Important Information

Thank you for choosing Freenove products!

Getting Started

First, please read the **Start Here.pdf** document in the unzipped folder you created.
If you have not yet downloaded the zip file, associated with this kit, please do so now and unzip it.

Get Support and Offer Input

Freenove provides free and responsive product and technical support, including but not limited to:

- Product quality issues
- Product use and build issues
- Questions regarding the technology employed in our products for learning and education
- Your input and opinions are always welcome
- We also encourage your ideas and suggestions for new products and product improvements

For any of the above, you may send us an email to:

support@freenove.com

Safety and Precautions

Please follow the following safety precautions when using or storing this product:

- Keep this product out of the reach of children under 6 years old.
- This product should be **used only when there is adult supervision present** as young children lack necessary judgment regarding safety and the consequences of product misuse.
- This product contains small parts and parts, which are sharp. This product contains electrically conductive parts. **Use caution with electrically conductive parts near or around power supplies, batteries and powered (live) circuits.**
- When the product is turned ON, activated or tested, some parts will move or rotate. **To avoid injuries to hands and fingers keep them away from any moving parts!**
- It is possible that an improperly connected or shorted circuit may cause overheating. **Should this happen, immediately disconnect the power supply or remove the batteries and do not touch anything until it cools down!** When everything is safe and cool, review the product tutorial to identify the cause.
- Only operate the product in accordance with the instructions and guidelines of this tutorial, otherwise parts may be damaged or you could be injured.
- Store the product in a cool dry place and avoid exposing the product to direct sunlight.
- After use, always turn the power OFF and remove or unplug the batteries before storing.

About Freenove

Freenove provides open source electronic products and services worldwide.

Freenove is committed to assist customers in their education of robotics, programming and electronic circuits so that they may transform their creative ideas into prototypes and new and innovative products. To this end, our services include but are not limited to:

- Educational and Entertaining Project Kits for Robots, Smart Cars and Drones
- Educational Kits to Learn Robotic Software Systems for Arduino, Raspberry Pi and micro: bit
- Electronic Component Assortments, Electronic Modules and Specialized Tools
- **Product Development and Customization Services**

You can find more about Freenove and get our latest news and updates through our website:

http://www.freenove.com

sale@freenove.com

Copyright

FREENOVE™

Contents

Preface

ESP32 is a micro control unit with integrated Wi-Fi launched by Espressif, which features strong properties and integrates rich peripherals. It can be designed and studied as an ordinary Single Chip Micyoco(SCM) chip, or connected to the Internet and used as an Internet of Things device.

ESP32 can be developed using the Arduino platform, which will definitely make it easier for people who have learned arduino to master. Moreover, the code of ESP32 is completely open-source, so beginners can quickly learn how to develop and design IOT smart household products including smart curtains, fans, lamps and clocks.

Generally, ESP32 projects consist of code and circuits. Don't worry even if you've never learned code and circuits, because we will gradually introduce the basic knowledge of C programming language and electronic circuits, from easy to difficult. Our products contain all the electronic components and modules needed to complete these projects. It's especially suitable for beginners.

We divide each project into four parts, namely Component List, Component Knowledge, Circuit and Code. Component List helps you to prepare material for the experiment more quickly. Component Knowledge allows you to quickly understand new electronic modules or components, while Circuit helps you understand the operating principle of the circuit. And Code allows you to easily master the use of SEP32 and accessory kit. After finishing all the projects in this tutorial, you can also use these components and modules to make products such as smart household, smart cars and robots to transform your creative ideas into prototypes and new and innovative products.

In addition, if you have any difficulties or questions with this tutorial or toolkit, feel free to ask for our quick and free technical support through **support@freenove.com**

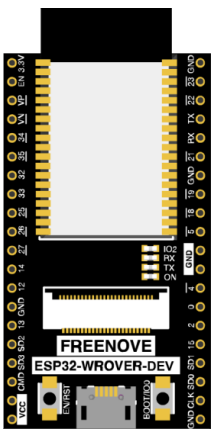
ESP32-WROVER

ESP32-WROVER has launched a total of two antenna packages, PCB on-board antenna and IPEX antenna respectively. The PCB on-board antenna is an integrated antenna in the chip module itself, so it is convenient to carry and design. The IPEX antenna is a metal antenna derived from the integrated antenna of the chip module itself, which is used to enhance the signal of the module.

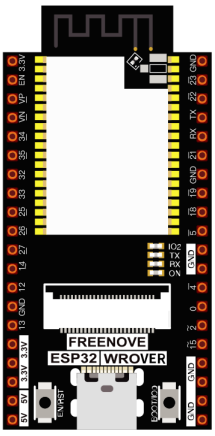
PCB on-board antenna	IPEX antenna
	

In this tutorial, the ESP32-WROVER is designed based on the PCB on-board antenna-packaged ESP32-WROVER module.

ESP32-WROVER



or

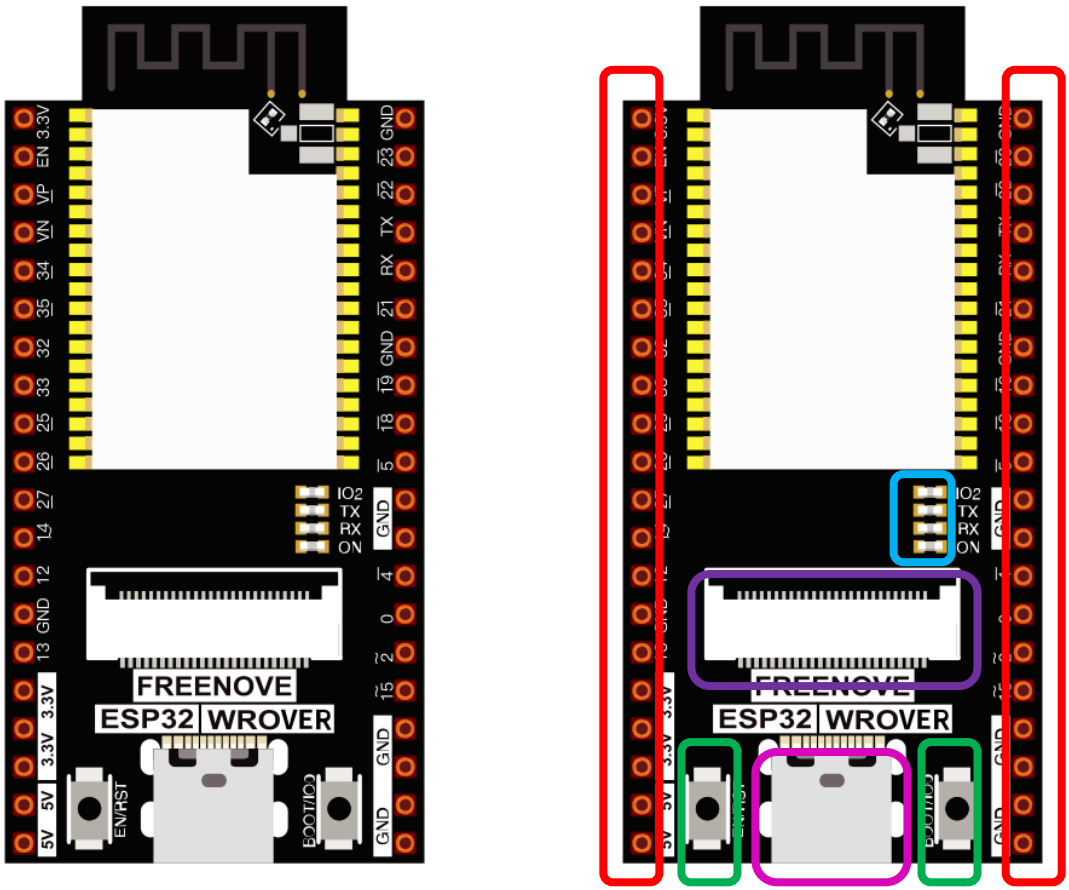


The version on the left is no longer mass-produced, and we mainly maintain the version on the right.

Please note that there are many pirated versions of the ESP32 WROVER that look very similar to the version on the left. None of them will carry our logo and Freenove font.

We do not sell pirated ESP32 WROVER, nor do we provide after-sales service for pirated.

The hardware interfaces of ESP32-WROVER are distributed as follows:



Compare the left and right images. We've boxed off the resources on the ESP32-WROVER in different colors to facilitate your understanding of the ESP32-WROVER.

Box color	Corresponding resources introduction
	GPIO pin
	LED indicator
	Camera interface
	Reset button, Boot mode selection button
	USB port

Name	No.	Type	Function
GND	1	P	Ground
3V3	2	P	Power supply
EN	3	I	Module-enable signal. Active high.
SENSOR_VP	4	I	GPIO36, ADC1_CH0, RTC_GPIO0
SENSOR_VN	5	I	GPIO39, ADC1_CH3, RTC_GPIO3
IO34	6	I	GPIO34, ADC1_CH6, RTC_GPIO4
IO35	7	I	GPIO35, ADC1_CH7, RTC_GPIO5
IO32	8	I/O	GPIO32, XTAL_32K_P (32.768 kHz crystal oscillator input), ADC1_CH4, TOUCH9, RTC_GPIO9
IO33	9	I/O	GPIO33, XTAL_32K_N (32.768 kHz crystal oscillator output), ADC1_CH5, TOUCH8, RTC_GPIO8
IO25	10	I/O	GPIO25, DAC_1, ADC2_CH8, RTC_GPIO6, EMAC_RXD0
IO26	11	I/O	GPIO26, DAC_2, ADC2_CH9, RTC_GPIO7, EMAC_RXD1
IO27	12	I/O	GPIO27, ADC2_CH7, TOUCH7, RTC_GPIO17, EMAC_RX_DV
IO14	13	I/O	GPIO14, ADC2_CH6, TOUCH6, RTC_GPIO16, MTMS, HSPICLK, HS2_CLK, SD_CLK, EMAC_TXD2
IO12 [1]	14	I/O	GPIO12, ADC2_CH5, TOUCH5, RTC_GPIO15, MTDI, HSPIQ, HS2_DATA2, SD_DATA2, EMAC_TXD3
GND	15	P	Ground
IO13	16	I/O	GPIO13, ADC2_CH4, TOUCH4, RTC_GPIO14, MTCK, HSPID, HS2_DATA3, SD_DATA3, EMAC_RX_ER
SHD/SD2 [2]	17	I/O	GPIO9, SD_DATA2, SPIHD, HS1_DATA2, U1RXD
SWP/SD3 [2]	18	I/O	GPIO10, SD_DATA3, SPIWP, HS1_DATA3, U1TXD
SCS/CMD [2]	19	I/O	GPIO11, SD_CMD, SPICS0, HS1_CMD, U1RTS
SCK/CLK [2]	20	I/O	GPIO6, SD_CLK, SPICLK, HS1_CLK, U1CTS
SDO/SD0 [2]	21	I/O	GPIO7, SD_DATA0, SPIQ, HS1_DATA0, U2RTS
SDI/SD1 [2]	22	I/O	GPIO8, SD_DATA1, SPID, HS1_DATA1, U2CTS
IO15	23	I/O	GPIO15, ADC2_CH3, TOUCH3, MTDO, HSPICS0, RTC_GPIO13, HS2_CMD, SD_CMD, EMAC_RXD3
IO2	24	I/O	GPIO2, ADC2_CH2, TOUCH2, RTC_GPIO12, HSPIWP, HS2_DATA0, SD_DATA0
IO0	25	I/O	GPIO0, ADC2_CH1, TOUCH1, RTC_GPIO11, CLK_OUT1, EMAC_TX_CLK
IO4	26	I/O	GPIO4, ADC2_CH0, TOUCH0, RTC_GPIO10, HSPIHD, HS2_DATA1, SD_DATA1, EMAC_TX_ER
NC1	27	-	-
NC2	28	-	-
IO5	29	I/O	GPIO5, VSPICS0, HS1_DATA6, EMAC_RX_CLK
IO18	30	I/O	GPIO18, VSPICLK, HS1_DATA7
IO19	31	I/O	GPIO19, VSPIQ, U0CTS, EMAC_TXD0
NC	32	-	-
IO21	33	I/O	GPIO21, VSPIHD, EMAC_TX_EN
RXD0	34	I/O	GPIO3, U0RXD, CLK_OUT2
TXD0	35	I/O	GPIO1, U0TXD, CLK_OUT3, EMAC_RXD2
IO22	36	I/O	GPIO22, VSPIWP, U0RTS, EMAC_TXD1
IO23	37	I/O	GPIO23, VSPID, HS1_STROBE
GND	38	P	Ground

Notice:

1. GPIO12 is internally pulled high in the module and is not recommended for use as a touch pin.
2. Pins SCK/CLK, SDO/SD0, SDI/SD1, SHD/SD2, SWP/SD3 and SCS/CMD, namely, GPIO6 to GPIO11 are connected to the SPI flash integrated on the module and are not recommended for other uses.

