# Scan Class

This class is represented in [Arduino WiFi library](https://www.arduino.cc/en/Reference/WiFi) by [scanNetworks()](https://www.arduino.cc/en/Reference/WiFiScanNetworks) function. Developers of esp8266 / Arduino core extend this functionality by additional methods and properties.

Цей клас представлений у бібліотеці [Arduino WiFi](https://www.arduino.cc/en/Reference/WiFi) як фукція [scanNetworks()](https://www.arduino.cc/en/Reference/WiFiScanNetworks). Розробники esp8266 / Arduino core розширюють цю функціональність додатковими методами та властивостями.

Documentation of this class is divided into two parts. First covers functions to scan for available networks. Second describes what information is collected during scanning process and how to access it.

Документація цього класу поділяється на дві частини. Перша охоплює функції для сканування доступних мереж. Друга описує, яку інформацію можна зібрати, під час сканування, та як отримати доступ до неї.

# Scan for Networks

Scanning for networks takes hundreds of milliseconds to complete. This may be done in a single run when we are triggering scan process, waiting for completion, and providing result - all by a single function. Another option is to split this into steps, each done by a separate function. This way we can execute other tasks while scanning is in progress. This is called asynchronous scanning. Both methods of scanning are documented below.

Сканування мережі вимагає сотні мілісекунд. Це може бути зроблено в однин прогін, коли ми запускаємо процес сканування, очікуємо завершення, і забезпечуємо результат - все це здійснюється за допомогою однієї функції. Інший варіант полягає в тому, щоб розділити це на кроки, кожен з яких виконується окремою функцією. Таким чином, ми можемо виконувати інші завдання під час сканування. Це називається асинхронним скануванням. Обидва способи сканування описані нижче.

## scanNetworks

Scan for available Wi-Fi networks in one run and return the number of networks that has been discovered.

Для сканування доступних мереж Wi-Fi за один прогін і повертає кількість виявлених мереж.

WiFi**.**scanNetworks()

There is on [overload](https://en.wikipedia.org/wiki/Function_overloading) of this function that accepts two optional parameters to provide extended functionality of asynchronous scanning as well as looking for hidden networks.

Існує [перевантаження](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) цієї функції, яке приймає два необов'язкових параметри, щоб забезпечити розширену функціональність асинхронного сканування, а також шукати приховані мережі.

WiFi**.**scanNetworks(**async**, show\_hidden)

Both function parameters are of boolean type. They provide the flowing functionality: \* asysnc - if set to true then scanning will start in background and function will exit without waiting for result. To check for result use separate function scanComplete that is described below. \* show\_hidden - set it to true to include in scan result networks with hidden SSID.

Обидва параметри функції типу boolean . Вони забезпечують наступну функціональність:

asysnc - якщо встановлено значення true, то сканування буде розпочато у фоновому режимі, а функція вийде без очікування результату. Для перевірки результатів використовується окрема функція scanComplete, яка описана нижче.

show\_hidden - встановіть його – true для включення в результати пошуку мережі з прихованим SSID.

## scanComplete

Check for result of asynchronous scanning.

Для перевірки результату асинхронного сканування.

WiFi**.**scanComplete()

On scan completion function returns the number of discovered networks.

Після завершення сканування функція повертає кількість виявлених мереж.

If scan is not done, then returned value is < 0 as follows: \* Scanning still in progress: -1 \* Scanning has not been triggered: -2

Якщо сканування не завершене, то повернене значення буде менше нуля та буде означати:

Сканування досі виконується: -1 \* Сканування не було розпочате: -2

## scanDelete

Delete the last scan result from memory.

Для видалення останнього результату сканування з пам'яті.

WiFi**.**scanDelete()

## scanNetworksAsync

Start scanning for available Wi-Fi networks. On completion execute another function.

Для початку сканування наявних мереж Wi-Fi. Після завершення виконується інша функція.

