**Station Class**

The number of features provided by ESP8266 in the station mode is far more extensive than covered in original [Arduino WiFi library](https://www.arduino.cc/en/Reference/WiFi). Therefore, instead of supplementing original documentation, we have decided to write a new one from scratch.

Кількість функцій, підтримуваних ESP8266 в режимі станції, набагато більше, ніж в оригінальній бібліотеці [Arduino WiFi](https://www.arduino.cc/en/Reference/WiFi). Тому, замість того, щоб доповнювати оригінальну документацію, було вирішено написати нову з нуля.

Description of station class has been broken down into four parts. First discusses methods to establish connection to an access point. Second provides methods to manage connection like e.g. reconnect or isConnected. Third covers properties to obtain information about connection like MAC or IP address. Finally the fourth section provides alternate methods to connect like e.g. Wi-Fi Protected Setup (WPS).

Опис класу *station* був розбитий на чотири частини. Перший описує способи встановлення зв'язку з точкою доступу. Другий надає способи керування з'єднанням, наприклад такі як: reconnect, або isConnected. Третій охоплює налаштування виводу інформації про з'єднання, таку як MAC або IP-адресу. Нарешті, четвертий розділ надає альтернативні способи з'єднання, наприклад такі як Wi-Fi Protected Setup (WPS).

Points below provide description and code snippets how to use particular methods.

Наведені нижче пункти містять описи та фрагменти коду, про те як використовувати певні методи.

For more code samples please refer to separate section with [examples](http://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/station-examples.html) dedicated specifically to the Station Class.

Для отримання додаткових зразків коду, будь ласка, зверніться до розділу з прикладами, спеціально присвяченими Класу Станції.

## Start Here

Switching the module to Station mode is done with beginfunction. Typical parameters passed to begin include SSID and password, so module can connect to specific Access Point.

Переключення модуля в режим станції здійснюється за допомогою функції begin. Типові параметри, що передаються нею, включають SSID та пароль, так модуль може підключатися до певної точки доступу.

WiFi**.**begin(ssid, password)

By default, ESP will attempt to reconnect to Wi-Fi network whenever it is disconnected. There is no need to handle this by separate code. A good way to simulate disconnection would be to reset the access point. ESP will report disconnection, and then try to reconnect automatically.

За замовчуванням, ESP спробує підключитися до Wi-Fi мережі, коли він *offline*, і обробляти підключення окремо не має потреби. Хорошим способом імітації відключення буде скидання точки доступу. ESP повідомить про відключення, а потім спробуйте знов підключитися автоматично.

### begin

There are several versions (called `function overloads <https://en.wikipedia.org/wiki/Function\_overloading>`\_\_ in C++) of begin function. One was presented just above:WiFi.begin(ssid, password). Overloads provide flexibility in number or type of accepted parameters.

Існує кілька версій функції begin (в C++ це називається – [function overloads](https://uk.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%B2%D0%B0%D0%BD%D1%82%D0%B0%D0%B6%D0%B5%D0%BD%D0%BD%D1%8F_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D1%96%D1%97)). Одина з них була представлена вище: WiFi.begin (ssid, pass). Перевантаження забезпечує гнучкість функції у кількості та типі вхідних параметрів.

The simplest overload of begin is as follows:

Так, за допомогою перенавантаження, ми можемо викликати begin без параметрів:

WiFi**.**begin()

Calling it will instruct module to switch to the station mode and connect to the last used access point basing on configuration saved in flash memory.

Викликавши begin таким чином модуль переключитися в режим станції та підключитися до останньої використаної точки доступу, виходячи з конфігурації, збереженої у флеш-пам'яті.

Below is the syntax of another overload of begin with the all possible parameters:

Нижче наведено синтаксис іншого перевантаження з усіма можливими параметрами:

WiFi**.**begin(ssid, password, channel, bssid, connect)

Meaning of parameters is as follows: \* ssid - a character string containing the SSID of Access Point we would like to connect to, may have up to 32 characters \* password to the access point, a character string that should be minimum 8 characters long and not longer than 64 characters \* channel of AP, if we like to operate using specific channel, otherwise this parameter may be omitted \* bssid - mac address of AP, this parameter is also optional \* connect - a boolean parameter that if set to false, will instruct module just to save the other parameters without actually establishing connection to the access point

Зміст параметрів – наступний:

ssid - символьна рядок, що містить [назву](https://uk.wikipedia.org/wiki/SSID) точки доступу, до якої ми хочемо підключитися, може мати до 32 символів

password - це пароль до точки доступу, тип змінної – символьний рядок, що має бути не менше 8 символів і не більше 64 символів

channel - це канал [AP](https://uk.wikipedia.org/wiki/%D0%91%D0%B5%D0%B7%D0%B4%D1%80%D0%BE%D1%82%D0%BE%D0%B2%D0%B0_%D1%82%D0%BE%D1%87%D0%BA%D0%B0_%D0%B4%D0%BE%D1%81%D1%82%D1%83%D0%BF%D1%83) ([всього їх 14, в Україні дозволеними є 1-13](https://uk.wikipedia.org/wiki/%D0%A1%D0%BC%D1%83%D0%B3%D0%B8_%D1%80%D0%B0%D0%B4%D1%96%D0%BE%D1%87%D0%B0%D1%81%D1%82%D0%BE%D1%82_Wi-Fi)). Цей параметр потрібний якщо нам знадобиться працювати за допомогою певного каналу, інакше цей параметр може бути опущений

bssid - [mac address](https://uk.wikipedia.org/wiki/MAC-%D0%B0%D0%B4%D1%80%D0%B5%D1%81%D0%B0) AP, цей параметр також є необов'язковим

connect - логічний параметр, і при значенні – *false*, буде вказувати модулю зберегти надані параметри, фактично, не встановлюючи з'єднання з точкою доступу, також є необов’язковим

