
578 {
579     String data_tmp;
580     data_tmp = recvString(target, timeout);
581     if (data_tmp.indexOf(target) != -1) {
582         int32_t index1 =
583             data_tmp.indexOf(begin);
584         int32_t index2 =
585             data_tmp.indexOf(end);
586         if (index1 != -1 && index2 != -1) {
587             index1 += begin.length();
588             data =
589             data_tmp.substring(index1, index2);
590             return true;
591         }
592     }
593     data = "";
594     return false;
595 }
596
597 bool ESP8266::eAT(void)
598 {
599     rx_empty();
600     m_puart->println("AT");
601     return recvFind("OK");
602 }
603
604 bool ESP8266::eATRST(void)
605 {
606     rx_empty();
607     m_puart->println("ATRST");
608     return recvFind("OK");
609 }
```

```
601     rx_empty();
602     m_puart->println("AT+RST");
603     return recvFind("OK");
604 }
605
606 bool ESP8266::eATGMR(String &version)
607 {
608     rx_empty();
609     delay(3000);
610     m_puart->println("AT+GMR");
611     return recvFindAndFilter("OK", "\r\r\n",
612                             "\r\n\r\nOK", version, 10000);
613 }
614
615 bool ESP8266::eATGSLP(uint32_t time)
616 {
617     rx_empty();
618     m_puart->print("AT+GSLP=");
619     m_puart->println(time);
620     return recvFind("OK");
621
622
623 bool ESP8266::eATE(uint8_t mode)
624 {
625     rx_empty();
626     m_puart->print("ATE");
627     m_puart->println(mode);
628     return recvFind("OK");
629 }
630
631 bool ESP8266::eATRESTORE(void)
632 {
633     rx_empty();
634     m_puart->println("AT+RESTORE");
635     return recvFind("OK");
636 }
```

```
637
638
639 bool ESP8266::eATSETUART(uint32_t
640   baudrate,uint8_t pattern)
641 {
642   rx_empty();
643   if(pattern>3||pattern<1){
644     return false;
645   }
646   switch(pattern){
647     case 1:
648       m_puart->print("AT+UART=");
649       break;
650     case 2:
651       m_puart->print("AT+UART_CUR=");
652       break;
653     case 3:
654       m_puart->print("AT+UART_DEF=");
655       break;
656   m_puart->print(baudrate);
657   m_puart->print(",");
658   m_puart->print(8);
659   m_puart->print(",");
660   m_puart->print(1);
661   m_puart->print(",");
662   m_puart->print(0);
663   m_puart->print(",");
664   m_puart->println(0);
665   if(recvFind("OK",5000)){
666
667     m_puart->begin(baudrate);
668     return true;
669   }
670   else{
671     return false;
672   }
```

```
673 }
674 }
675
676
677 bool ESP8266::qATCWMODE(uint8_t
678 *mode,uint8_t pattern)
679 {
680     String str_mode;
681     bool ret;
682     if (!mode||!pattern) {
683         return false;
684     }
685     rx_empty();
686     switch(pattern)
687     {
688         case 1 :
689             m_puart-
690             >println("AT+CWMODE_DEF?");
691             break;
692         case 2:
693             m_puart-
694             >println("AT+CWMODE_CUR?");
695             break;
696         default:
697             m_puart->println("AT+CWMODE?");
698     }
699     ret = recvFindAndFilter("OK", ":" ,
700     "\r\n\r\nOK", str_mode);
701     if (ret) {
702         *mode = (uint8_t)str_mode.toInt();
703         return true;
704     } else {
705         return false;
706     }
707 }
708 bool ESP8266::eATCWMODE(String &list)
709 {
```

```
706     rx_empty();
707     m_puart->println("AT+CWMODE=?");
708     return recvFindAndFilter("OK", "+CWMODE:",
709     ("", "")\r\n\r\nOK", list);
710 }
711 bool ESP8266::sATCWMODE(uint8_t mode, uint8_t
712 pattern)
713 {
714     if(!pattern){
715         return false;
716     }
717     String data;
718     rx_empty();
719     switch(pattern)
720     {
721         case 1 :
722             m_puart-
723             >print("AT+CWMODE_DEF=");
724             break;
725         case 2:
726             m_puart-
727             >print("AT+CWMODE_CUR=");
728             break;
729         default:
730             m_puart->print("AT+CWMODE=");
731     }
732     m_puart->println(mode);
733     data = recvString("OK", "no change");
734
735     if (data.indexOf("OK") != -1 || 
736     data.indexOf("no change") != -1) {
737         return true;
738     }
739     return false;
740 }
```