WiFi**.**scanNetworksAsync(onComplete, show\_hidden)

Function parameters: \* onComplete - the event handler executed when the scan is done

\* show\_hidden - optional boolean parameter, set it to trueto scan for hidden networks

Параметри функції:

onComplete - обробник подій, який виконується під час сканування

show\_hidden - необов'язковий boolean  параметр, встановіть його true для сканування для прихованих мереж

Приклад коду:

#include "ESP8266WiFi.h"

void prinScanResult(int networksFound)

{

Serial.printf("%d network(s) found\n", networksFound);

for (int i = 0; i < networksFound; i++)

{

Serial.printf("%d: %s, Ch:%d (%ddBm) %s\n", i + 1, WiFi.SSID(i).c\_str(), WiFi.channel(i), WiFi.RSSI(i), WiFi.encryptionType(i) == ENC\_TYPE\_NONE ? "open" : "");

}

}

void setup()

{

Serial.begin(115200);

Serial.println();

WiFi.mode(WIFI\_STA);

WiFi.disconnect();

delay(100);

WiFi.scanNetworksAsync(prinScanResult);

}

void loop() {}

Приклад виводу:

5 network(s) found

1: Tech\_D005107, Ch:6 (**-**72dBm)

2: HP**-**Print**-**A2**-**Photosmart 7520, Ch:6 (**-**79dBm)

3: ESP\_0B09E3, Ch:9 (**-**89dBm) open

4: Hack**-**4**-**fun**-**net, Ch:9 (**-**91dBm)

5: UPC Wi**-**Free, Ch:11 (**-**79dBm)

# Show Results

Functions below provide access to result of scanning. It does not matter if scanning has been done in synchronous or asynchronous mode, scan results are available using the same API.

Наведені нижче функції забезпечують доступ до результату сканування. Неважливо, чи сканування було виконано в синхронному або асинхронному режимі, результати сканування доступні за допомогою того самого API.

Individual results are accessible by providing a `networkItem’ that identifies the index (zero based) of discovered network.

Індивідуальні результати доступні шляхом надання «networkItem», який ідентифікує індекс (з нуля) виявленої мережі.

## SSID

Return the SSID of a network discovered during the scan.

Для повернення SSID мережі, виявленої під час сканування.

WiFi**.**SSID(networkItem)

Returned SSID is of the String type. The networkItem is a zero based index of network discovered during scan.

Повернутий SSID має тип String. Змінна – networkItem  є індексом мережі, виявленої під час сканування.

## encryptionType

Return the encryption type of a network discovered during the scan.

Повертайте тип шифрування мережі, знайденої під час сканування.

WiFi**.**encryptionType(networkItem)

Function returns a number that encodes encryption type as follows: \* 5 : ENC\_TYPE\_WEP - WEP \* 2 : ENC\_TYPE\_TKIP - WPA / PSK \* 4 : ENC\_TYPE\_CCMP - WPA2 / PSK \* 7 : ENC\_TYPE\_NONE - open network \* 8 : ENC\_TYPE\_AUTO - WPA / WPA2 / PSK

Функція повертає номер, який означає такі типи шифрування:

5: ENC\_TYPE\_WEP - WEP

2 : ENC\_TYPE\_TKIP - WPA / PSK

4 : ENC\_TYPE\_CCMP - WPA2 / PSK

7 : ENC\_TYPE\_NONE - відкрита мережа

8 : ENC\_TYPE\_AUTO - WPA / WPA2 / PSK

The networkItem is a zero based index of network discovered during scan.

Змінна – networkItem  є індексом мережі, виявленої під час сканування.

## RSSI

Return the [RSSI](https://en.wikipedia.org/wiki/Received_signal_strength_indication) (Received Signal Strength Indication) of a network discovered during the scan.

Для повернення [RSSI](https://uk.wikipedia.org/wiki/RSSI) (Received Signal Strength Indication – сила сигналу) мережі, виявленої під час сканування.

WiFi**.**RSSI(networkItem)

Returned RSSI is of the int32\_t type. The networkItem is a zero based index of network discovered during scan.

Отриманий RSSI має тип int32\_t. Змінна – networkItem  є індексом мережі, виявленої під час сканування.