### config

Disable [DHCP](https://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol) client (Dynamic Host Configuration Protocol) and set the IP configuration of station interface to user defined arbitrary values. The interface will be a static IP configuration instead of values provided by DHCP.

Вимкніть [DHCP](https://uk.wikipedia.org/wiki/DHCP)-клієнт (Dynamic Host Configuration Protocol) та встановіть IP-конфігурацію інтерфейсу станції на визначені користувачем довільні значення. Будуть встановлені статичні IP-конфігурації замість значень, наданих DHCP.

WiFi**.**config(local\_ip, gateway, subnet, dns1, dns2)

Function will return true if configuration change is applied successfully. If configuration can not be applied, because e.g. module is not in station or station + soft access point mode, then false will be returned.

Функція повертає true , якщо зміна конфігурації була успішно застосована. Якщо конфігурація не може бути застосована, наприклад якщо модуль не у режимі *station* або *station + soft access point*, то повернеться false .

The following IP configuration may be provided:

Наступні конфігурації IP забезпечують:

* local\_ip - enter here IP address you would like to assign the ESP station’s interface
* gateway - should contain IP address of gateway (a router) to access external networks
* subnet - this is a mask that defines the range of IP addresses of the local network
* dns1, dns2 - optional parameters that define IP addresses of Domain Name Servers (DNS) that maintain a directory of domain names (like e.g. www.google.co.uk) and translate them for us to IP addresses
* local\_ip - введіть тут [IP-адресу](https://uk.wikipedia.org/wiki/IP-%D0%B0%D0%B4%D1%80%D0%B5%D1%81%D0%B0), яку ви хочете призначити станції ESP
* gateway - повинен містити IP-адресу [шлюзу](https://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D1%80%D0%B5%D0%B6%D0%B5%D0%B2%D0%B8%D0%B9_%D1%88%D0%BB%D1%8E%D0%B7) (маршрутизатора) для доступу до зовнішніх мереж
* subnet - це [маска](https://uk.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D0%BA%D0%B0_%D0%BF%D1%96%D0%B4%D0%BC%D0%B5%D1%80%D0%B5%D0%B6%D1%96), яка визначає діапазон IP-адрес локальної мережі
* dns1, dns2 - необов'язкові параметри, що визначають IP-адреси серверів доменних імен ([DNS](https://uk.wikipedia.org/wiki/%D0%94%D0%BE%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%96%D0%BC%D0%B5%D0%BD)), які підтримують каталог доменних імен (наприклад, www.google.co.uk) і перекладають їх для нас на IP-адреси

Приклад коду:

*#include <ESP8266WiFi.h>*

const char**\*** ssid **=** "\*\*\*\*\*\*\*\*";

const char**\*** password **=** "\*\*\*\*\*\*\*\*";

IPAddress staticIP(192,168,1,22);

IPAddress gateway(192,168,1,9);

IPAddress subnet(255,255,255,0);

void setup(void)

{

Serial**.**begin(115200);

Serial**.**println();

Serial**.**printf("Connecting to %s\n", ssid);

WiFi**.**begin(ssid, password);

WiFi**.**config(staticIP, gateway, subnet);

**while** (WiFi**.**status() **!=** WL\_CONNECTED)

{

delay(500);

Serial**.**print(".");

}

Serial**.**println();

Serial**.**print("Connected, IP address: ");

Serial**.**println(WiFi**.**localIP());

}

void loop() {}

Приклад виводу:

Connecting to sensor**-**net

**.**

Connected, IP address: 192.168**.**1.22

Please note that station with static IP configuration usually connects to the network faster. In the above example it took about 500ms (one dot . displayed). This is because obtaining of IP configuration by DHCP client takes time and in this case this step is skipped. If you pass all three parameter as 0.0.0.0 (local\_ip, gateway and subnet), it will re enable DHCP. You need to re-connect the device to get new IPs.

Зверніть увагу, що станція зі статичною IP-конфігурацією, як правило, швидше з'єднується з мережею. У наведеному вище прикладі було необхідно близько 500 мс (вивід знаку точки). Це тому, що отримання конфігурації IP через DHCP-клієнт потребує часу, і в цьому випадку цей крок пропускається. Якщо ви передаєте всі три параметри як 0.0.0.0 (local\_ip, шлюз та підмережі), він знову активує DHCP. Щоб отримати нові IP-адреси, потрібно знову перепід'єднати пристрій.

## Manage Connection

### reconnect

Reconnect the station. This is done by disconnecting from the access point an then initiating connection back to the same AP.

Перепід'єднання станції. Ця функція виконує від'єднання від точки доступу, а потім відновлює з'єднання назад до тієї ж самої точки AP.

