# Server Class

Methods documented for the [Server Class](https://www.arduino.cc/en/Reference/WiFiServerConstructor) in [Arduino](https://github.com/arduino/Arduino)

Методи для класу [Server Class](https://www.arduino.cc/en/Reference/WiFiServerConstructor) документовані в [Arduino](https://github.com/arduino/Arduino)

1. [WiFiServer()](https://www.arduino.cc/en/Reference/WiFiServer)

Створює об’єкт сервера, який слухає вхідні з'єднання на вказаному порту.

WiFiServer server(80);

Створює об’єкт сервера, який слухає вхідні з'єднання на вказаному порту.

Змінна – port це номер порту для прослуховування, типу int

1. [begin()](https://www.arduino.cc/en/Reference/WiFiServerBegin)

Для налаштування сервера на початок прослуховування вхідних з'єднань.

server.begin();

Нічого не поветає

1. [available()](https://www.arduino.cc/en/Reference/WiFiServerAvailable)

Для отримання клієнта, який підключено до сервера і має дані, доступні для читання. З'єднання зберігається, коли повернений об'єкт клієнта виходить з області видимості; Ви можете завершити його, викликавши client.stop().

available() успадкований від допоміжного класу [Stream](https://www.arduino.cc/en/Reference/Stream).

server.available()

Щоб дізнатися кількість байтів, доступних для читання (тобто кількість даних, написаних клієнтом на сервері, до якого він підключений).

Повертає об'єкт Client типу WiFiClient, якщо жоден Client не має даних для читання, цей об'єкт оцінюватиметься як false для оператора – if(client)

1. [write()](https://www.arduino.cc/en/Reference/WiFiServerWrite)

Для передачі даних всім клієнтам, підключених до сервера.

client.write(data)

Змінна – data це один байт інформації типу byte або char.

Повертає кількість написаних символів в типі byte.

1. [print()](https://www.arduino.cc/en/Reference/WiFiServerPrint)

Для надсилання даних у вигляді масиву символів на сервер, до якого підключено клієнт. Цей метод буде надсилати числа як послідовність символів ASCII (наприклад, номер 123 надсилається як три символи '1', '2', '3').

client.print(data)

Існує перевантаження функції яке задає формат виводу чисел, таких як DEC, OCT, HEX, BIN.

client.print(data, BASE)

Зміст параметрів – наступний:

data: дані для друку (char, byte, int, long, string)

BASE – база, в якій друкуватимуться номери : DEC для десяткової (база 10), OCT для вісімкової (база 8), HEX для шістнадцяткового (база 16), BIN для двійкової (база 2).

Повертає кількість написаних символів в типі byte.

1. [println()](https://www.arduino.cc/en/Reference/WiFiServerPrintln)

Аналогічний сенс що і в print() з відмінністю лише в тому, що в кінець повідомлення додається символ нового рядку ‘\n’.

#### Може використовуватися трьома різними перевантаженнями, наприклад:

client.println()   
client.println(data)   
client.println(data, BASE)

data: дані для друку (char, byte, int, long, string)

BASE – база, в якій друкуватимуться номери : DEC для десяткової (база 10), OCT для вісімкової (база 8), HEX для шістнадцяткового (база 16), BIN для двійкової (база 2).

Повертає кількість написаних символів в типі byte.

Methods and properties described further down are specific to ESP8266. They are not covered in [Arduino WiFi library](https://www.arduino.cc/en/Reference/WiFi) documentation. Before they are fully documented please refer to information below.

Методи та властивості, описані нижче, є специфічними для ESP8266. Вони не розглядаються в документації [Arduino WiFi library](https://www.arduino.cc/en/Reference/WiFi). Перш ніж вони повністю документовані, будь ласка, зверніться до інформації нижче.

## setNoDelay

setNoDelay(nodelay)

With nodelay set to true, this function will to disable [Nagle algorithm](https://en.wikipedia.org/wiki/Nagle%27s_algorithm).

Якщо nodelay налаштовано на значення true, ця функція відключить [Nagle algorithm](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9D%D0%B5%D0%B9%D0%B3%D0%BB%D0%B0).