For more information, please visit: https://www.espressif.com/sites/default/files/documentation/esp32-wrover_datasheet_en.pdf

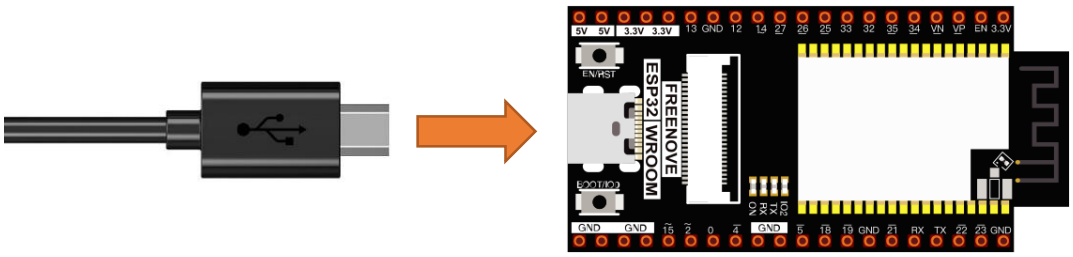
CH340 (Importance)

ESP32 uses CH340 to download codes. So before using it, we need to install CH340 driver in our computers.

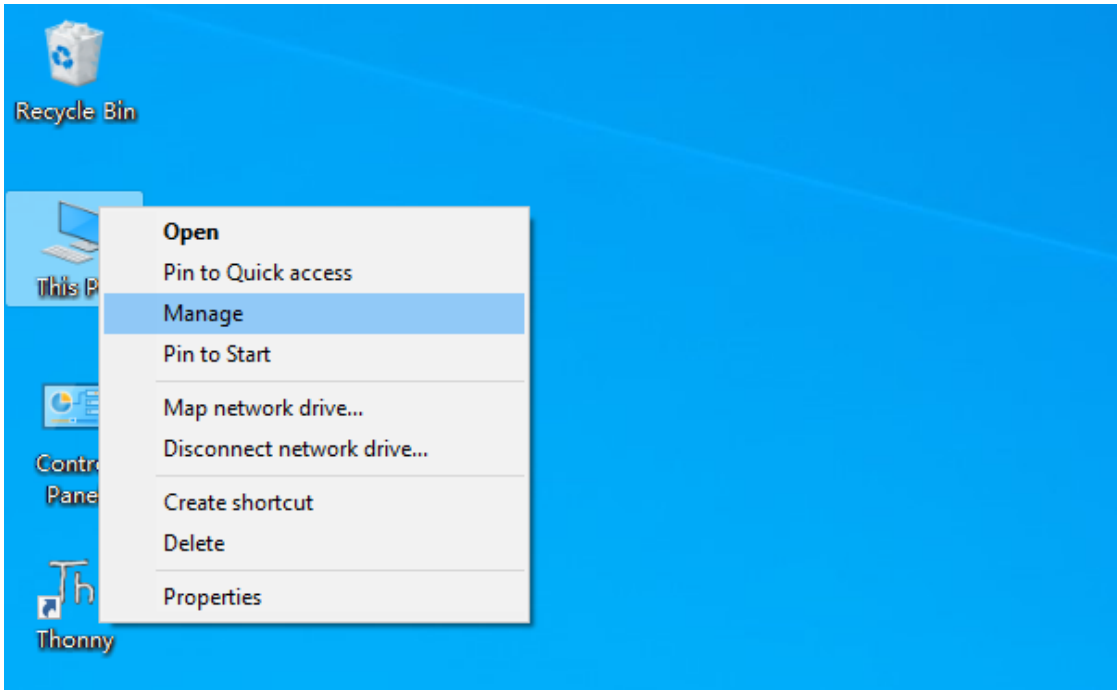
Windows

Check whether CH340 has been installed

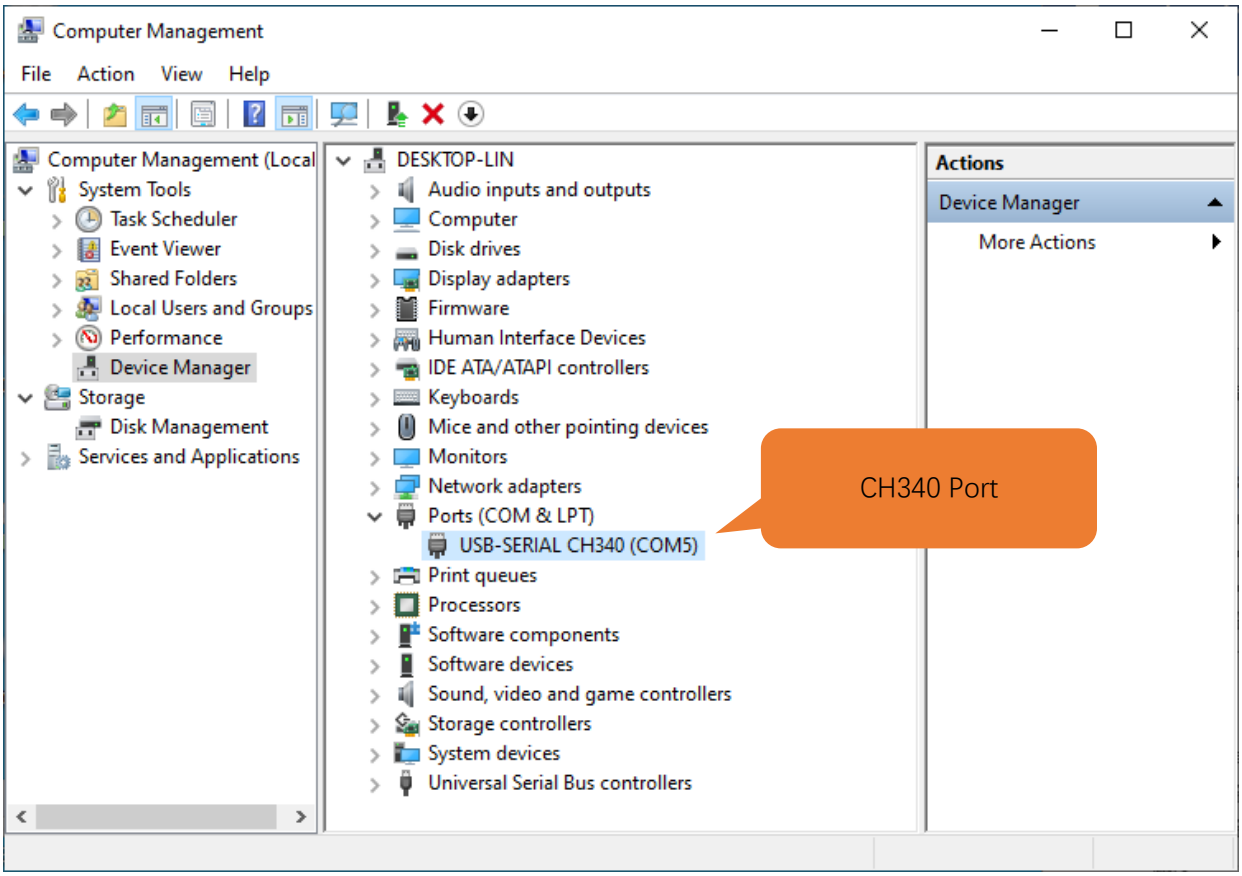
1. Connect your computer and ESP32 with a USB cable.



2. Turn to the main interface of your computer, select “This PC” and right-click to select “Manage”.

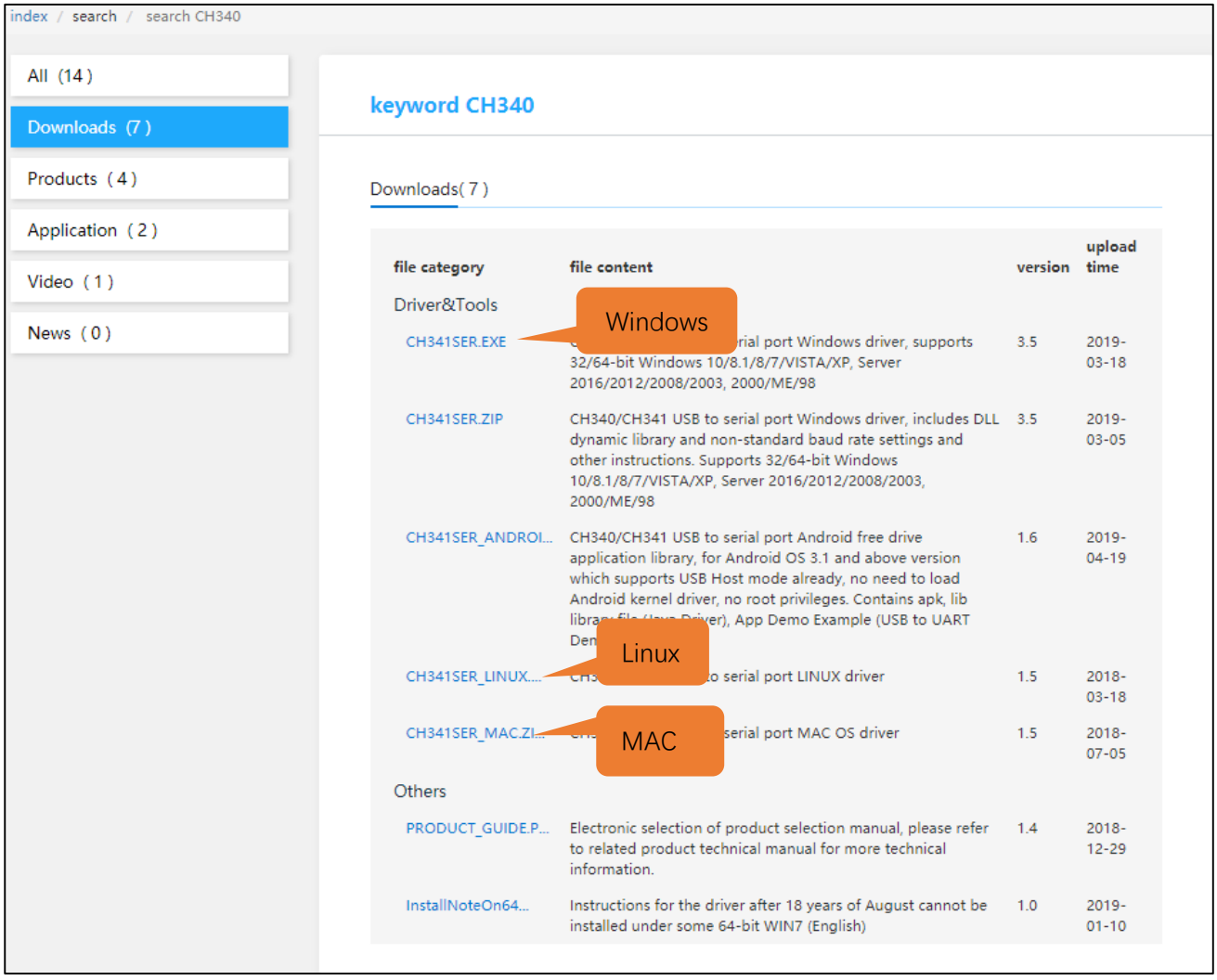


3. Click "Device Manager". If your computer has installed CH340, you can see"USB-SERIAL CH340 (COMx)". And you can click here to move to the next step.