```
738
739 bool ESP8266::qATCWJAP(String &ssid,uint8_t
    pattern)
740 {
741
742     bool ret;
743     if (!pattern) {
744         return false;
745     }
746     rx_empty();
747     switch(pattern)
748     {
749         case 1 :
750             m_puart-
751             >println("AT+CWJAP_DEF?");
752             break;
753         case 2:
754             m_puart-
755             >println("AT+CWJAP_CUR?");
756             break;
757         default:
758             m_puart->println("AT+CWJAP?");
759             ssid = recvString("OK", "No AP");
760             if (ssid.indexOf("OK") != -1 || 
761                 ssid.indexOf("No AP") != -1) {
762                 return true;
763             }
764     }
765
766     bool ESP8266::sATCWJAP(String ssid, String
        pwd,uint8_t pattern)
767 {
768     String data;
769     if (!pattern) {
```

```
770         return false;
771     }
772     rx_empty();
773     switch(pattern)
774     {
775         case 1 :
776             m_puart-
777                 >print("AT+CWJAP_DEF=\\" );
778                 break;
779         case 2:
780             m_puart-
781                 >print("AT+CWJAP_CUR=\\" );
782                 break;
783         default:
784             m_puart->print("AT+CWJAP=\\" );
785         }
786         m_puart->print(ssid);
787         m_puart->print("\\", "\\");
788         m_puart->print(pwd);
789         m_puart->println("\\\"");
790
791         data = recvString("OK", "FAIL", 10000);
792         if (data.indexOf("OK") != -1) {
793             return true;
794         }
795         return false;
796     }
797
798     bool ESP8266::eATCWLAP(String &list)
799     {
800         String data;
801         rx_empty();
802         m_puart->println("AT+CWLAP");
803         return recvFindAndFilter("OK", "\r\r\n",
804             "\r\n\r\nOK", list, 15000);
```

```
804 }
805
806
807
808
809 bool ESP8266::eATCWQAP(void)
810 {
811     String data;
812     rx_empty();
813     m_puart->println("AT+CWQAP");
814     return recvFind("OK");
815 }
816
817
818 bool ESP8266::qATCWSAP(String &List, uint8_t
pattern)
819 {
820     if (!pattern) {
821         return false;
822     }
823     rx_empty();
824     switch(pattern)
825     {
826         case 1 :
827             m_puart-
828             >println("AT+CWSAP_DEF?");
829             break;
830         case 2:
831             m_puart-
832             >println("AT+CWSAP_CUR?");
833             break;
834         default:
835             m_puart->println("AT+CWSAP?");
836     }
837     return recvFindAndFilter("OK", "\r\r\n",
"\r\n\r\nOK", List, 10000);
```

```
837
838 }
839
840 bool ESP8266::sATCWSAP(String ssid, String
841   pwd, uint8_t chl, uint8_t ecn,uint8_t pattern)
842 {
843   String data;
844   if (!pattern) {
845     return false;
846   }
847   rx_empty();
848   switch(pattern){
849     case 1 :
850       m_puart-
851         >print("AT+CWSAP_DEF=\\" );
852         break;
853     case 2:
854       m_puart-
855         >print("AT+CWSAP_CUR=\\" );
856         break;
857     default:
858       m_puart->print("AT+CWSAP=\\" );
859
860       m_puart->print(ssid);
861       m_puart->print("\",\"");
862       m_puart->print(pwd);
863       m_puart->print("\",");
864       m_puart->print(chl);
865       m_puart->print(",");
866       m_puart->println(ecn);
867
868       data = recvString("OK", "ERROR", 5000);
869       if (data.indexOf("OK") != -1) {
870         return true;
871       }
872     }
```

```
871         return false;
872     }
873
874     bool ESP8266::eATCWLIF(String &list)
875     {
876         String data;
877         rx_empty();
878         m_puart->println("AT+CWLIF");
879         return recvFindAndFilter("OK", "\r\r\n",
880 "OK", list);
881     }
882
883     bool ESP8266::qATCWDHCP(String &List, uint8_t
884 pattern)
885     {
886         if (!pattern) {
887             return false;
888         }
889         rx_empty();
890         switch(pattern)
891         {
892             case 1 :
893                 m_puart-
894             >println("AT+CWDHCP_DEF?");
895                 break;
896             case 2:
897                 m_puart-
898             >println("AT+CWDHCP_CUR?");
899                 break;
900             default:
901                 m_puart->println("AT+CWDHCP?");
902         }
903
904         return recvFindAndFilter("OK", "\r\r\n",
905 "OK", List, 10000);
906     }
907 }
```