This algorithm is intended to reduce TCP/IP traffic of small packets sent over the network by combining a number of small outgoing messages, and sending them all at once. The downside of such approach is effectively delaying individual messages until a big enough packet is assembled.

This algorithm is intended to reduce TCP/IP traffic of small packets sent over the network by combining a number of small outgoing messages, and sending them all at once. The downside of such approach is effectively delaying individual messages until a big enough packet is assembled.

Цей алгоритм призначений для зменшення трафіку TCP/IP для невеликих пакетів, що надсилаються по мережі, об'єднуючи їх в ряд невеликих вихідних повідомлень і відправляючи їх усім одночасно. Недоліком такого підходу є затримка окремих повідомлень, допоки не буде зібрано досить великий пакет.

Приклад:

server**.**begin();

server**.**setNoDelay(true);

## Other Function Calls

bool hasClient ()

bool getNoDelay ()

virtual size\_t write (const uint8\_t **\***buf, size\_t size)

uint8\_t status ()

void close ()

void stop ()

Documentation for the above functions is not yet prepared.

Документація для вищезгаданих функцій ще не підготовлена, але ви можете зрозуміти як вони влаштовані та як з ними працювати переглянувши вихідний код (ви також можете допомогти розширити цю документацію).

За шляхом нижче ви знайдете бібліотеку WiFiServer (h та cpp файли).

C:\Users\(ВАШ\_ПРОФІЛЬ)\AppData\Local\Arduino15\packages\esp8266\hardware\esp8266\2.4.0\libraries\ESP8266WiFi\src\

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

**EXAMPLES**

# Server

Setting up web a server on ESP8266 requires very little code and is surprisingly straightforward. This is thanks to functionality provided by the versatile ESP8266WiFi library.

Налаштування веб-сервера на ESP8266 вимагає дуже мало коду і напрочуд просто. Це завдяки функціональності, наданій універсальною бібліотекою ESP8266WiFi.

The purpose of this example will be to prepare a web page that can be opened in a web browser. This page should show the current raw reading of ESP’s analog input pin.

Мета цього прикладу - підготувати веб-сторінку, яку можна відкрити в веб-браузері. Ця сторінка повинна відображати поточне необроблене читання аналогового входу АЦП ESP.

## The Object

We will start off by creating a server object.

Почнемо, створивши об'єкт серверу.

WiFiServer server(80);

The server responds to clients (in this case - web browsers) on port 80, which is a standard port web browsers talk to web servers.

Сервер реагує на клієнтів (у цьому випадку - веб-браузери) на порту 80, який є стандартним портом веб-браузера, який спілкується з веб-серверами.

## The Page

Then let’s write a short function prepareHtmlPage(), that will return a String class variable containing the contents of the web page. We will then pass this variable to server to pass it over to a client.

Давайте напишемо коротку функцію prepareHtmlPage(), яка повертає змінну класу String, що містить вміст веб-сторінки. Після цього ми передаємо цю змінну серверу, щоб передати її клієнту.

String prepareHtmlPage()

{

String htmlPage **=**

String("HTTP/1.1 200 OK\r\n") **+**

"Content-Type: text/html\r\n" **+**

"Connection: close\r\n" **+** **//** the connection will be closed after completion of the response

"Refresh: 5\r\n" **+** **//** refresh the page automatically every 5 sec

"\r\n" **+**

"<!DOCTYPE HTML>" **+**

"<html>" **+**

"Analog input: " **+** String(analogRead(A0)) **+**

"</html>" **+**

"\r\n";

**return** htmlPage;

}

The function does nothing fancy but just puts together a text header and [HTML](http://www.w3schools.com/html/) contents of the page.

Функція не робить нічого фантастичного, а лише об'єднує текстовий заголовок та [HTML](https://www.w3schools.com/html/default.asp)-вміст сторінки.

## Header First

The header is to inform client what type of contents is to follow and how it will be served:

Заголовок полягає в тому, щоб інформувати клієнта про те, який тип вмісту слід дотримуватися та як він буде подаватися:

Content**-**Type: text**/**html

Connection: close

Refresh: 5

In our example the content type is text/html, the connection will be closed after serving and the content should be requested by the client again every 5 seconds. The header is concluded with an empty line \r\n. This is to distinguish header from the content to follow.