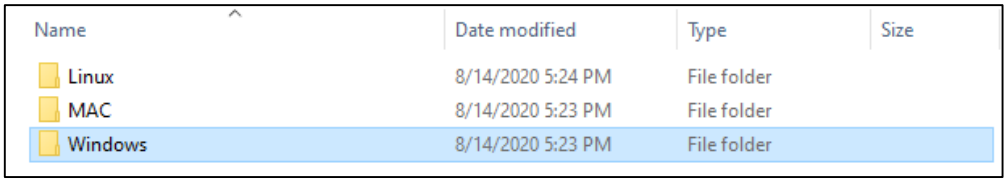


Installing CH340

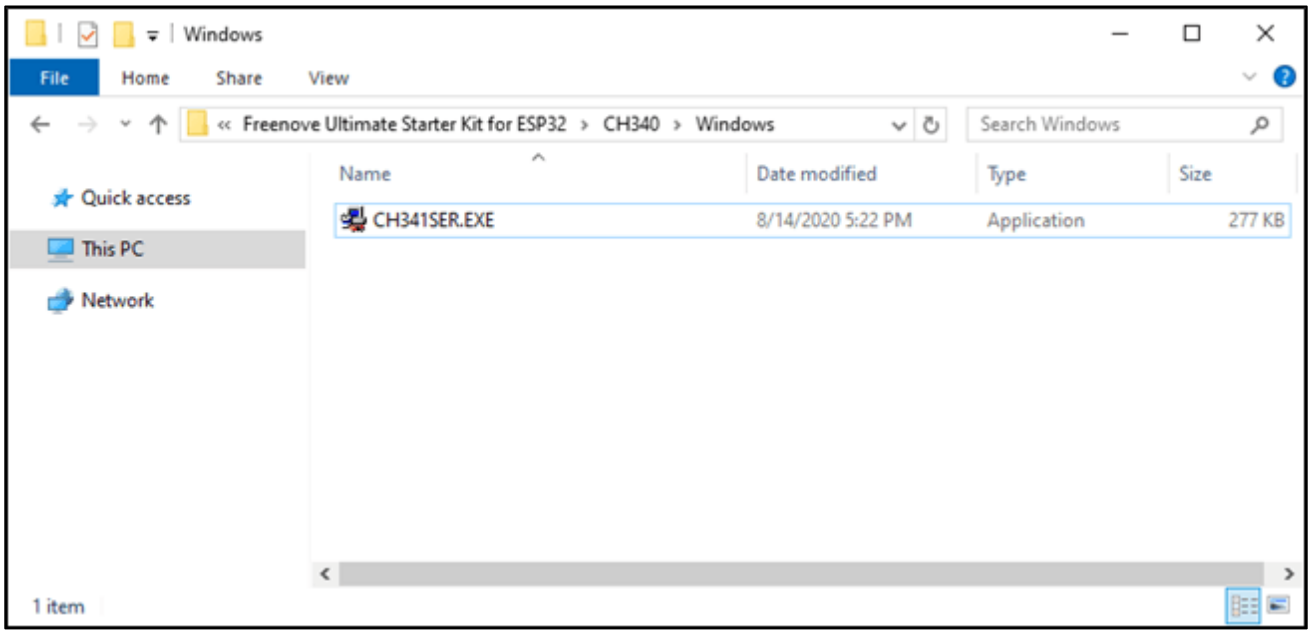
1. First, download CH340 driver, click http://www.wch-ic.com/search?q=CH340&t=downloads to download the appropriate one based on your operating system.



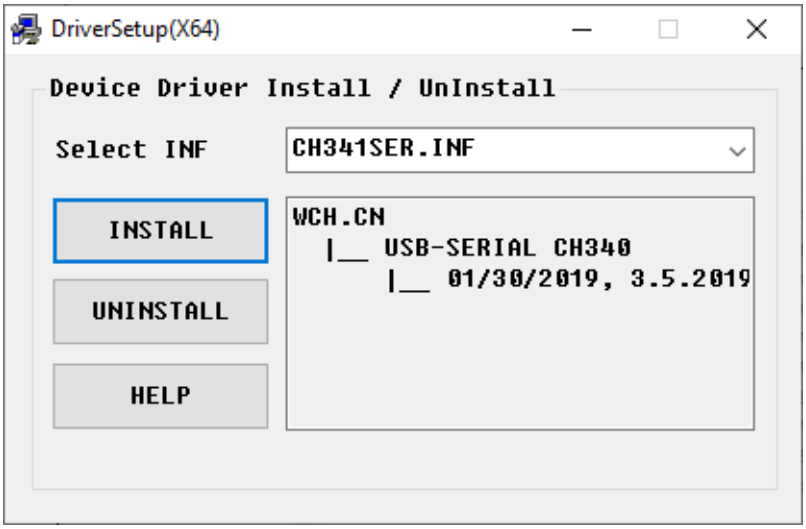
If you would not like to download the installation package, you can open **"Freenove_ESP32_WROVER_Board/CH340"**, we have prepared the installation package.



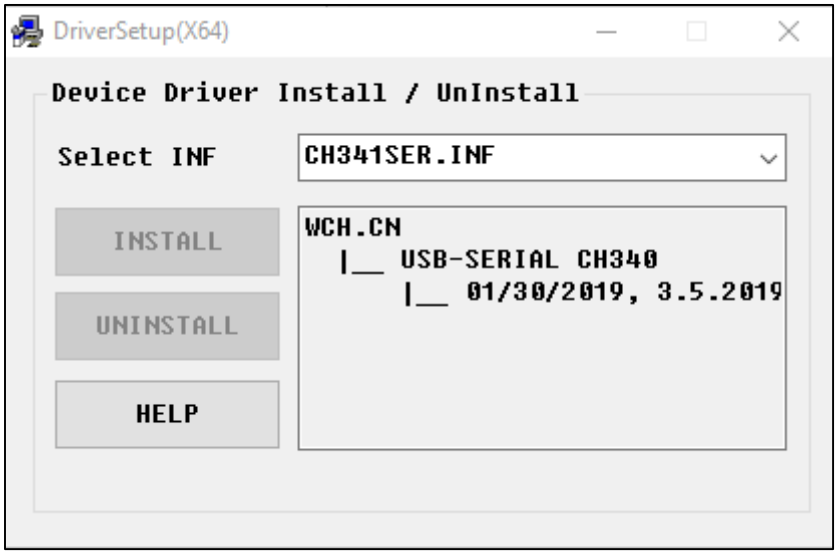
2. Open the folder "**Freenove_ESP32_WROVER_Board/CH340/Windows/**"



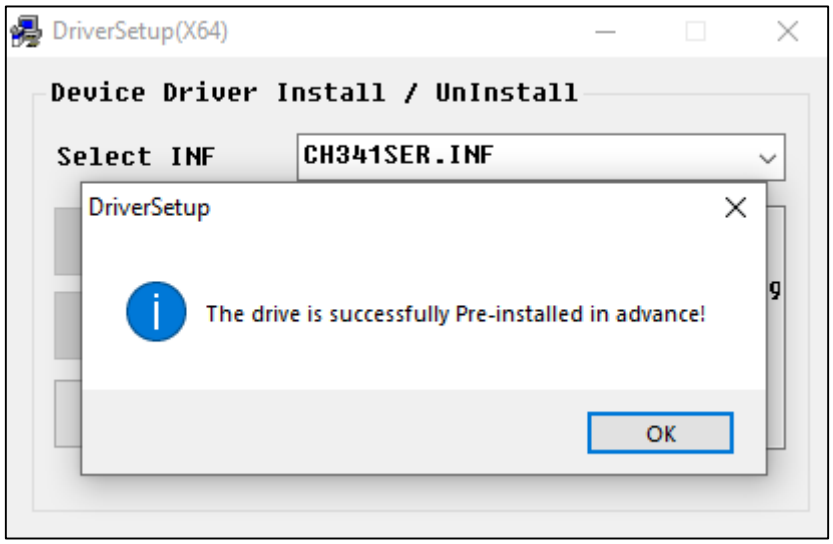
3. Double click "**CH341SER.EXE**".



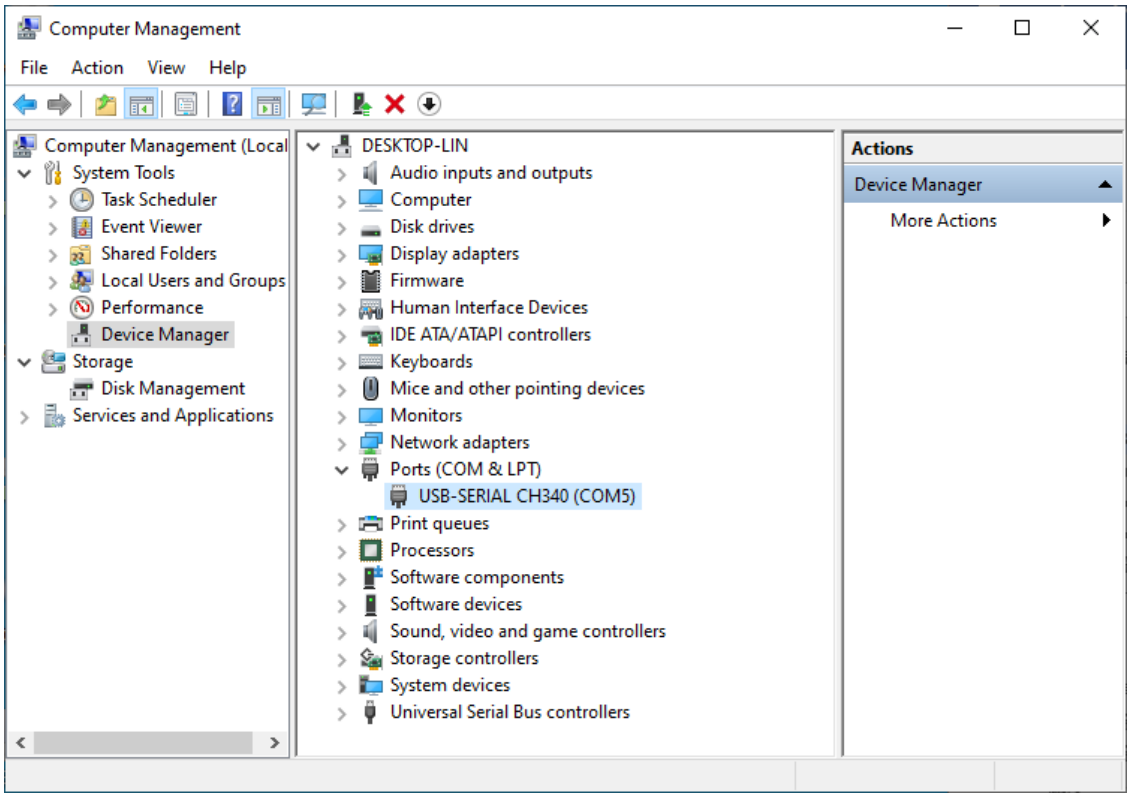
4. Click “INSTALL” and wait for the installation to complete.