```
903
904
905 bool ESP8266::sATCWDHCP(uint8_t mode,
906     uint8_t en, uint8_t pattern)
907 {
908     String data;
909     if (!pattern) {
910         return false;
911     }
912     rx_empty();
913     switch(pattern){
914         case 1 :
915             m_puart-
916                 >print("AT+CWDHCP_DEF=");
917                 break;
918         case 2:
919             m_puart-
920                 >print("AT+CWDHCP_CUR=");
921                 break;
922         default:
923             m_puart->print("AT+CWDHCP=");
924             m_puart->print(mode);
925             m_puart->print(",");
926             m_puart->println(en);
927             data = recvString("OK", "ERROR", 2000);
928
929             if (data.indexOf("OK") != -1) {
930                 return true;
931             }
932             return false;
933     }
934
935
936 bool ESP8266::eATCWCONN(uint8_t en)
```

```
937    {
938
939        rx_empty();
940        if(en>1||en<0){
941            return false;
942        }
943        m_puart->print("AT+CWAUTOCONN=");
944        m_puart->println(en);
945        return recvFind("OK");
946
947    }
948
949    bool ESP8266::qATCIPSTAMAC(String
950        &mac,uint8_t pattern)
951    {
952
953        rx_empty();
954        if (!pattern) {
955            return false;
956        }
957        switch(pattern){
958            case 1 :
959                m_puart-
960                >println("AT+CIPSTAMAC_DEF?");
961                break;
962            case 2:
963                m_puart-
964                >println("AT+CIPSTAMAC_CUR?");
965                break;
966            default:
967                m_puart-
968                >println("AT+CIPSTAMAC?");
969
970        }
971        return recvFindAndFilter("OK", "\r\r\n",
972            "\r\n\r\nOK", mac,2000);
```

```
969
970 }
971
972
973
974 bool ESP8266::eATCIPSTAMAC(String
  mac,uint8_t pattern)
975 {
976
977     rx_empty();
978     if (!pattern) {
979         return false;
980     }
981     switch(pattern){
982         case 1 :
983             m_puart-
984             >print("AT+CIPSTAMAC_DEF=");
985             break;
986         case 2:
987             m_puart-
988             >print("AT+CIPSTAMAC_CUR=");
989             break;
990         default:
991             m_puart->print("AT+CIPSTAMAC=");
992     }
993     m_puart->print("\\" );
994     m_puart->print(mac);
995     m_puart->println("\\\"");
996     return recvFind("OK");
997
998 }
999
1000 bool ESP8266::qATCIPSTAIP(String &ip,uint8_t
  pattern)
1001 {
```

```
1002     rx_empty();
1003     if (!pattern) {
1004         return false;
1005     }
1006     switch(pattern){
1007         case 1 :
1008             m_puart-
1009             >println("AT+CIPSTA_DEF?");
1010             break;
1011         case 2:
1012             m_puart-
1013             >println("AT+CIPSTA_CUR?");
1014             break;
1015         default:
1016             m_puart->println("AT+CIPSTA?");
1017         }
1018     }
1019     return recvFindAndFilter("OK", "\r\r\n",
1020     "\r\n\r\nOK", ip, 2000);
1021 }
1022
1023 bool ESP8266::eATCIPSTAIP(String ip, String
1024 gateway, String netmask, uint8_t pattern)
1025 {
1026     rx_empty();
1027     if (!pattern) {
1028         return false;
1029     }
1030     switch(pattern){
1031         case 1 :
1032             m_puart-
1033             >print("AT+CIPSTA_DEF=");
```

```
1034         break;
1035     case 2:
1036         m_puart-
1037             >print("AT+CIPSTA_CUR=");
1038             break;
1039     default:
1040         m_puart->print("AT+CIPSTA=");
1041     }
1042     m_puart->print("\\");
1043     m_puart->print(ip);
1044     m_puart->print("\\," "\\");
1045     m_puart->print(gateway);
1046     m_puart->print("\\," "\\");
1047     m_puart->print(netmask);
1048     m_puart->println("\\\"");
1049     return recvFind("OK");
1050
1051 }
1052
1053
1054 bool ESP8266::qATCIPAP(String &ip,uint8_t
pattern)
1055 {
1056
1057     rx_empty();
1058     if (!pattern) {
1059         return false;
1060     }
1061     switch(pattern){
1062         case 1 :
1063             m_puart-
1064                 >println("AT+CIPAP_DEF?");
1065                 break;
1066         case 2:
1067             m_puart-
```