## BSSID

Return the [BSSID](https://en.wikipedia.org/wiki/Service_set_(802.11_network)#Basic_service_set_identification_.28BSSID.29) (Basic Service Set Identification) that is another name of MAC address of a network discovered during the scan.

Для повернення <BSSID> (Basic Service Set Identification), яка є іншою назвою MAC-адреси мережі, виявленої під час сканування.

WiFi**.**BSSID(networkItem)

Function returns a pointer to the memory location (an uint8\_tarray with the size of 6 elements) where the BSSID is saved.

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

If you do not like to pointers, then there is another version of this function that returns a String.

Якщо вам не подобаються вказівники, то є ще одна версія цієї функції, яка повертає String.

WiFi**.**BSSIDstr(networkItem)

The networkItem is a zero based index of network discovered during scan.

Змінна – networkItem  є індексом мережі, виявленої під час сканування.

## channel

Return the channel of a network discovered during the scan.

Для повернення каналу мережі, виявленої під час сканування.

WiFi**.**channel(networkItem)

Returned channel is of the int32\_t type. The networkItem is a zero based index of network discovered during scan.

Повернутий канал має тип int32\_t. Змінна – networkItem  є індексом мережі, виявленої під час сканування.

## isHidden

Return information if a network discovered during the scan is hidden or not.

Для інформації чи була виявлена мережа прихованою, під час сканування, чи ні.

WiFi**.**isHidden(networkItem)

Returned value if the bolean type, and true means that network is hidden. The networkItem is a zero based index of network discovered during scan.

Повертається значення bolean  типу, і відповідь true означає, що мережа прихована. Змінна – networkItem  є індексом мережі, виявленої під час сканування.

## getNetworkInfo

Return all the network information discussed in this chapter above in a single function call.

Для повернення всієї мережевої інформації, описану в цій главі вище, в одному виклику функції.

WiFi**.**getNetworkInfo(networkItem, **&**ssid, **&**encryptionType, **&**RSSI, **\*&**BSSID, **&**channel, **&**isHidden)

The networkItem is a zero based index of network discovered during scan. All other input parameters are passed to function by reference. Therefore they will be updated with actual values retrieved for particular networkItem. The function itself returns boolean true or false to confirm if information retrieval was successful or not.

Змінна – networkItem  є індексом мережі, виявленої під час сканування. Всі інші вхідні параметри передаються для роботи за посиланням. Тому вони будуть оновлюватися з фактичними значеннями, отриманими для певної мережі. Сама функція повертає логічне значення true або false для підтвердження успішності пошуку інформації.

Приклад коду:

int n = WiFi.scanNetworks(false, true);

String ssid;

uint8\_t encryptionType;

int32\_t RSSI;

uint8\_t\* BSSID;

int32\_t channel;

bool isHidden;

for (int i = 0; i < n; i++)

{

WiFi.getNetworkInfo(i, ssid, encryptionType, RSSI, BSSID, channel, isHidden);

Serial.printf("%d: %s, Ch:%d (%ddBm) %s %s\n", i + 1, ssid.c\_str(), channel, RSSI, encryptionType == ENC\_TYPE\_NONE ? "open" : "", isHidden ? "hidden" : "");

}

Приклад виводу:

6 network(s) found

1: Tech\_D005107, Ch:6 (**-**72dBm)

2: HP**-**Print**-**A2**-**Photosmart 7520, Ch:6 (**-**79dBm)

3: ESP\_0B09E3, Ch:9 (**-**89dBm) open

4: Hack**-**4**-**fun**-**net, Ch:9 (**-**91dBm)

5: , Ch:11 (**-**77dBm) hidden

6: UPC Wi**-**Free, Ch:11 (**-**79dBm)

For code samples please refer to separate section with [examples](http://arduino-esp8266.readthedocs.io/en/2.4.0/esp8266wifi/scan-examples.md) dedicated specifically to the Scan Class.

**EXAMPLES**

**Scan**

To connect a mobile phone to a hot spot, you typically open Wi-Fi settings app, list available networks and then pick the hot spot you need. You can also list the networks with ESP8266 and here is how.