У нашому прикладі тип вмісту - text/html, зв'язок буде завершено після відправлення, а вміст має бути запрошений повторно кожні 5 секунд. Заголовок завершується порожньою лінією \r\n. Це робиться для того, щоб відрізнити заголовоки від звичайного тексту.

<!DOCTYPE HTML>

<html>

Analog input: [Value]

</html>

The content contains two basic [HTML](http://www.w3schools.com/html/) tags, one to denote HTML document type <!DOCTYPE HTML> and another to mark beginning <html> and end </html> of the document. Inside there is a raw value read from ESP’s analog input analogRead(A0)converted to the String type.

Зміст містить дві основні [HTML](http://www.w3schools.com/html/)-теги, одина для позначення типу документа HTML <! DOCTYPE HTML>, а інша - для позначення початку <html> та закінчення </html> документа. Всередині є необроблена інформація, що була прочитана з аналогового вводу ESP analogRead(A0), перетворена в тип String.

String(analogRead(A0))

## The Page is Served

Serving of this web page will be done in the loop() where server is waiting for a new client to connect and send some data containing a request:

Обслуговування цієї веб-сторінки буде виконано в циклі loop(), де сервер чекає підключення нового клієнта та надсилає деякі данні, що містять запит:

void loop()

{

WiFiClient client **=** server**.**available();

**if** (client)

{

**//** we have a new client sending some request

}

}

Once a new client is connected, server will read the client’s request and print it out on a serial monitor.

Після підключення нового клієнта сервер буде читати запит клієнта та друкувати його в послідовному моніторі.

**while** (client**.**connected())

{

**if** (client**.**available())

{

String line **=** client**.**readStringUntil('\r');

Serial**.**print(line);

}

}

Request from the client is marked with an empty new line. If we find this mark, we can send back the web page and exit while()loop using break.

Запит від клієнта позначається порожньою новою лінією. Якщо ми знайдемо цю позначку, ми можемо відправити назад веб-сторінку та вийти з циклу while(), використовуючи break.

**if** (line**.**length() **==** 1 **&&** line[0] **==** '\n')

{

client**.**println(prepareHtmlPage());

**break**;

}

The whole process is concluded by stopping the connection with client:

Весь процес завершується, розривом зв'язку з клієнтом:

client**.**stop();

## Put it Together

Complete sketch is presented below.

Повний приклад представлений нижче.

*#include <ESP8266WiFi.h>*

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

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

WiFiServer server(80);

void setup()

{

Serial**.**begin(115200);

Serial**.**println();

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

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

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

{

delay(500);

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

}

Serial**.**println(" connected");

server**.**begin();

Serial**.**printf("Web server started, open %s in a web browser\n", WiFi**.**localIP()**.**toString()**.**c\_str());

}

**//** prepare a web page to be send to a client (web browser)

String prepareHtmlPage()

{

String htmlPage **=**

String("HTTP/1.1 200 OK\r\n") **+**

"Content-Type: text/html\r\n" **+**

"Connection: close\r\n" **+** **//** the connection will be closed after completion of the response

"Refresh: 5\r\n" **+** **//** refresh the page automatically every 5 sec

"\r\n" **+**

"<!DOCTYPE HTML>" **+**

"<html>" **+**

"Analog input: " **+** String(analogRead(A0)) **+**

"</html>" **+**

"\r\n";

**return** htmlPage;

}

void loop()

{

WiFiClient client **=** server**.**available();

**//** wait **for** a client (web browser) to connect

**if** (client)

{

Serial**.**println("\n[Client connected]");

**while** (client**.**connected())

{

**//** read line by line what the client (web browser) **is** requesting

**if** (client**.**available())

{

String line **=** client**.**readStringUntil('\r');

Serial**.**print(line);

**//** wait **for** end of client's request, that is marked with an empty line

**if** (line**.**length() **==** 1 **&&** line[0] **==** '\n')

{

client**.**println(prepareHtmlPage());

**break**;

}

}

}

delay(1); **//** give the web browser time to receive the data

**//** close the connection:

client**.**stop();

Serial**.**println("[Client disonnected]");

}

}

## Get it Run

Update ssid and password in sketch to match credentials of your access point. Load sketch to ESP module and open a serial monitor. First you should see confirmation that module connected to the access point and the web server started.