WiFi**.**reconnect()

Notes: 1. Station should be already connected to an access point. If this is not the case, then function will return false not performing any action. 2. If true is returned it means that connection sequence has been successfully started. User should still check for connection status, waiting until WL\_CONNECTED is reported:

Примітки: По-перше, станція повинна бути вже з'єднана з точкою доступу, а якщо це не так, то функція поверне - false, не виконуючи жодних дій. По-друге, якщо відповідь - true, це означає, що відбулося успішне з'єднання. Користувач все ще повинен перевіряти стан підключення, очікуючи повідомлення WL\_CONNECTED:

WiFi**.**reconnect();

**while** (WiFi**.**status() **!=** WL\_CONNECTED)

{

delay(500);

Serial**.**print(".");

}

### disconnect

Sets currently configured SSID and password to null values and disconnects the station from an access point.

Встановлює поточні налаштування SSID та пароля в значення null  і від'єднує станцію від точки доступу.

WiFi**.**disconnect(wifioff)

The wifioff is an optional boolean parameter. If set to true, then the station mode will be turned off.

wifioff - необов'язковий логічний параметр. Якщо встановлено значення true, то режим станції буде вимкнений.

### isConnected

Returns true if Station is connected to an access point or false if not.

Повертає true, якщо станція підключена до точки доступу, або – false  в інакшому випадку.

WiFi**.**isConnected()

### setAutoConnect

Configure module to automatically connect on power on to the last used access point.

Налаштування модуля для автоматичного підключення до останньої використаної точки доступу при подачі живлення.

WiFi**.**setAutoConnect(autoConnect)

The autoConnect is an optional parameter. If set to false then auto connection functionality up will be disabled. If omitted or set to true, then auto connection will be enabled.

autoConnect  є необов'язковим параметром. Якщо встановлено значення false, автоматичне підключення буде вимкнено. Якщо опустити або встановити значення true, автоматичне з'єднання буде активовано.

### getAutoConnect

This is “companion” function to setAutoConnect(). It returns true if module is configured to automatically connect to last used access point on power on.

Це функція супутник до setAutoConnect() (зазвичай [get](https://docs.microsoft.com/uk-ua/dotnet/csharp/language-reference/keywords/get;%20https:/docs.microsoft.com/uk-ua/dotnet/csharp/language-reference/keywords/set) та [set](https://docs.microsoft.com/uk-ua/dotnet/csharp/language-reference/keywords/set) функції застосовують як спосіб доступу до змінних об’єкту, вони є [властивостями пограмування](https://uk.wikipedia.org/wiki/%D0%92%D0%BB%D0%B0%D1%81%D1%82%D0%B8%D0%B2%D1%96%D1%81%D1%82%D1%8C_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F))). Вона повертає true, якщо модуль налаштований на автоматичне підключення до останньої використовуваної точки доступу при ввімкненні живлення.

WiFi**.**getAutoConnect()

If auto connection functionality is disabled, then function returns false.

Якщо функція автоматичного підключення вимкнена, функція повертає значення false.

### setAutoReconnect

Set whether module will attempt to reconnect to an access point in case it is disconnected.

Вказує, чи буде модуль намагатися повторно підключитись до точки доступу, у випадку від'єднання.

WiFi**.**setAutoReconnect(autoReconnect)

If parameter autoReconnect is set to true, then module will try to reestablish lost connection to the AP. If set to false then module will stay disconnected.

Якщо параметр autoReconnect встановлений у значення true, модуль спробує відновити втрачене з'єднання з AP. Якщо встановлено значення false, модуль залишатиметься відключеним.

Note: running setAutoReconnect(true) when module is already disconnected will not make it reconnect to the access point. Instead reconnect() should be used.

Примітка: виклик – setAutoReconnect(true), коли модуль вже втратив з’єднання, він не під'єднається до точки доступу. Замість цього слід використати reconnect() .

### waitForConnectResult

Wait until module connects to the access point. This function is intended for module configured in station or station + soft access point mode.

Зачекайте, поки модуль підключиться до точки доступу. Ця функція призначена для модуля, налаштованого в режимі станції або станції + обмежена точка доступу.

WiFi**.**waitForConnectResult()

Function returns one of the following connection statuses: \* WL\_CONNECTED after successful connection is established \* WL\_NO\_SSID\_AVAILin case configured SSID cannot be reached \* WL\_CONNECT\_FAILED if password is incorrect \* WL\_IDLE\_STATUSwhen Wi-Fi is in process of changing between statuses \* WL\_DISCONNECTED if module is not configured in station mode

Функція повертає один з наступних станів підключення:

WL\_CONNECTED після встановлення успішного з'єднання

WL\_NO\_SSID\_AVAIL Конфікований ідентифікатор SSID не може бути досягнутий

WL\_CONNECT\_FAILED, якщо пароль невірний

WL\_IDLE\_STATUS , якщо Wi-Fi переходить між станами

WL\_DISCONNECTED, якщо модуль не налаштовано режим станції

## Configuration

### macAddress

Get the MAC address of the ESP station’s interface.

Отримайте MAC-адресу станції ESP.