5. Install successfully. Close all interfaces.



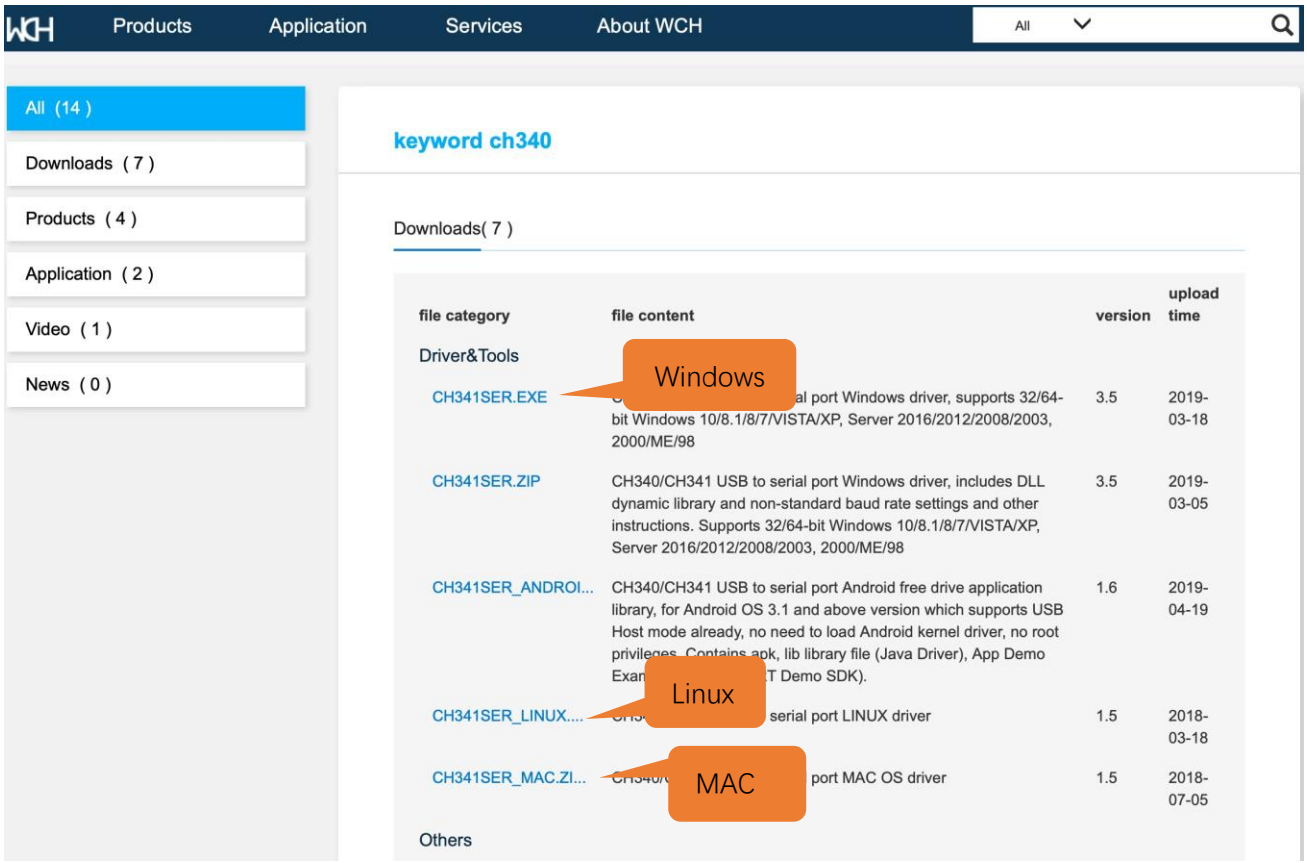
6. When ESP32 is connected to computer, select “This PC”, right-click to select “Manage” and click “Device Manager” in the newly pop-up dialog box, and you can see the following interface.



7. So far, CH340 has been installed successfully. Close all dialog boxes.

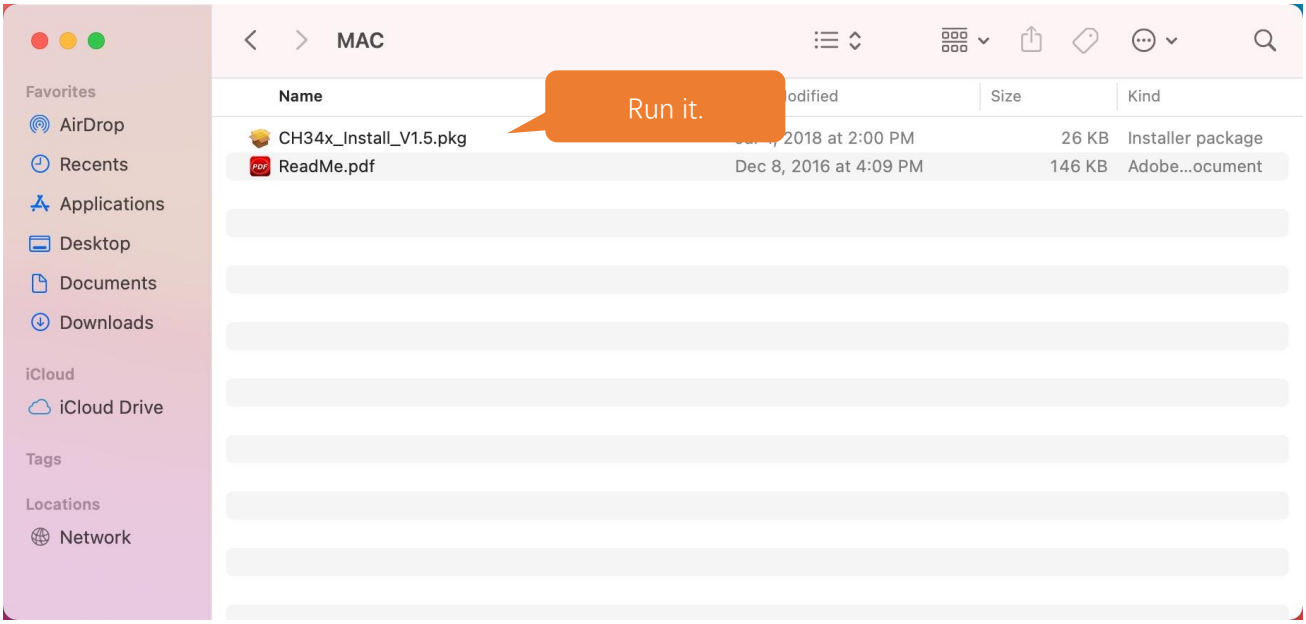
MAC

First, download CH340 driver, click http://www.wch-ic.com/search?q=CH340&t=downloads to download the appropriate one based on your operating system.

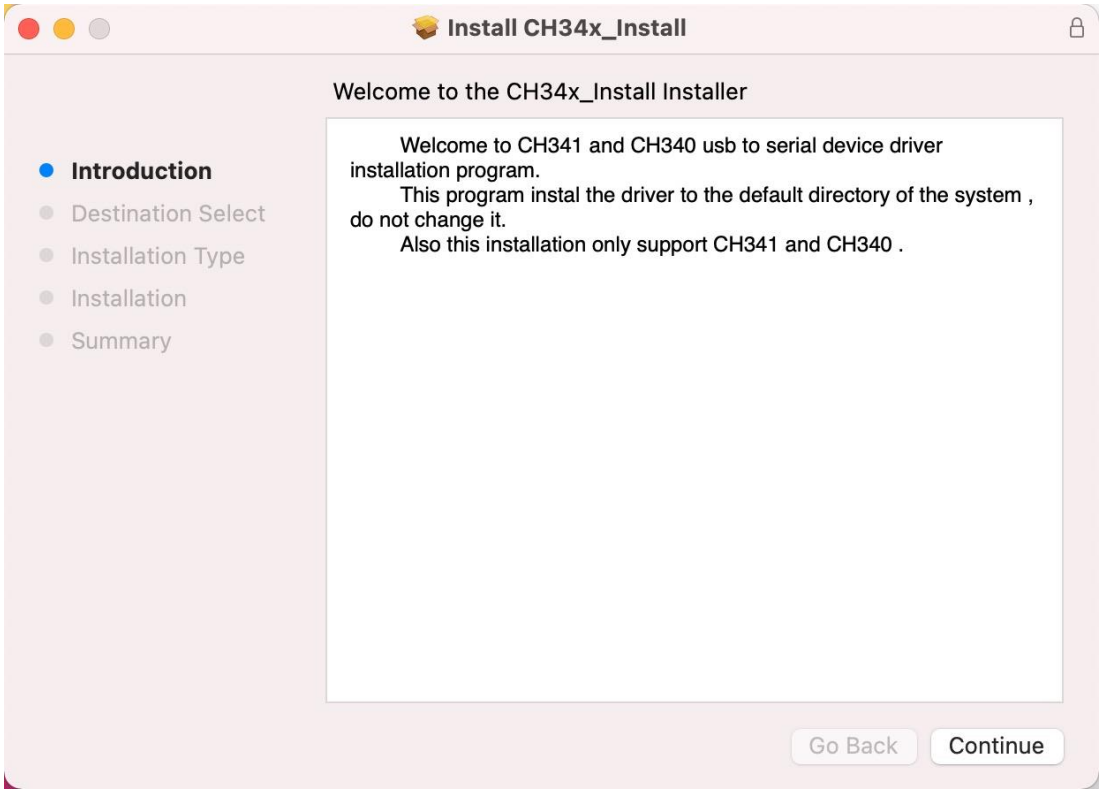


If you would not like to download the installation package, you can open **"Freenove_ESP32_WROVER_Board/CH340"**, we have prepared the installation package.

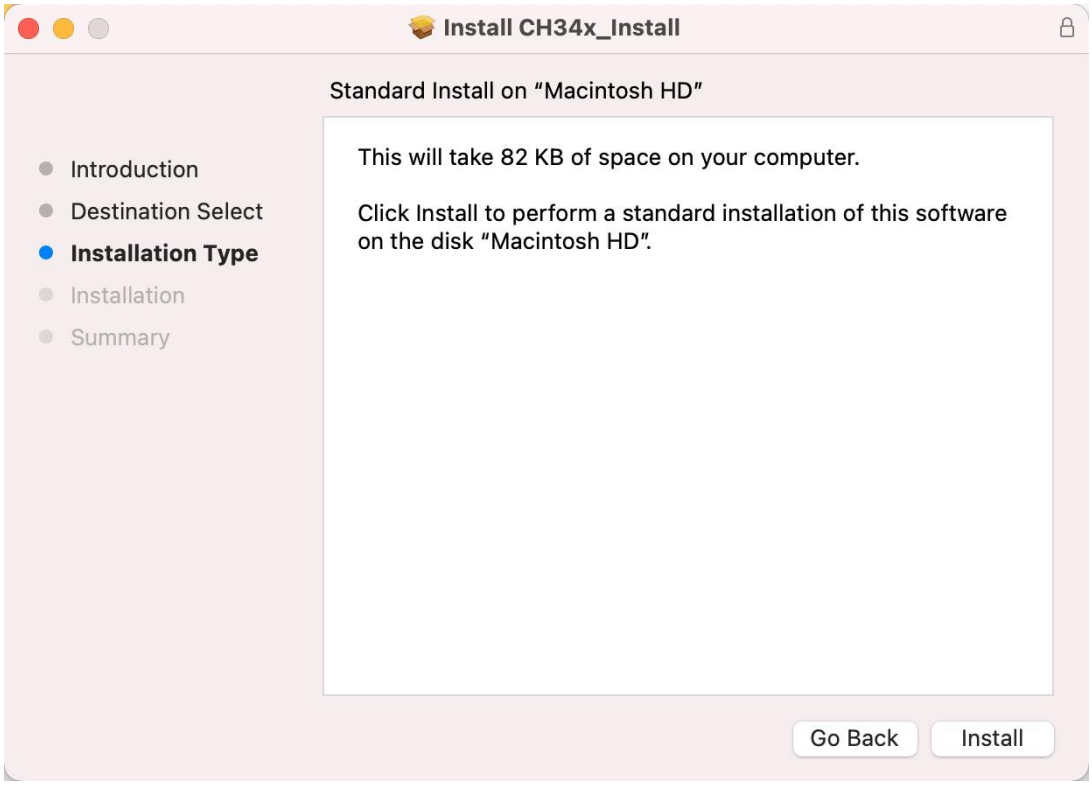
Second, open the folder "**Freenove_ESP32_WROVER_Board/CH340/MAC/**"