Щоб підключити мобільний телефон до точки доступу, ви зазвичай відкриваєте налаштування Wi-Fi, скануєте доступні мережі та потім вибираєте потрібну вам. Аналогічно ви можете перерахувати мережі за допомогою ESP8266.

**Simple Scan**

This example shows the bare minimum code we need to check for the list of available networks.

Цей приклад показує, як перевірити список доступних мереж з мінімальною кількістю коду.

**Disconnect**

To start with, enable module in station mode and then disconnect.

Щоб почати, увімкніть модуль у режимі станції, а потім відключіть його.

WiFi**.**mode(WIFI\_STA);

WiFi**.**disconnect();

Running WiFi.disconnect() is to shut down a connection to an access point that module may have automatically made using previously saved credentials.

Запуск команди – Wi-Fi.disconnect() розриває з'єднання з точкою доступу, яке модуль міг автоматично встановити, використовуючи раніше збережені облікові дані.

**Scan for Networks**

After some delay to let the module disconnect, go to scanning for available networks:

Після деякої затримки після розриву зв’язку, перейдіть до сканування доступних мереж:

int n **=** WiFi**.**scanNetworks();

Now just check if returned n if greater than 0 and list found networks:

Тепер просто перевірте значення n (чи воно більше 0), та список знайдених мереж:

**for** (int i **=** 0; i **<** n; i**++**)

{

Serial**.**println(WiFi**.**SSID(i));

}

This is that simple.

Це робиться ось так просто.

**Complete Example**

The sketch should have obligatory #include <ESP8266WiFi.h> and looks as follows:

Приклад повинен обов'язково містити рядок #include <ESP8266WiFi.h> і це все буде виглядати наступним чином:

*#include "ESP8266WiFi.h"*

void setup()

{

Serial**.**begin(115200);

Serial**.**println();

WiFi**.**mode(WIFI\_STA);

WiFi**.**disconnect();

delay(100);

}

void loop()

{

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

int n **=** WiFi**.**scanNetworks();

Serial**.**print(n);

Serial**.**println(" network(s) found");

**for** (int i **=** 0; i **<** n; i**++**)

{

Serial**.**println(WiFi**.**SSID(i));

}

Serial**.**println();

delay(5000);

}

**Example in Action**

Upload this sketch to ESP module and open a serial monitor. If there are access points around (sure there are) you will see a similar list repeatedly printed out:

Завантажте цей скетч у модуль ESP та відкрийте послідовний монітор. Якщо є точки доступу (певно, що є), ви побачите подібний список, який періодично друкуватиметься:

Scan start **...** 5 network(s) found

Tech\_D005107

HP**-**Print**-**A2**-**Photosmart 7520

ESP\_0B09E3

Hack**-**4**-**fun**-**net

UPC Wi**-**Free

When looking for the text scan start ... displayed, you will notice that it takes noticeable time for the following text n network(s) found to show up. This is because execution of WiFi.scanNetworks() takes time and our program is waiting for it to complete before moving to the next line of code. What if at the same time we would like ESP to run time critical process (e.g. animation) that should not be disturbed?

Коли ви переглядаєте повідомлення scan start ... , ви помітите, що знадобиться помітний час для появи наступного повідомлення n network(s) found . Це відбувається тому, що виконання WiFi.scanNetworks() займає деякий час, і наша програма чекає його завершення, перш ніж перейти до наступного рядка коду. Що робити, якщо в той же час ми хочемо, щоб ESP виконував критичний процес по часу (наприклад, анімацію), який не слід порушувати?

It turns out that this is fairly easy to do by scanning networks in async mode.

Виявляється, це досить легко зробити, скануючи мережі в режимі асинхронного режиму.

Check it out in next example below that will also demonstrate printing out other parameters of available networks besides SSID.

Перевірте це в наступному прикладі нижче, який також продемонструє друк інших параметрів доступних мереж окрім SSID.

**Async Scan**

What we like to do, is to trigger process of scanning for networks and then return to executing code inside the loop(). Once scanning is complete, at a convenient time, we will check the list of networks. The “time critical process” will be simulated by a blinking LED at 250ms period.