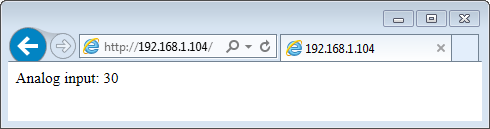
Оновіть ssid та пароль у скетчі, щоб вони відповідали обліковим даним вашої точки доступу. Завантажте скетч до модуля ESP та відкрийте послідовний монітор. Спочатку ви повинні побачити підтвердження того, що модуль підключений до точки доступу та веб-сервера.

Connecting to sensor**-**net **........** connected

Web server started, open 192.168**.**1.104 **in** a web browser

Enter provided IP address in a web browser. You should see the page served by ESP8266:

Введіть вказану IP-адресу в веб-браузері. Ви повинні побачити сторінку, що обслуговується ESP8266:



*alt text*

The page would be refreshed every 5 seconds. Each time this happens, you should see a request from the client (your web browser) printed out on the serial monitor:

Сторінка буде оновлюватися кожні 5 секунд. Кожного разу, коли це трапляється, ви повинні побачити запит від клієнта (вашого веб-браузера), надрукованого в послідовному моніторі:

[Client connected]

GET **/** HTTP**/**1.1

Accept: text**/**html, application**/**xhtml**+**xml, **\*/\***

Accept**-**Language: en**-**US

User**-**Agent: Mozilla**/**5.0 (Windows NT 6.1; WOW64; Trident**/**7.0; rv:11.0) like Gecko

Accept**-**Encoding: gzip, deflate

Host: 192.168**.**1.104

DNT: 1

Connection: Keep**-**Alive

[client disonnected]

## What Else?

Looking on [client examples](http://arduino-esp8266.readthedocs.io/en/2.4.0/esp8266wifi/client-examples.html) you will quickly find out the similarities in protocol to the server. The protocol starts with a header that contains information what communication will be about. It contains what content type is communicated or accepted like text/html. It states whether connection will be kept alive or closed after submission of the header. It contains identification of the sender like User-Agent: Mozilla/5.0 (Windows NT 6.1), etc.

Переглядаючи приклади клієнта, ви швидко знайдете подібність до протоколу сервера. Протокол починається з заголовка, який містить інформацію про те, яке спілкування буде. Він містить інформацію про тип вмісту, який повідомляється або приймається, як text/html. У ньому зазначено, чи буде з'єднання зберігатися живим або закриється після представлення заголовка. Він містить ідентифікацію відправника як User- Agent: Mozilla/5.0 (Windows NT 6.1), і т. Д.

## Conclusion

The above example shows that a web server on ESP8266 can be set up in almost no time. Such server can easily stand up requests from much more powerful hardware and software like a PC with a web browser. Check out other classes like [ESP8266WebServer](https://github.com/esp8266/Arduino/tree/master/libraries/ESP8266WebServer) that let you program more advanced applications.

Наведений вище приклад показує, що веб-сервер на ESP8266 можна налаштувати практично моментально. Такий сервер може легко витримати запити від набагато більш потужного обладнання та програмного забезпечення, як ПК з веб-браузером. Перевірте інші класи, такі як [ESP8266WebServer](https://github.com/esp8266/Arduino/tree/master/libraries/ESP8266WebServer), які дозволяють програмувати більш прогресивні програми.

If you like to try another server example, check out [WiFiWebServer.ino](https://github.com/esp8266/Arduino/blob/master/libraries/ESP8266WiFi/examples/WiFiWebServer/WiFiWebServer.ino), that provides functionality of toggling the GPIO pin on and off out of a web browser.

Якщо ви хочете спробувати інший приклад сервера, перегляньте програму [WiFiWebServer.ino](https://github.com/esp8266/Arduino/blob/master/libraries/ESP8266WiFi/examples/WiFiWebServer/WiFiWebServer.ino), яка забезпечує функцію перемикання станів GPIO у веб-браузері.

For the list of functions provided to implement and manage servers, please refer to the [Server Class](http://arduino-esp8266.readthedocs.io/en/2.4.0/esp8266wifi/server-class.html) documentation.