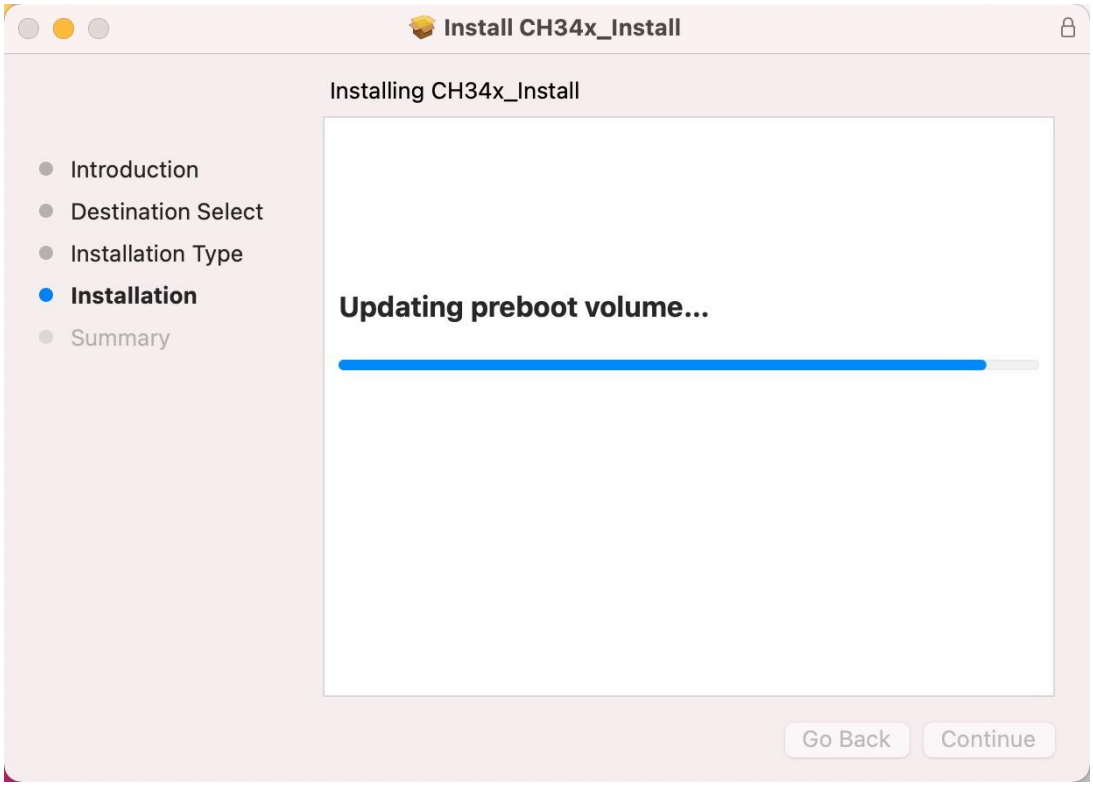
Third, click Continue.



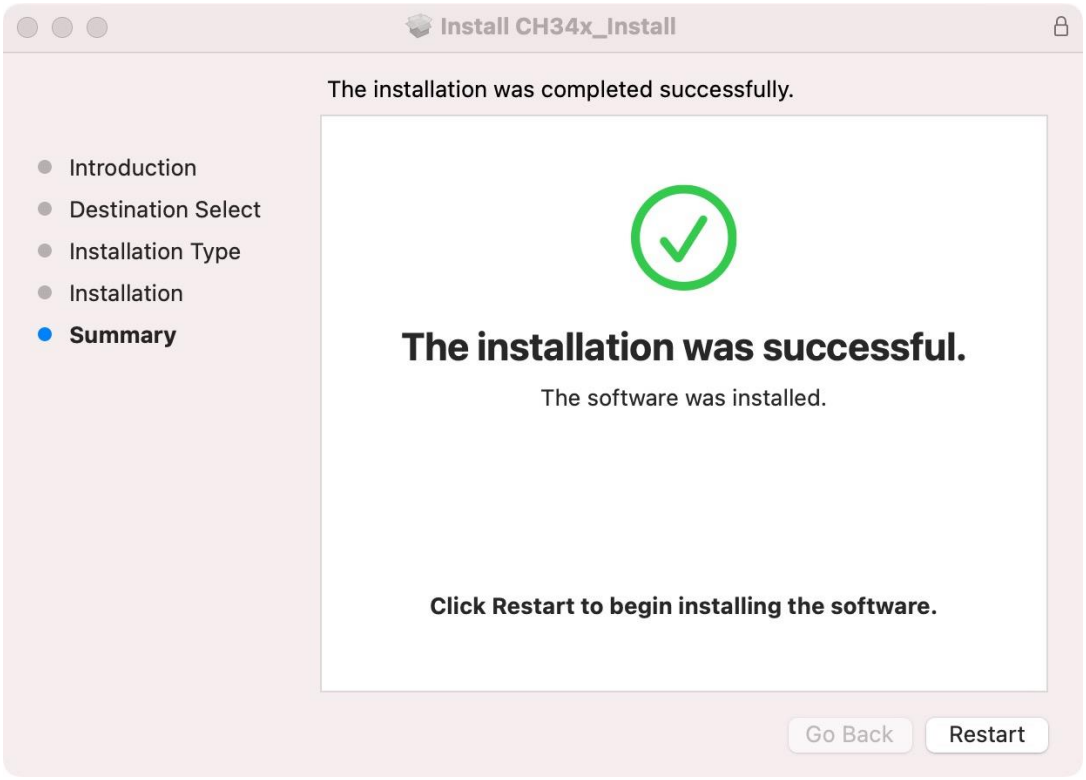
Fourth, click Install.



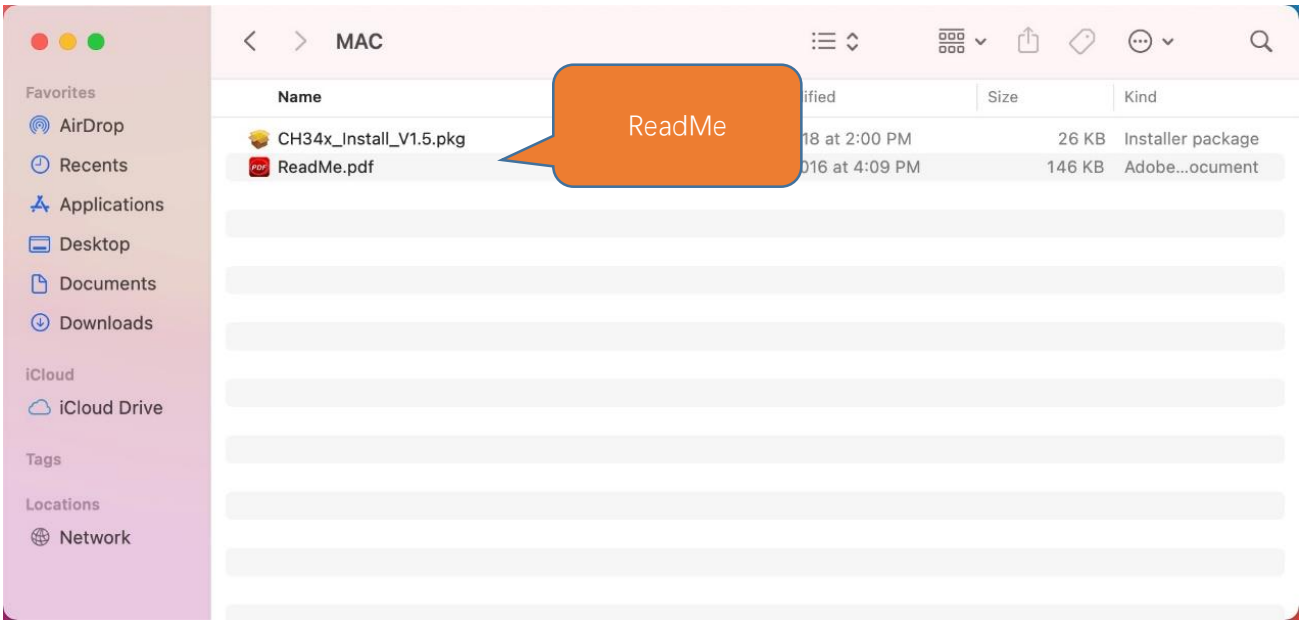
Then, waiting Finsh.



Finally, restart your PC.

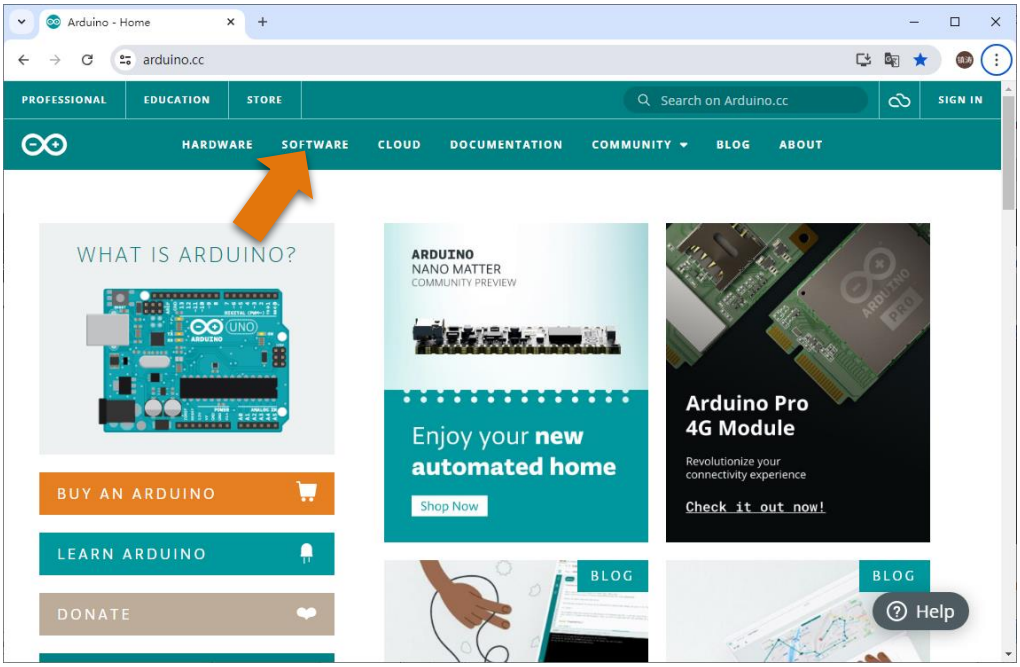


If you still haven't installed the CH340 by following the steps above, you can view readme.pdf to install it.