```
        >println("AT+CIPAP_CUR?");  
1068         break;  
1069     default:  
1070         m_puart->println("AT+CIPAP?");  
1071     }  
1072     return recvFindAndFilter("OK", "\r\r\n",  
1073                               "\r\n\r\nOK", ip, 2000);  
1074  
1075 }  
1076  
1077  
1078 bool ESP8266::eATCIPAP(String ip,uint8_t  
    pattern)  
1079 {  
1080     rx_empty();  
1081     if (!pattern) {  
1082         return false;  
1083     }  
1084     switch(pattern){  
1085         case 1 :  
1086             m_puart->print("AT+CIPAP_DEF=");  
1087             break;  
1088         case 2:  
1089             m_puart->print("AT+CIPAP_CUR=");  
1090             break;  
1091         default:  
1092             m_puart->print("AT+CIPAP=");  
1093         }  
1094         m_puart->print("\\"");  
1095         m_puart->print(ip);  
1096         m_puart->println("\\"");  
1097         return recvFind("OK");  
1098  
1099  
1100  
1101
```

```
1102 }
1103
1104
1105 bool ESP8266::eCWSTARTSMART(uint8_t type)
1106 {
1107     rx_empty();
1108     m_puart->print("AT+CWSTARTSMART=");
1109     m_puart->println(type);
1110     return recvFind("OK");
1111 }
1112
1113 bool ESP8266::eCWSTOPSMART(void)
1114 {
1115     rx_empty();
1116     m_puart->println("AT+CWSTOPSMART");
1117     return recvFind("OK");
1118 }
1119
1120 bool ESP8266::eATCIPSTATUS(String &list)
1121 {
1122     String data;
1123     delay(100);
1124     rx_empty();
1125     m_puart->println("AT+CIPSTATUS");
1126     return recvFindAndFilter("OK", "\r\r\n",
1127                             "\r\n\r\nOK", list);
1128 }
1129 bool ESP8266::sATCIPSTARTSingle(String type,
1130                                 String addr, uint32_t port)
1131 {
1132     String data;
1133     rx_empty();
1134     m_puart->print("AT+CIPSTART=\\" );
1135     m_puart->print(type);
1136     m_puart->print("\\,\\\" );
1137     m_puart->print(addr);
1138     m_puart->print("\\,\\");
```

```
1137     m_puart->println(port);
1138
1139     data = recvString("OK", "ERROR",
1140                         "ALREADY CONNECT", 10000);
1140     if (data.indexOf("OK") != -1 || data.indexOf("ALREADY CONNECT") != -1) {
1141         return true;
1142     }
1143     return false;
1144 }
1145 bool ESP8266::sATCIPSTARTMultiple(uint8_t
1146 mux_id, String type, String addr, uint32_t
1147 port)
1148 {
1149     String data;
1150     rx_empty();
1151     m_puart->print("AT+CIPSTART=");
1152     m_puart->print(mux_id);
1153     m_puart->print(",\"");
1154     m_puart->print(type);
1155     m_puart->print("\\",\"");
1156     m_puart->print(addr);
1157     m_puart->print("\\",");
1158     m_puart->println(port);

1159     data = recvString("OK", "ERROR",
1160                         "ALREADY CONNECT", 10000);
1160     if (data.indexOf("OK") != -1 || data.indexOf("ALREADY CONNECT") != -1) {
1161         return true;
1162     }
1163     return false;
1164 }
1165 bool ESP8266::sATCIPSENDSingle(const uint8_t
1166 *buffer, uint32_t len)
1167 {
1168     rx_empty();
```