WiFi**.**macAddress(mac)

Function should be provided with mac that is a pointer to memory location (an uint8\_t array the size of 6 elements) to save the mac address. The same pointer value is returned by the function itself.

Для функції повинна бути передбачена змінна - mac, яка є вказівником на розташування пам'яті (масив uint8\_t розміром 6 елементів) для збереження адреси mac. Той самий вказівник змінної повертається самою функцією.

Приклад коду:

**if** (WiFi**.**status() **==** WL\_CONNECTED)

{

uint8\_t macAddr[6];

WiFi**.**macAddress(macAddr);

Serial**.**printf("Connected, mac address: %02x:%02x:%02x:%02x:%02x:%02x\n", macAddr[0], macAddr[1], macAddr[2], macAddr[3], macAddr[4], macAddr[5]);

}

Приклад виводу:

Mac address: 5C:CF:7F:08:11:17

If you do not feel comfortable with pointers, then there is optional version of this function available. Instead of the pointer, it returns a formatted String that contains the same mac address.

Якщо ви не відчуваєте себе комфортно працюючи з вказівниками, тоді є додаткова версія цієї функції. Замість вказівника, вона повертає форматований рядок - String , що містить ту саму адресу mac.

WiFi**.**macAddress()

Приклад коду:

**if** (WiFi**.**status() **==** WL\_CONNECTED)

{

Serial**.**printf("Connected, mac address: %s\n", WiFi**.**macAddress()**.**c\_str());

}

### localIP

Function used to obtain IP address of ESP station’s interface.

Функція, що використовується для отримання IP-адреси станції ESP.

WiFi**.**localIP()

The type of returned value is [IPAddress](https://github.com/esp8266/Arduino/blob/master/cores/esp8266/IPAddress.h). There is a couple of methods available to display this type of data. They are presented in examples below that cover description of subnetMask, gatewayIP and dnsIP that return the IPAdress as well.

Тип поверненого значення - [IPAddress](https://github.com/esp8266/Arduino/blob/master/cores/esp8266/IPAddress.h). Існує декілька методів, доступних для відображення цього типу даних. Вони представлені в прикладах нижче, які містять описи subnetMask, gatewayIP та dnsIP, які також повертають IPAdress.

Приклад коду:

**if** (WiFi**.**status() **==** WL\_CONNECTED)

{

Serial**.**print("Connected, IP address: ");

Serial**.**println(WiFi**.**localIP());

}

Приклад виводу:

Connected, IP address: 192.168**.**1.10

### subnetMask

Get the subnet mask of the station’s interface.

Для отримання маски підмережі станції.

WiFi**.**subnetMask()

Module should be connected to the access point to obtain the subnet mask.

Модуль потрібно підключити до точки доступу, щоб отримати маску підмережі.

Приклад коду:

Serial**.**print("Subnet mask: ");

Serial**.**println(WiFi**.**subnetMask());

Приклад виводу:

Subnet mask: 255.255**.**255.0

### gatewayIP

Get the IP address of the gateway.

Для отримання IP-адреси шлюзу.

WiFi**.**gatewayIP()

Приклад коду:

Serial**.**printf("Gataway IP: %s\n", WiFi**.**gatewayIP()**.**toString()**.**c\_str());

Приклад виводу:

Gataway IP: 192.168**.**1.9

### dnsIP

Get the IP addresses of Domain Name Servers (DNS).

Для отримання IP-адреси серверів доменних імен (DNS).

WiFi**.**dnsIP(dns\_no)

With the input parameter dns\_no we can specify which Domain Name Server’s IP we need. This parameter is zero based and allowed values are none, 0 or 1. If no parameter is provided, then IP of DNS #1 is returned.

За допомогою вхідного параметра dns\_no ми можемо вказати, який саме Domain Name Server’s IP нам потрібен. Цей параметр може бути проігнорованим або приймати значення 0 або 1. Якщо параметр не заданий, то повертається IP-адреса DNS №1.

Приклад коду:

Serial**.**print("DNS #1, #2 IP: ");

WiFi**.**dnsIP()**.**printTo(Serial);

Serial**.**print(", ");

WiFi**.**dnsIP(1)**.**printTo(Serial);

Serial**.**println();

Приклад виводу:

DNS *#1, #2 IP: 62.179.1.60, 62.179.1.61*

### hostname

Get the DHCP hostname assigned to ESP station.

Для отримання ім'я хосту DHCP, призначеного для станції ESP.

WiFi**.**hostname()

Function returns String type. Default hostname is in format ESP\_24xMACwhere 24xMAC are the last 24 bits of module’s MAC address.

Функція повертає тип String . Ім'я хосту за умовчанням знаходиться у форматі ESP\_24xMAC, де 24xMAC є останніми 24 бітами MAC-адреси модуля.

The hostname may be changed using the following function:

Ім'я хосту може бути змінено за допомогою такої функції:

WiFi**.**hostname(aHostname)

Input parameter aHostname may be a type of char\*, const char\* or String. Maximum length of assigned hostname is 32 characters. Function returns either true or falsedepending on result. For instance, if the limit of 32 characters is exceeded, function will return false without assigning the new hostname.

Вхідний параметр aHostname повинен бути типу char\*, const char\* або String. Максимальна довжина призначеного імені хоста – 32 символів. Функція повертає або true , або false, залежно від результату. Наприклад, якщо обмеження до 32 символів перевищено, функція поверне false, не призначивши нове ім'я хосту.