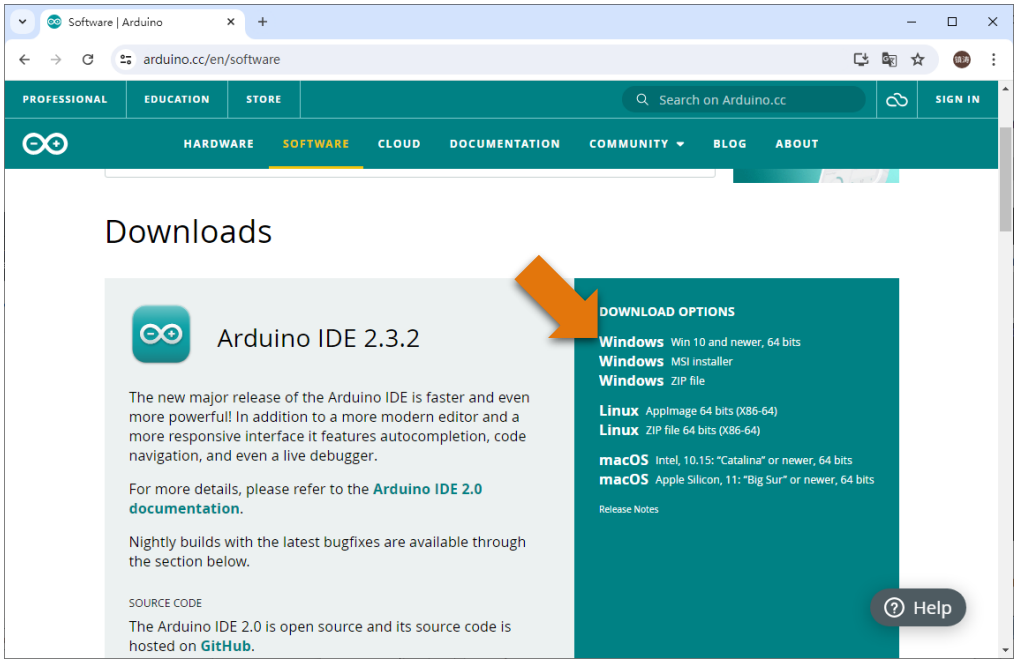


Programming Software

Arduino Software (IDE) is used to write and upload the code for Arduino Board.
First, install Arduino Software (IDE): visit https://www.arduino.cc, click "Download" to enter the download page.



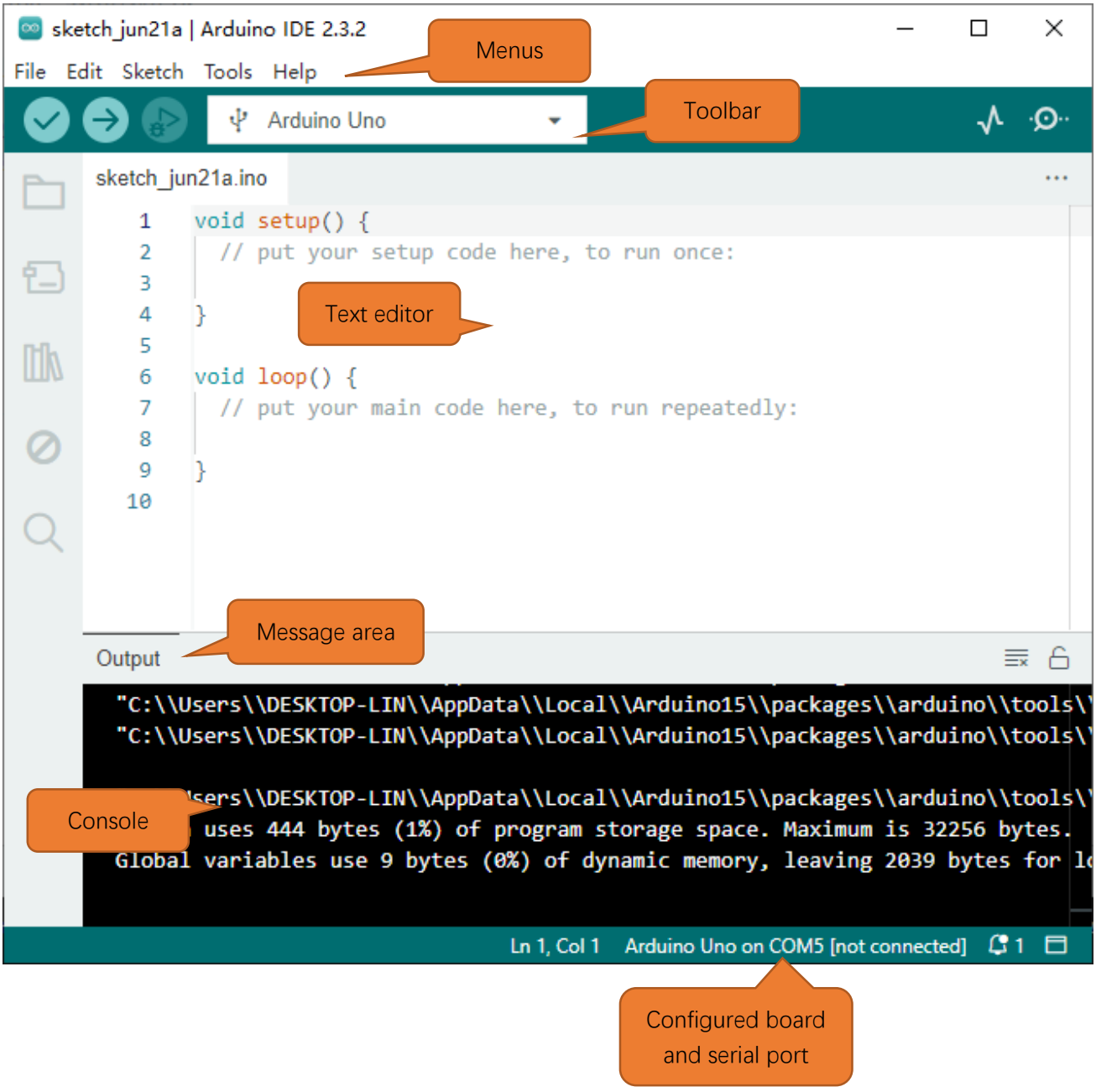
Select and download corresponding installer according to your operating system. If you are a windows user, please select the "Windows Installer" to download to install the driver correctly.



After the download completes, run the installer. For Windows users, there may pop up an installation dialog box of driver during the installation process. When it popes up, please allow the installation.

After installation is complete, an Arduino Software shortcut will be generated in the desktop. Run the Arduino Software.

The interface of Arduino Software is as follows:



Programs written with Arduino Software (IDE) are called **sketches**. These sketches are written in the text editor and saved with the file extension**.ino**. The editor has features for cutting/pasting and searching/replacing text. The message area gives feedback while saving and exporting and also displays errors. The console displays text output by the Arduino Software (IDE), including complete error messages and other information. The bottom right-hand corner of the window displays the configured board and serial port. The toolbar buttons allow you to verify and upload programs, create, open, and save sketches, and open the serial monitor.

Verify
Check your code for compile errors .

Upload
Compile your code and upload them to the configured board.

Debug
With a debugger or USB port, debug the code.

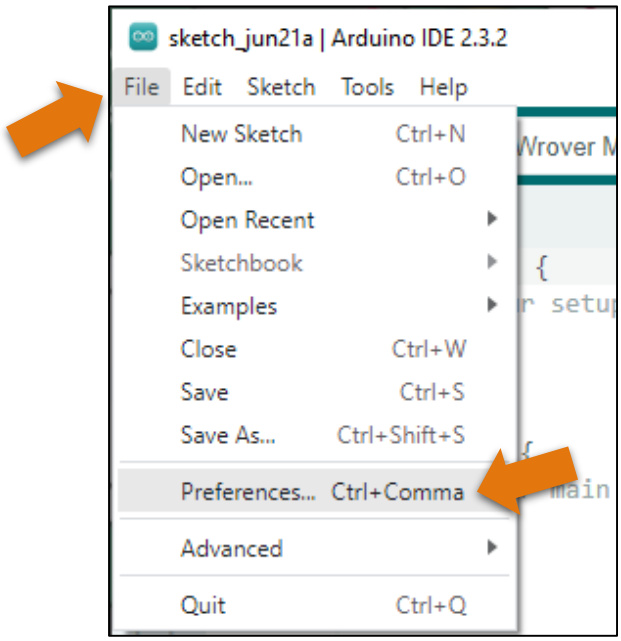
Serial Plotter
Open the serial plotter.

Serial Monitor
Open the serial monitor.

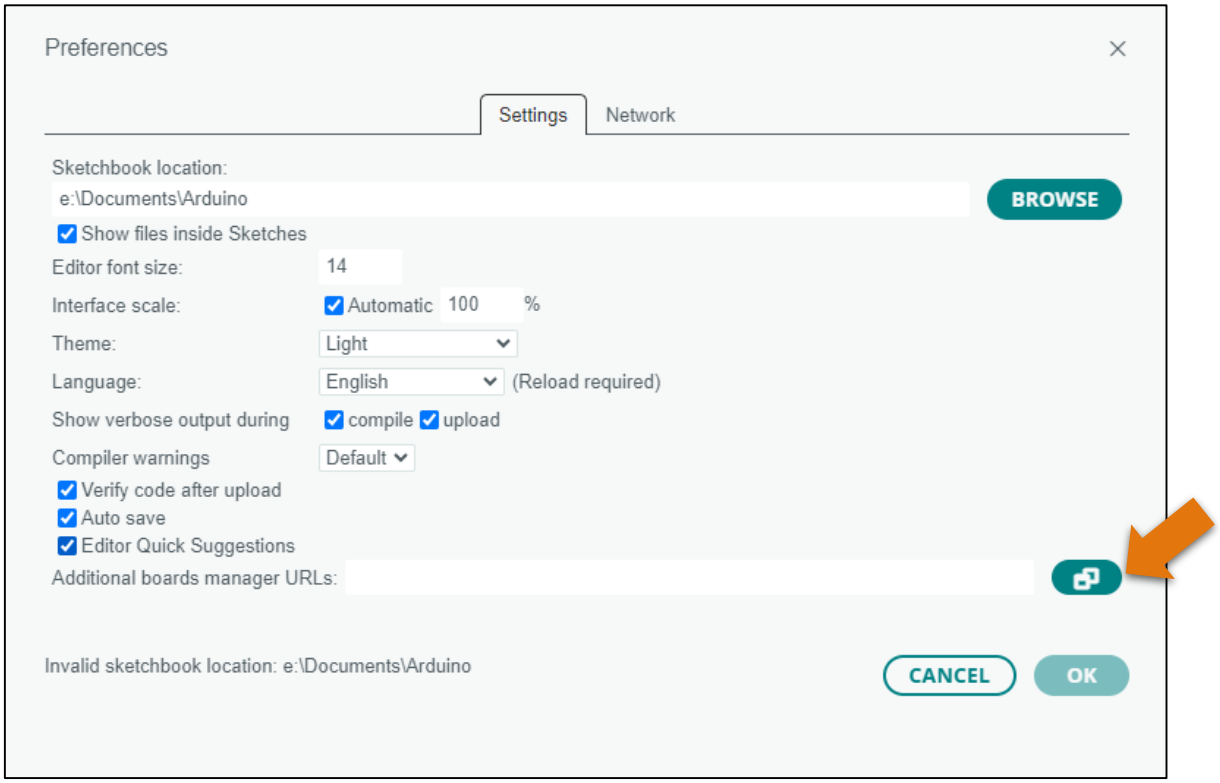
Additional commands are found within the five menus: File, Edit, Sketch, Tools, Help. The menus are context sensitive, which means only those items relevant to the work currently being carried out are available.