Що ми хочемо зробити - запустити процес сканування для мереж, а потім повернутися до виконання коду всередині циклу loop(). Після завершення сканування, в зручний час, ми перевіримо список мереж. "Критичний процес часу" буде імітуватися блиманням світлодіодним світлом з періодичністю 250 мс.

We would like the blinking pattern not be disturbed at any time.

Ми бажаємо, щоб миготіння відбувалося весь час, і не залежало від роботи WiFi.

**No delay()**

To implement such functionality we should refrain from using any delay() inside the loop(). Instead we will define period when to trigger particular action. Then inside loop() we will check millis() (internal clock that counts milliseconds) and fire the action if the period expires.

Щоб реалізувати таку функціональність, ми повинні утримуватися від використання будь-якої затримки delay()  у циклі loop(). Замість цього ми визначимо період, коли необхідно виконати певні дії. Потім всередині циклу loop() ми звіримося з часом за допомогою – millis() (внутрішній годинник, який розраховує мілісекунди) і запусктимо дію, якщо виділений час минув.

Please check how this is done in [BlinkWithoutDelay.ino](http://arduino-esp8266.readthedocs.io/en/2.4.0/esp8266wifi/BlinkWithoutDelay.ino) example sketch. Identical technique can be used to periodically trigger scanning for Wi-Fi networks.

Будь ласка, перегляньте, як це робиться в прикладі [BlinkWithoutDelay.ino](http://arduino-esp8266.readthedocs.io/en/2.4.0/esp8266wifi/BlinkWithoutDelay.ino). Ідентична методика може використовуватися для періодичного запуску сканування для мереж Wi-Fi.

**Setup**

First we should define scanning period and internal variable lastScanMillis that will hold time when the last scan has been made.

Спочатку потрібно визначити період сканування та внутрішню змінну lastScanMillis, яка буде запам’ятовувати час, коли було здійснено останнє сканування.

*#define SCAN\_PERIOD 5000*

long lastScanMillis;

**When to Start**

Then inside the loop() we will check if SCAN\_PERIOD expired, so it is time to fire next scan:

Потім всередині циклу loop() ми перевіримо, чи період сканувань SCAN\_PERIOD минув, якщо так то прийшов час запустити наступне сканування:

**if** (currentMillis **-** lastScanMillis **>** SCAN\_PERIOD)

{

WiFi**.**scanNetworks(true);

Serial**.**print("\nScan start ... ");

lastScanMillis **=** currentMillis;

}

Please note that WiFi.scanNetworks(true) has an extra parameter true that was not present in [previous example](http://arduino-esp8266.readthedocs.io/en/2.4.0/esp8266wifi/scan-examples.html#simple-scan) above. This is an instruction to scan in asynchronous mode, i.e. trigger scanning process, do not wait for result (processing will be done in background) and move to the next line of code. We need to use asynchronous mode otherwise 250ms LED blinking pattern would be disturbed as scanning takes longer than 250ms.

Зверніть увагу, що WiFi.scanNetworks(true) має додатковий параметр true, який не був присутній у попередньому прикладі вище. Це є інструкція для сканування в асинхронному режимі, тобто тригерний процес сканування, не чекатиме результату (обробка буде виконуватися у фоновому режимі) і переміститися до наступного рядка коду. Нам неоюхідно використовувати асинхронний режим, інакше періодичне світлодіодне мерехтіння буде порушуватись, оскільки сканування займає більше 250мс.

**Check When Done**

Finally we should periodically check for scan completion to print out the result once ready. To do so, we will use function WiFi.scanComplete(), that upon completion returns the number of found networks. If scanning is still in progress it returns -1. If scanning has not been triggered yet, it would return -2.