Приклад коду:

Serial**.**printf("Default hostname: %s\n", WiFi**.**hostname()**.**c\_str());

WiFi**.**hostname("Station\_Tester\_02");

Serial**.**printf("New hostname: %s\n", WiFi**.**hostname()**.**c\_str());

Приклад виводу:

Default hostname: ESP\_081117

New hostname: Station\_Tester\_02

### status

Return the status of Wi-Fi connection.

Для повернення статусу з'єднання Wi-Fi.

WiFi**.**status()

Function returns one of the following connection statuses: \* WL\_CONNECTED after successful connection is established \* WL\_NO\_SSID\_AVAILin case configured SSID cannot be reached \* WL\_CONNECT\_FAILED if password is incorrect \* WL\_IDLE\_STATUSwhen Wi-Fi is in process of changing between statuses \* WL\_DISCONNECTED if module is not configured in station mode

Функція повертає один з наступних станів підключення:

WL\_CONNECTED - після встановлення успішного з'єднання

WL\_NO\_SSID\_AVAIL - у даному випадку неможливо налаштувати SSID

WL\_CONNECT\_FAILED - пароль невірний

WL\_IDLE\_STATUS - Wi-Fi переходить між станами

WL\_DISCONNECTED - якщо модуль не налаштовано в режим станції

Returned value is type of wl\_status\_t defined in [wl\_definitions.h](https://github.com/esp8266/Arduino/blob/master/libraries/ESP8266WiFi/src/include/wl_definitions.h)

Повертається значення типу wl\_status\_t, визначений у [wl\_definitions.h](https://github.com/esp8266/Arduino/blob/master/libraries/ESP8266WiFi/src/include/wl_definitions.h)

Приклад коду:

*#include <ESP8266WiFi.h>*

void setup(void)

{

Serial**.**begin(115200);

Serial**.**printf("Connection status: %d\n", WiFi**.**status());

Serial**.**printf("Connecting to %s\n", ssid);

WiFi**.**begin(ssid, password);

Serial**.**printf("Connection status: %d\n", WiFi**.**status());

**while** (WiFi**.**status() **!=** WL\_CONNECTED)

{

delay(500);

Serial**.**print(".");

}

Serial**.**printf("\nConnection status: %d\n", WiFi**.**status());

Serial**.**print("Connected, IP address: ");

Serial**.**println(WiFi**.**localIP());

}

void loop() {}

Приклад виводу:

Connection status: 6

Connecting to sensor**-**net

Connection status: 6

**......**

Connection status: 3

Connected, IP address: 192.168**.**1.10

Particular connection statuses 6 and 3 may be looked up in [wl\_definitions.h](https://github.com/esp8266/Arduino/blob/master/libraries/ESP8266WiFi/src/include/wl_definitions.h) as follows:

Конкретні стани з'єднання 6 і 3, можуть бути розглянуті в [wl\_definitions.h](https://github.com/esp8266/Arduino/blob/master/libraries/ESP8266WiFi/src/include/wl_definitions.h) наступним чином:

3 **-** WL\_CONNECTED

6 **-** WL\_DISCONNECTED

Basing on this example, when running above code, module is initially disconnected from the network and returns connection status 6 - WL\_DISCONNECTED. It is also disconnected immediately after running WiFi.begin(ssid, password). Then after about 3 seconds (basing on number of dots displayed every 500ms), it finally gets connected returning status 3 - WL\_CONNECTED.

Виходячи з цього прикладу, під час виконання вищезазначеного коду модуль спочатку буде відключений від мережі та поверне статус підключення **6** - WL\_DISCONNECTED. Він також не встигне підключитися відразу після запуску WiFi.begin(ssid, password). Потім приблизно через 3 секунди (на основі кількості точок, які відображається кожні 500 мс), він, нарешті, з’єднається і поверне статус **3** - WL\_CONNECTED.

### SSID

Return the name of Wi-Fi network, formally called [Service Set Identification (SSID)](http://www.juniper.net/techpubs/en_US/network-director1.1/topics/concept/wireless-ssid-bssid-essid.html#jd0e34).

Повертайте назву мережі Wi-Fi, яка формально називається [Service Set Identification (SSID)](http://www.juniper.net/techpubs/en_US/network-director1.1/topics/concept/wireless-ssid-bssid-essid.html#jd0e34).

WiFi**.**SSID()

Returned value is of the String type.

Повернене значення має тип String.

Приклад коду:

Serial**.**printf("SSID: %s\n", WiFi**.**SSID()**.**c\_str());

Приклад виводу:

SSID: sensor**-**net

### psk

Return current pre shared key (password) associated with the Wi-Fi network.

Повертає поточний попередньо узгоджений ключ (пароль), пов'язаний із мережею Wi-Fi.

WiFi**.**psk()

Function returns value of the String type.

Функція повертає значення типу String.

### BSSID

Return the mac address the access point where ESP module is connected to. This address is formally called [Basic Service Set Identification (BSSID)](http://www.juniper.net/techpubs/en_US/network-director1.1/topics/concept/wireless-ssid-bssid-essid.html#jd0e47).