Environment Configuration

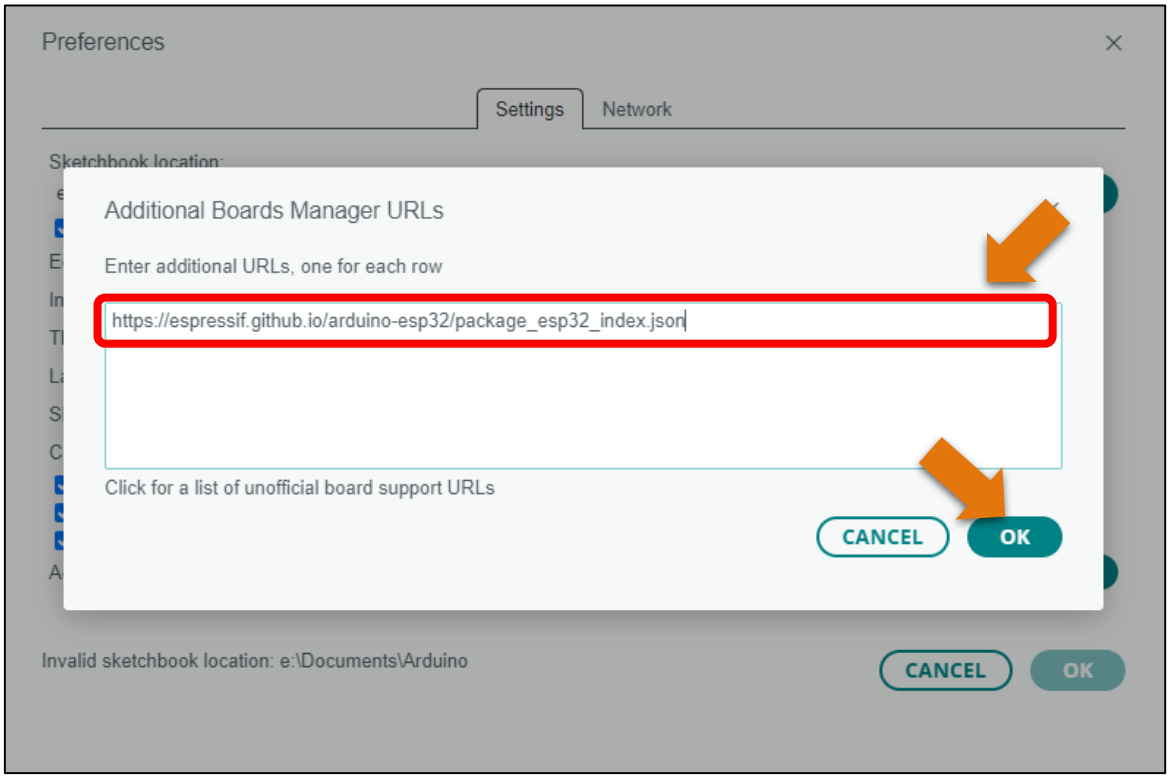
First, open the software platform arduino, and then click File in Menus and select Preferences.



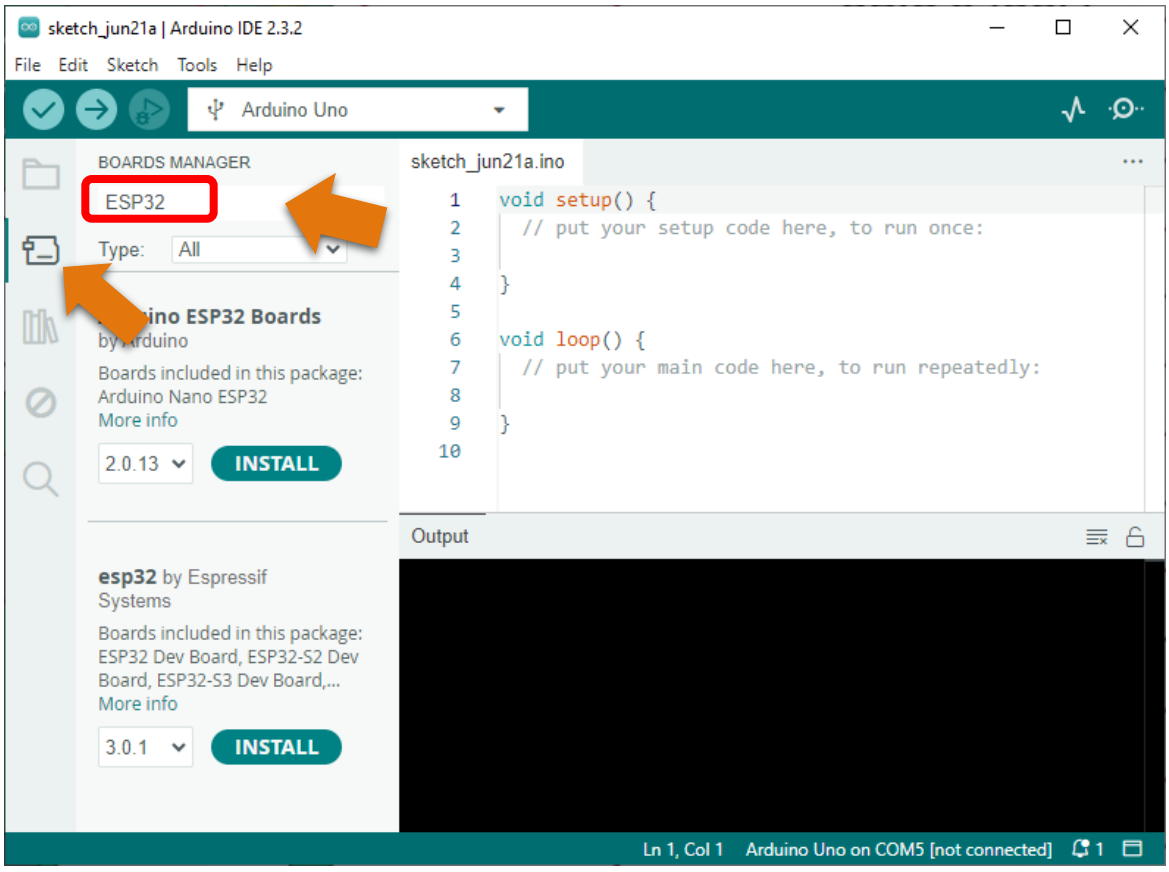
Second, click on the symbol behind "Additional Boards Manager URLs"



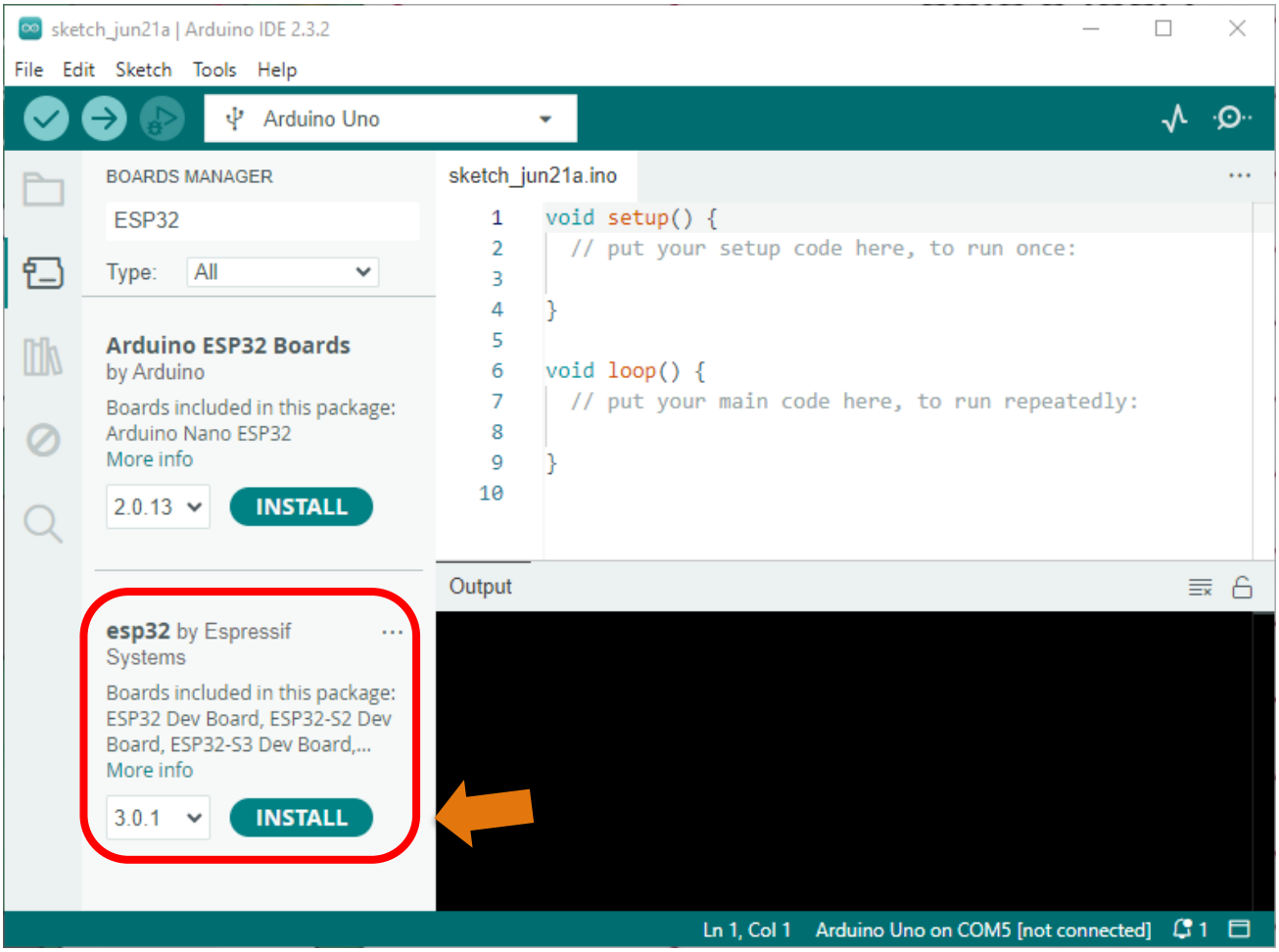
Third, fill in https://espressif.github.io/arduino-esp32/package_esp32_index.json in the new window, click OK, and click OK on the Preferences window again.



Fourth, click "BOARDS MANAGER" on the left and type "ESP32" in the search box.