Нарешті, ми повинні періодично перевіряти завершення сканування, щоб вивести результат після готовності. Для цього ми використаємо функцію WiFi.scanComplete(), яка після завершення повертає кількість знайдених мереж. Якщо сканування все ще триває, він повертає -1. Якщо сканування не було запущене, він поверне -2.

int n = WiFi.scanComplete();

if(n >= 0)

{

Serial.printf("%d network(s) found\n", n);

for (int i = 0; i < n; i++)

{

Serial.printf("%d: %s, Ch:%d (%ddBm) %s\n", i+1, WiFi.SSID(i).c\_str(), WiFi.channel(i), WiFi.RSSI(i), WiFi.encryptionType(i) == ENC\_TYPE\_NONE ? "open" : "");

}

WiFi.scanDelete();

}

Please note function WiFi.scanDelete() that is deleting scanning result from memory, so it is not printed out over and over again on each loop() run.

Будь ласка, зверніть увагу на функцію WiFi.scanDelete(), яка видаляє результат сканування з пам'яті, щоб він не друкувався знову і знову по кожному запуску циклу loop() .

**Complete Example**

Complete sketch is below. The code inside setup() is the same as described in [previous example](http://arduino-esp8266.readthedocs.io/en/2.4.0/esp8266wifi/scan-examples.html#simple-scan) except for an additional pinMode() to configure the output pin for LED.

Повний приклад наведено нижче. Код всередині setup() аналогічний тому, що описано в попередньому прикладі, за винятком додаткового pinMode() для налаштування вихідного піна для світлодіодів.

#include "ESP8266WiFi.h"

#define BLINK\_PERIOD 250

long lastBlinkMillis;

boolean ledState;

#define SCAN\_PERIOD 5000

long lastScanMillis;

void setup()

{

Serial.begin(115200);

Serial.println();

pinMode(LED\_BUILTIN, OUTPUT);

WiFi.mode(WIFI\_STA);

WiFi.disconnect();

delay(100);

}

void loop()

{

long currentMillis = millis();

// blink LED

if (currentMillis - lastBlinkMillis > BLINK\_PERIOD)

{

digitalWrite(LED\_BUILTIN, ledState);

ledState = !ledState;

lastBlinkMillis = currentMillis;

}

// trigger Wi-Fi network scan

if (currentMillis - lastScanMillis > SCAN\_PERIOD)

{

WiFi.scanNetworks(true);

Serial.print("\nScan start ... ");

lastScanMillis = currentMillis;

}

// print out Wi-Fi network scan result uppon completion

int n = WiFi.scanComplete();

if(n >= 0)

{

Serial.printf("%d network(s) found\n", n);

for (int i = 0; i < n; i++)

{

Serial.printf("%d: %s, Ch:%d (%ddBm) %s\n", i+1, WiFi.SSID(i).c\_str(), WiFi.channel(i), WiFi.RSSI(i), WiFi.encryptionType(i) == ENC\_TYPE\_NONE ? "open" : "");

}

WiFi.scanDelete();

}

}

**Example in Action**

Upload above sketch to ESP module and open a serial monitor. You should see similar list printed out every 5 seconds:

Завантажте цей скетч в ESP та відкрийте послідовний монітор. Ви повинні побачити подібний список, надрукований кожні 5 секунд:

Scan start **...** 5 network(s) found

1: Tech\_D005107, Ch:6 (**-**72dBm)

2: HP**-**Print**-**A2**-**Photosmart 7520, Ch:6 (**-**79dBm)

3: ESP\_0B09E3, Ch:9 (**-**89dBm) open

4: Hack**-**4**-**fun**-**net, Ch:9 (**-**91dBm)

5: UPC Wi**-**Free, Ch:11 (**-**79dBm)

Check the LED. It should be blinking undisturbed four times per second.

Перевірте світлодіод. Він повинен блимати чотири рази в секунду.

**Conclusion**

The scan class API provides comprehensive set of methods to do scanning in both synchronous as well as in asynchronous mode. Therefore we can easy implement code that is doing scanning in background without disturbing other processes running on ESP8266 module.

API класу сканування надає повний набір методів сканування як в синхронному, так і в асинхронному режимі. Тому ми можемо легко застосувати код, який виконує сканування у фоновому режимі, не порушуючи інших процесів, що працюють на модулі ESP8266.

For the list of functions provided to manage scan mode please refer to the [Scan Class](http://arduino-esp8266.readthedocs.io/en/2.4.0/esp8266wifi/scan-class.md) documentation.