Повертає mac адресу точки доступу, до якої підключено модуль ESP. Ця адреса формально називається [Basic Service Set Identification (BSSID)](http://www.juniper.net/techpubs/en_US/network-director1.1/topics/concept/wireless-ssid-bssid-essid.html#jd0e47).

WiFi**.**BSSID()

The BSSID() function returns a pointer to the memory location (an uint8\_t array with the size of 6 elements) where the BSSID is saved.

Функція BSSID() повертає вказівник на розташування пам'яті (масив uint8\_t з розміром 6 елементів), де зберігається BSSID.

Below is similar function, but returning BSSID but as a Stringtype.

Нижче наведено подібну функцію, що повертає BSSID як тип String.

WiFi**.**BSSIDstr()

Приклад коду:

Serial**.**printf("BSSID: %s\n", WiFi**.**BSSIDstr()**.**c\_str());

Приклад виводу:

BSSID: 00:1A:70:DE:C1:68

### RSSI

Return the signal strength of Wi-Fi network, that is formally called [Received Signal Strength Indication (RSSI)](https://en.wikipedia.org/wiki/Received_signal_strength_indication).

Для повернення рівня сигналу мережі Wi-Fi, що формально називається [Received Signal Strength Indication (RSSI)](https://uk.wikipedia.org/wiki/RSSI).

WiFi**.**RSSI()

Signal strength value is provided in dBm. The type of returned value is int32\_t.

Значення сили сигналу надається в [dBm](https://uk.wikipedia.org/wiki/DBm). Тип поверненого значення - int32\_t.

Приклад коду:

Serial**.**printf("RSSI: %d dBm\n", WiFi**.**RSSI());

Приклад виводу:

RSSI: **-**68 dBm

## Connect Different

[ESP8266 SDK](http://bbs.espressif.com/viewtopic.php?f=51&t=1023) provides alternate methods to connect ESP station to an access point. Out of them [esp8266 / Arduino](https://github.com/esp8266/Arduino) core implements [WPS](http://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/station-class.html#wps) and [Smart Config](http://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/station-class.html#smart-config) as described in more details below.

SDK ESP8266 надає альтернативні способи підключення станції ESP до точки доступу. З них ядро [esp8266 / Arduino](https://github.com/esp8266/Arduino) реалізує WPS та Smart Config, про які більш докладно описано нижче.

### WPS

The following beginWPSConfig function allows connecting to a network using [Wi-Fi Protected Setup (WPS)](https://en.wikipedia.org/wiki/Wi-Fi_Protected_Setup). Currently only [push-button configuration](http://www.wi-fi.org/knowledge-center/faq/how-does-wi-fi-protected-setup-work) (WPS\_TYPE\_PBC mode) is supported (SDK 1.5.4).

Наступна функція beginWPSConfig дозволяє підключитися до мережі за допомогою [Wi-Fi Protected Setup (WPS)](https://en.wikipedia.org/wiki/Wi-Fi_Protected_Setup). Наразі (SDK 1.5.4) підтримується лише [push-button](https://www.wi-fi.org/knowledge-center/faq/how-does-wi-fi-protected-setup-work) конфігурація (режим WPS\_TYPE\_PBC).

WiFi**.**beginWPSConfig()

Depending on connection result function returns either trueor false (boolean type).

Залежно від підключення результат повертає або true, або false (тип bool).

Приклад коду:

#include <ESP8266WiFi.h>

void setup(void)

{

Serial.begin(115200);

Serial.println();

Serial.printf("Wi-Fi mode set to WIFI\_STA %s\n", WiFi.mode(WIFI\_STA) ? "" : "Failed!");

Serial.print("Begin WPS (press WPS button on your router) ... ");

Serial.println(WiFi.beginWPSConfig() ? "Success" : "Failed");

while (WiFi.status() != WL\_CONNECTED)

{

delay(500);

Serial.print(".");

}

Serial.println();

Serial.print("Connected, IP address: ");

Serial.println(WiFi.localIP());

}

void loop() {}

Приклад виводу:

Wi**-**Fi mode set to WIFI\_STA

Begin WPS (press WPS button on your router) **...** Success

**.........**

Connected, IP address: 192.168**.**1.102

### Smart Config

The Smart Config connection of an ESP module an access point is done by sniffing for special packets that contain SSID and password of desired AP. To do so the mobile device or computer should have functionality of broadcasting of encoded SSID and password.

Підключення Smart Config модуля ESP до точки доступу здійснюється шляхом вишукування спеціальних пакетів, що містять ідентифікатор SSID та пароль потрібної AP. Для цього мобільний пристрій або комп'ютер повинен мати функціональність трансляції закодованого SSID та пароля.

The following three functions are provided to implement Smart Config.

Для реалізації Smart Config надаються три функції.

Start smart configuration mode by sniffing for special packets that contain SSID and password of desired Access Point. Depending on result either true or `false is returned.

Запустіть режим *smart configuration* для пошуку спеціальних пакетів, які містять SSID та пароль потрібної точки доступу. Залежно від результату повертається або true, або false.

beginSmartConfig()

Query Smart Config status, to decide when stop configuration. Function returns either true or false ofboolean` type.