```
1167     m_puart->print("AT+CIPSEND=");
1168     m_puart->println(len);
1169     if (recvFind(">", 5000)) {
1170         rx_empty();
1171         for (uint32_t i = 0; i < len; i++) {
1172             m_puart->write(buffer[i]);
1173         }
1174         return recvFind("SEND OK", 10000);
1175     }
1176     return false;
1177 }
1178 bool ESP8266::sATCIPSENDMultiple(uint8_t
1179     mux_id, const uint8_t *buffer, uint32_t len)
1180 {
1181     rx_empty();
1182     m_puart->print("AT+CIPSEND=");
1183     m_puart->print(mux_id);
1184     m_puart->print(",");
1185     m_puart->println(len);
1186     if (recvFind(">", 5000)) {
1187         rx_empty();
1188         for (uint32_t i = 0; i < len; i++) {
1189             m_puart->write(buffer[i]);
1190         }
1191         return recvFind("SEND OK", 10000);
1192     }
1193     return false;
1194 }
1195 bool ESP8266::sATCIPCLOSEMulitple(uint8_t
1196     mux_id)
1197 {
1198     String data;
1199     rx_empty();
1200     m_puart->print("AT+CIPCLOSE=");
1201     m_puart->println(mux_id);
1202
1203     data = recvString("OK", "link is not",
1204
```

```
    5000);
1202    if (data.indexOf("OK") != -1 || data.indexOf("link is not") != -1) {
1203        return true;
1204    }
1205    return false;
1206}
1207bool ESP8266::eATCIPCLOSESingle(void)
1208{
1209    rx_empty();
1210    m_puart->println("AT+CIPCLOSE");
1211    return recvFind("OK", 5000);
1212}
1213bool ESP8266::eATCIFSR(String &list)
1214{
1215    rx_empty();
1216    m_puart->println("AT+CIFSR");
1217    return recvFindAndFilter("OK", "\r\r\n",
1218                             "\r\n\r\nOK", list);
1219}
1220bool ESP8266::sATCIPMUX(uint8_t mode)
1221{
1222    String data;
1223    rx_empty();
1224    m_puart->print("AT+CIPMUX=");
1225    m_puart->println(mode);
1226
1227    data = recvString("OK", "Link is
1228                      builded");
1229    if (data.indexOf("OK") != -1) {
1230        return true;
1231    }
1232    return false;
1233}
```

```
1234     String data;
1235     if (mode) {
1236         rx_empty();
1237         m_puart->print("AT+CIPSERVER=1, ");
1238         m_puart->println(port);
1239
1240         data = recvString("OK", "no
change");
1241         if (data.indexOf("OK") != -1 || 
data.indexOf("no change") != -1) {
1242             return true;
1243         }
1244         return false;
1245     } else {
1246         rx_empty();
1247         m_puart->println("AT+CIPSERVER=0");
1248         return recvFind("\r\r\n");
1249     }
1250 }
1251
1252
1253 bool ESP8266::sATCIPMODE(uint8_t mode)
1254 {
1255     String data;
1256     if(mode>1||mode<0){
1257         return false;
1258     }
1259     rx_empty();
1260     m_puart->print("AT+CIPMODE=");
1261     m_puart->println(mode);
1262
1263     data = recvString("OK", "Link is
builted",2000);
1264     if (data.indexOf("OK") != -1 ) {
1265         return true;
1266     }
1267     return false;
```

```
1268 }
1269
1270
1271
1272
1273 bool ESP8266::eATSAVETRANSLINK(uint8_t
    mode, String ip, uint32_t port)
1274 {
1275
1276     String data;
1277     rx_empty();
1278     m_puart->print("AT+SAVETRANSLINK=");
1279     m_puart->print(mode);
1280     m_puart->print(",\"");
1281     m_puart->print(ip);
1282     m_puart->print("\",");
1283     m_puart->println(port);
1284     data = recvString("OK", "ERROR", 2000);
1285     if (data.indexOf("OK") != -1) {
1286         return true;
1287     }
1288     return false;
1289 }
1290
1291
1292
1293 bool ESP8266::eATPING(String ip)
1294 {
1295     rx_empty();
1296     m_puart->print("AT+PING=");
1297     m_puart->print("\\"");
1298     m_puart->print(ip);
1299     m_puart->println("\\"");
1300     return recvFind("OK", 2000);
1301 }
1302
1303
```

```
1304  
1305     bool ESP8266::sATCIPST0(uint32_t timeout)  
1306     {  
1307         rx_empty();  
1308         m_puart->print("AT+CIPST0=");  
1309         m_puart->println(timeout);  
1310         return recvFind("OK");  
1311     }  
1312 }
```



# API

## For Arduino developers

[Main Page](#)[Related Pages](#)[Classes](#)[Files](#)[Examples](#)[File List](#)[File Members](#)

## doxygen.h

[Go to the documentation of this file.](#)

```
1
16 #ifndef __IOTGO_DOXYGEN_H__
17 #define __IOTGO_DOXYGEN_H__
18
19 #endif /* #ifndef __IOTGO_DOXYGEN_H__ */
```