Запросіть статус *Smart Config*, щоб вирішити, коли зупинити конфігурацію. Функція повертає або true , або false  типу bool.

smartConfigDone()

Stop smart config, free the buffer taken by beginSmartConfig(). Depending on result function return either true or false of boolean type.

Для зупинки *smart config*, звільніть буфер за допомогою функції – beginSmartConfig(). Залежно від результату, функція поверне або true, або false типу bool.

stopSmartConfig()

For additional details regarding Smart Config please refer to [ESP8266 API User Guide](http://bbs.espressif.com/viewtopic.php?f=51&t=1023).

Додаткові відомості про Smart Config можна знайти в [smartconfig.h](https://github.com/esp8266/Arduino/blob/master/tools/sdk/include/smartconfig.h).

**EXAMPLES**

# Station

Example of connecting to an access point has been shown in chapter [Quick Start](http://arduino-esp8266.readthedocs.io/en/2.4.0/esp8266wifi/readme.md#quick-start). In case connection is lost, ESP8266 will automatically reconnect to the last used access point, once it is again available.

Приклад підключення до точки доступу показано в главі Quick Start. Якщо з'єднання буде втрачено, ESP8266 автоматично відновить з'єднання з останньою використаною точкою доступу, коли вона знову стане доступною.

Can we provide more robust connection to Wi-Fi than that?

Чи можемо ми забезпечити більш надійне підключення до Wi-Fi, ніж це?

## Introduction

Following the example in [Quick Start](http://arduino-esp8266.readthedocs.io/en/2.4.0/esp8266wifi/readme.md#quick-start), we would like to go one step further and made ESP connect to next available access point if current connection is lost. This functionality is provided with ‘ESP8266WiFiMulti’ class and demonstrated in sketch below.

Слідуючи прикладу в " Quick Start ", ми хотіли б зробити ще один крок і змусили ESP підключитися до наступної доступної точки доступу, якщо поточне з'єднання буде втрачено. Ця функціональність забезпечується класом 'ESP8266WiFiMulti' і демонструється в прикладі нижче.

*#include <ESP8266WiFi.h>*

*#include <ESP8266WiFiMulti.h>*

ESP8266WiFiMulti wifiMulti;

boolean connectioWasAlive **=** true;

void setup()

{

Serial**.**begin(115200);

Serial**.**println();

wifiMulti**.**addAP("primary-network-name", "pass-to-primary-network");

wifiMulti**.**addAP("secondary-network-name", "pass-to-secondary-network");

wifiMulti**.**addAP("tertiary-network-name", "pass-to-tertiary-network");

}

void monitorWiFi()

{

**if** (wifiMulti**.**run() **!=** WL\_CONNECTED)

{

**if** (connectioWasAlive **==** true)

{

connectioWasAlive **=** false;

Serial**.**print("Looking for WiFi ");

}

Serial**.**print(".");

delay(500);

}

**else** **if** (connectioWasAlive **==** false)

{

connectioWasAlive **=** true;

Serial**.**printf(" connected to %s\n", WiFi**.**SSID()**.**c\_str());

}

}

void loop()

{

monitorWiFi();

}

## Prepare Access Points

To try this sketch in action you need two (or more) access points. In lines below replace primary-network-name and pass-to-primary-network with name and password to your primary network. Do the same for secondary network.

Щоб спробувати цей ескіз у дії, потрібні дві (або більше) точки доступу. У рядки primary-network-name і pass-to-primary-network потрібно вставити ім'я і пароль вашої першої мережі. Зробіть те ж саме для наступних мереж.

wifiMulti**.**addAP("primary-network-name", "pass-to-primary-network");

wifiMulti**.**addAP("secondary-network-name", "pass-to-secondary-network");

You may add more networks if you have more access points.

Ви можете додати ще більше мереж, якщо це потрібно.

wifiMulti**.**addAP("tertiary-network-name", "pass-to-tertiary-network");

**...**

## Try it Out

Now upload updated sketch to ESP module and open serial monitor. Module will first scan for available networks. Then it will select and connect to the network with stronger signal. In case connection is lost, module will connect to next one available.

Тепер завантажте оновлену програму до модуля ESP та відкрийте послідовний монітор. Модуль спочатку сканує наявні мережі. Потім він обере і підключатися до мережі з кращим сигналом. Якщо з'єднання буде втрачено, модуль підключиться до наступного.

This process may look something like:

Цей процес буде виглядати приблизно так:

Looking **for** WiFi **.....** connected to sensor**-**net**-**1

Looking **for** WiFi **.......** connected to sensor**-**net**-**2

Looking **for** WiFi **....** connected to sensor**-**net**-**1

In above example ESP connected first to sensor-net-1. Then I have switched sensor-net-1 off. ESP discovered that connection is lost and started searching for another configured network. That happened to be sensor-net-2 so ESP connected to it. Then I have switched sensor-net-1 back on and shut down sensor-net-2. ESP reconnected automatically to sensor-net-1.

У наведеному вище прикладі ESP спочатку з'єднано з sensor-net-1. Після вимкнення sensor-net-1 ESP виявив, що зв'язок втрачено, і почав шукати іншу знайому мережу. Через невеличку затримку ESP підключилося до sensor-net-2. Потім я знову включив sensor-net-1 і погасив sensor-net-2. ESP автоматично з'єдналося до sensor-net-1.

Function monitorWiFi() is in place to show when connection is lost by displaying Looking for WiFi. Dots .... are displayed during process of searching for another configured access point. Then a message like connected to sensor-net-2 is shown when connection is established.

Функція monitorWiFi() написана з метою показати, коли з'єднання буде втрачено, надрукувавши Looking for WiFi. Точки .... відображаються під час пошуку іншої відомої точки доступу. Після встановлення з'єднання з'являється таке повідомлення – connected to sensor-net-2 .

## Can we Make it Simpler?

Please note that you may simplify this sketch by removing function monitorWiFi() and putting inside loop() only wifiMulti.run(). ESP will still reconnect between configured access points if required. Now you won’t be able to see it on serial monitor unless you add Serial.setDebugOutput(true) as described in point [Enable Wi-Fi Diagnostic](http://arduino-esp8266.readthedocs.io/en/2.4.0/esp8266wifi/readme.md#enable-wi-fi-diagnostic).

Зверніть увагу, що ви можете спростити цей приклад, видаливши функцію monitorWiFi() і перевіряти з’єднання в циклі loop() за допомогою – wifiMulti.run(). ESP, як і раніше, при необхідності, буде перепідключатися між налагодженими точками доступу. Тепер ви не зможете прослідкувати за процесом перепід’єднання в послідовному моніторі, якщо ви не додасте – Serial.setDebugOutput(true), як описано в пункті Enable Wi-Fi Diagnostic.

Updated sketch for such scenario will look as follows:

Оновлений приклад для такого сценарію буде виглядати наступним чином:

*#include <ESP8266WiFi.h>*

*#include <ESP8266WiFiMulti.h>*

ESP8266WiFiMulti wifiMulti;

void setup()

{

Serial**.**begin(115200);

Serial**.**setDebugOutput(true);

Serial**.**println();

wifiMulti**.**addAP("primary-network-name", "pass-to-primary-network");

wifiMulti**.**addAP("secondary-network-name", "pass-to-secondary-network");

wifiMulti**.**addAP("tertiary-network-name", "pass-to-tertiary-network");

}

void loop()

{

wifiMulti**.**run();

}

That’s it! This is really all the code you need to make ESP automatically reconnecting between available networks.

Як бачите, нічого складного! Це дійсно весь код, який потрібно зробити для автоматичного підключення ESP між доступними мережами.

After uploading sketch and opening the serial monitor, the messages will look as below.

Після завантаження скетчу та відкриття послідовного монітора повідомлення виглядатимуть як показано нижче.

Initial connection to sensor-net-1 on power up:

*Початкове підключення до* sensor-net-1 *при подачі живлення:*

f r0, scandone

f r0, scandone

state: 0 **->** 2 (b0)

state: 2 **->** 3 (0)

state: 3 **->** 5 (10)

add 0

aid 1

cnt

chg\_B1:**-**40

connected **with** sensor**-**net**-**1, channel 1

dhcp client start**...**

ip:192.168**.**1.10,mask:255.255**.**255.0,gw:192.168**.**1.9

Lost connection to sensor-net-1 and establishing connection to sensor-net-2:

Втрата зв’язку з sensor-net-1 та встановлення з'єднання з sensor-net-2:

bcn\_timout,ap\_probe\_send\_start

ap\_probe\_send over, rest wifi status to disassoc

state: 5 **->** 0 (1)

rm 0

f r**-**40, scandone

f r**-**40, scandone

f r**-**40, scandone

state: 0 **->** 2 (b0)

state: 2 **->** 3 (0)

state: 3 **->** 5 (10)

add 0

aid 1

cnt

connected **with** sensor**-**net**-**2, channel 11

dhcp client start**...**

ip:192.168**.**1.102,mask:255.255**.**255.0,gw:192.168**.**1.234

Втрата зв’язку з sensor-net-2 та встановлення з'єднання з sensor-net-1:

bcn\_timout,ap\_probe\_send\_start

ap\_probe\_send over, rest wifi status to disassoc

state: 5 **->** 0 (1)

rm 0

f r**-**40, scandone

f r**-**40, scandone

f r**-**40, scandone

state: 0 **->** 2 (b0)

state: 2 **->** 3 (0)

state: 3 **->** 5 (10)

add 0

aid 1

cnt

connected **with** sensor**-**net**-**1, channel 6

dhcp client start**...**

ip:192.168**.**1.10,mask:255.255**.**255.0,gw:192.168**.**1.9

## Conclusion

I believe the minimalist sketch with ESP8266WiFiMulti class is a cool example what ESP8266 can do for us behind the scenes with just couple lines of code.

Я вважаю, що мінімалістський приклад класу ESP8266WiFiMulti є гарним прикладом можливостей, які ESP8266 може приховано зробити для нас, лише завдяки декількох рядків коду.

As shown in above example, reconnecting between access points takes time and is not seamless. Therefore, in practical applications, you will likely need to monitor connection status to decide e.g. if you can send the data to external system or should wait until connection is back.

Як показано в прикладі вище, повторне підключення між точкою доступу вимагає часу і не є безперервним. Тому, в практичних додатках, вам, імовірно, буде потрібно стежити за статусом зв'язку, щоб уникнути непередбачених перепід’єднань.