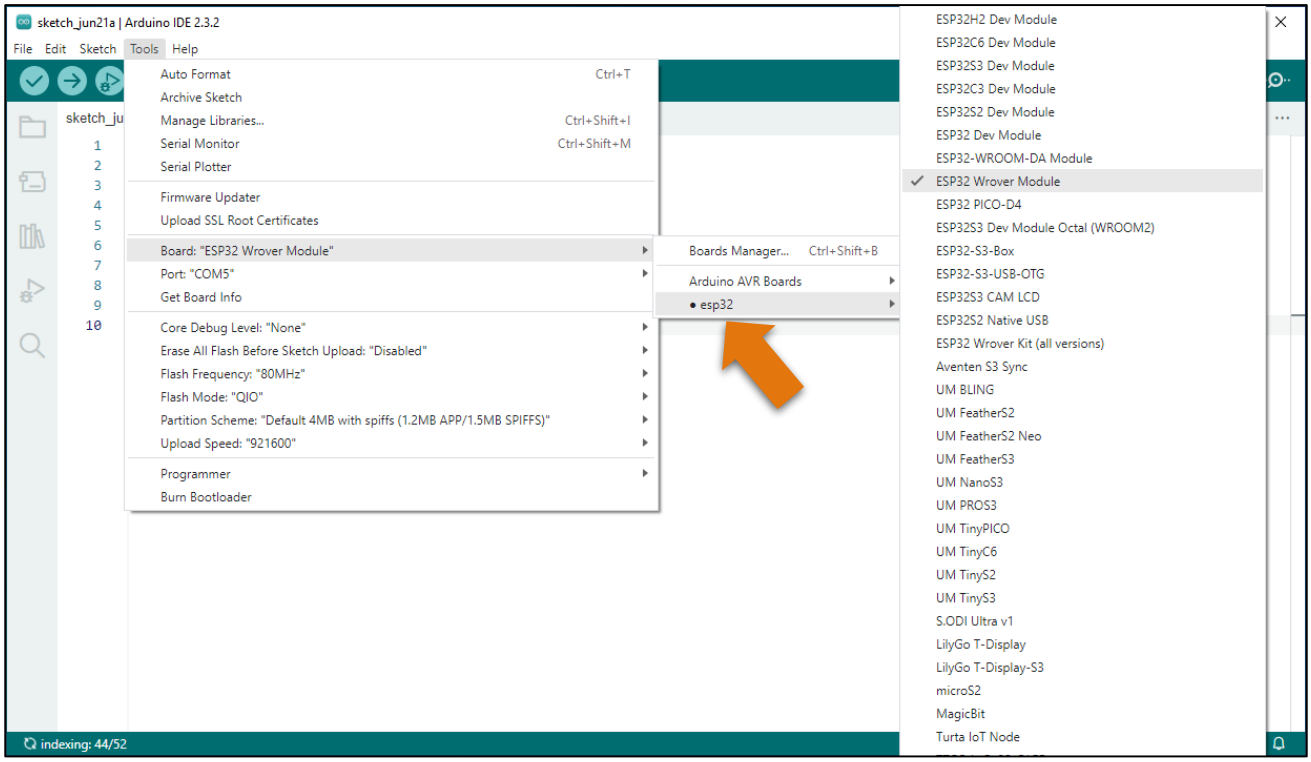


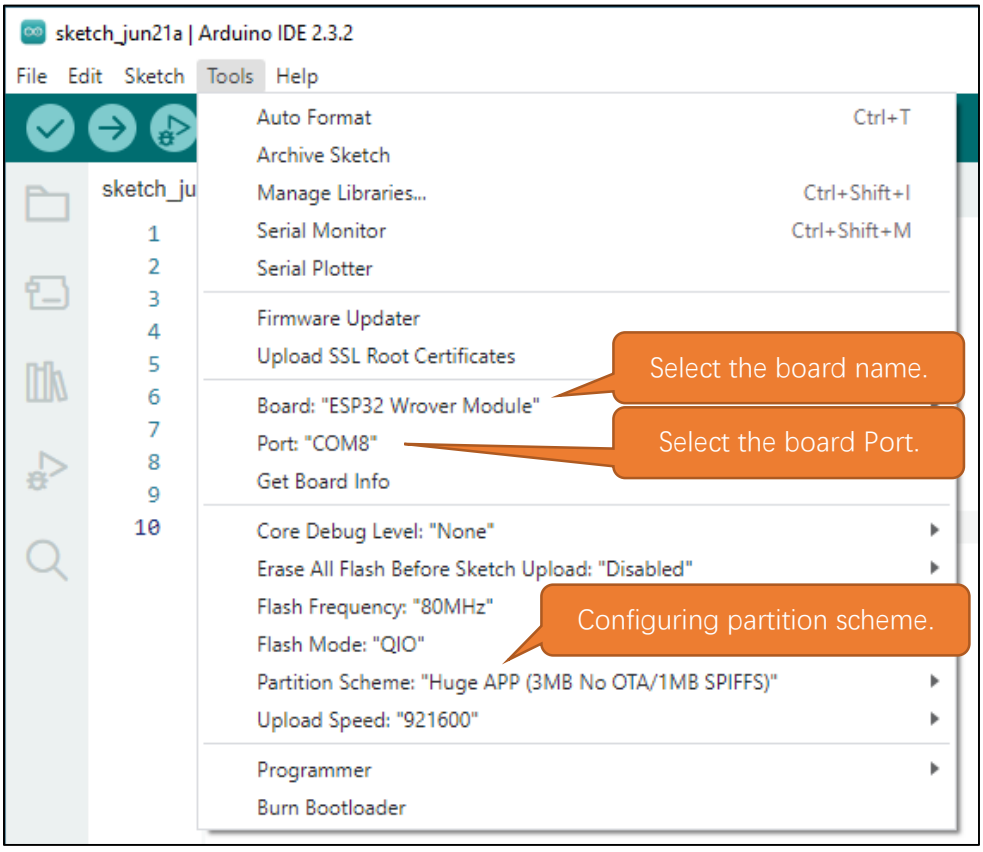
Fifth, select Espressif Systems' ESP32 and select version 3.0.x. Click "INSTALL" to install esp32.



Note: it takes a while to install the ESP32, make sure your network is stable.

When finishing installation, click Tools in the Menus again and select Board: "Arduino Uno", and then you can see information of **ESP32 Wrover Module**. Click " **ESP32 Wrover Module**" so that the ESP32 programming development environment is configured.





Notes for GPIO

Strapping Pin

There are five Strapping pins for ESP32：MTDI、GPIO0、GPIO2、MTDO、GPIO5。
With the release of the chip's system reset (power-on reset, RTC watchdog reset, undervoltage reset), the strapping pins sample the level and store it in the latch as "0" or "1" ", and keep it until the chip is powered off or turned off.
Each Strapping pin is connecting to internal pull-up/pull-down. Connecting to high-impedance external circuit or without an external connection, a strapping pin's default value of input level will be determined by internal weak pull-up/pull-down. To change the value of the Strapping, users can apply an external pull-down/pull-up resistor, or use the GPIO of the host MCU to control the level of the strapping pin when the ESP32's power on reset is released.
When releasing the reset, the strapping pin has the same function as a normal pin.
The followings are default configurations of these five strapping pins at power-on and their functions under the corresponding configuration.

Voltage of Internal LDO (VDD_SDIO)			
Pin	Default	3.3 V	1.8 V
MTDI	Pull-down	0	1
Booting Mode			
Pin	Default	SPI Boot	Download Boot
GPIO0	Pull-up	1	0
GPIO2	Pull-down	Don't-care	0
Enabling/Disabling Debugging Log Print over U0TXD During Booting			
Pin	Default	U0TXD Active	U0TXD Silent
MTDO	Pull-up	1	0

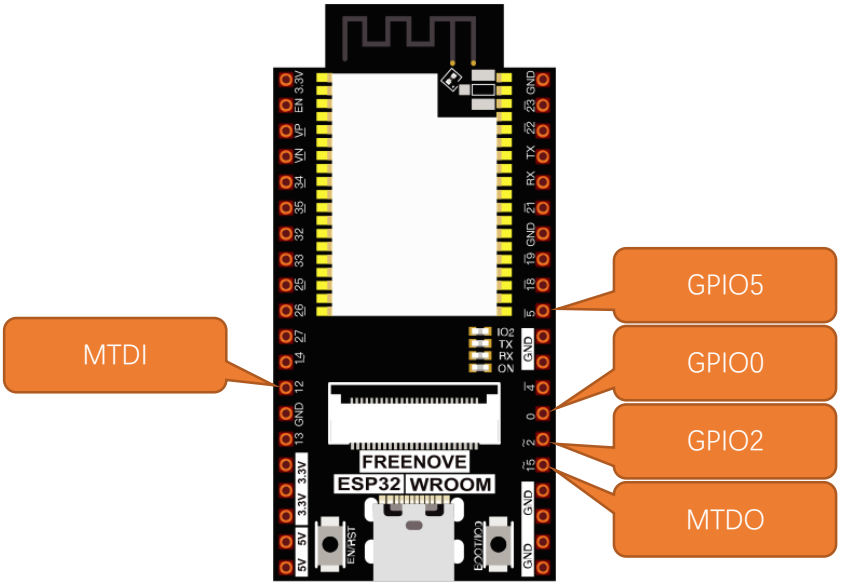
Timing of SDIO Slave					
Pin	Default	Falling-edge Sampling Falling-edge Output	Falling-edge Sampling Rising-edge Output	Rising-edge Sampling Falling-edge Output	Rising-edge Sampling Rising-edge Output
MTDO	Pull-up	0	0	1	1
GPIO5	Pull-up	0	1	0	1

Note:

- Firmware can configure register bits to change the settings of "Voltage of Internal LDO (VDD_SDIO)" and "Timing of SDIO Slave" after booting.
- The MTDI is internally pulled high in the module, as the flash and SRAM in ESP32-WROVER only support a power voltage of 1.8 V (output by VDD_SDIO).

If you have any questions about the information of GPIO, you can click here to go back to ESP32-WROVER to view specific information about GPIO.
If you have any difficulties or questions with this tutorial or toolkit, feel free to ask for our quick and free technical support through support@freenove.com at any time.
or check: https://www.espressif.com/sites/default/files/documentation/esp32-wrover_datasheet_en.pdf

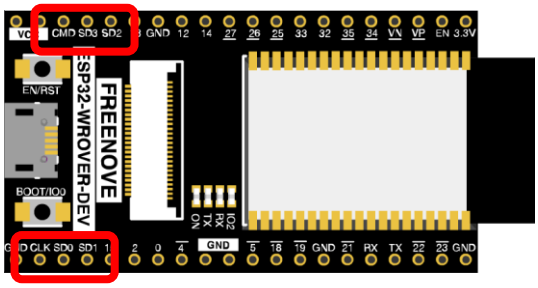
Strapping Pin



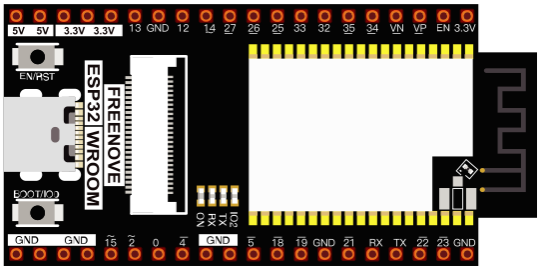
Flash Pin

GPIO6-11 has been used to connect the integrated SPI flash on the module, and is used when GPIO 0 is power on and at high level. Flash is related to the operation of the whole chip, so the external pin GPIO6-11 cannot be used as an experimental pin for external circuits, otherwise it may cause errors in the operation of the program.

In older versions, the flash pin looks like the image below.



In the new release, we no longer introduce GPIO6-11.



GPIO16-17 has been used to connect the integrated PSRAM on the module.

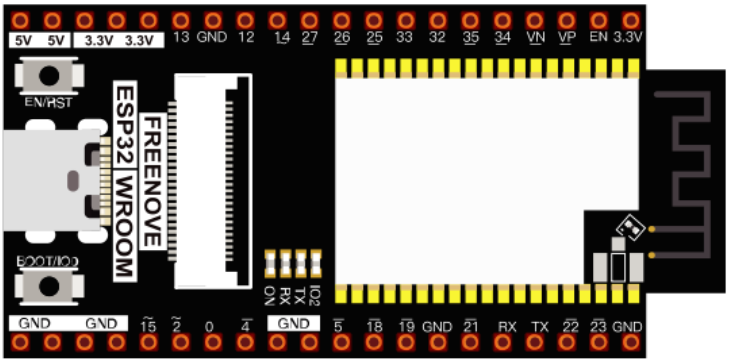
Because of external pull-up, MTDI pin is not suggested to be used as a touch sensor. For details, please refer to Peripheral Interface and Sensor chapter in "ESP32 Data_Sheet".

For more relevant information, please check:

https://www.espressif.com/sites/default/files/documentation/esp32-wrover_datasheet_en.pdf.

Cam Pin

When using the camera of our ESP32-WROVER, please check the pins of it.
Pins with underlined numbers are used by the camera function, if you want to use other functions besides it, please avoid using them.



CAM_Pin	GPIO_pin
I2C_SDA	GPIO26
I2C_SCL	GPIO27
CSI_VYSNC	GPIO25
CSI_HREF	GPIO23
CSI_Y9	GPIO35
XCLK	GPIO21
CSI_Y8	GPIO34
CSI_Y7	GPIO39
CSI_PCLK	GPIO22
CSI_Y6	GPIO36
CSI_Y2	GPIO4
CSI_Y5	GPIO19
CSI_Y3	GPIO5
CSI_Y4	GPIO18

If you have any questions about the information of GPIO, you can click here to go back to ESP32-WROVER to view specific information about GPIO.
or check: https://www.espressif.com/sites/default/files/documentation/esp32-wrover_datasheet_en.pdf.

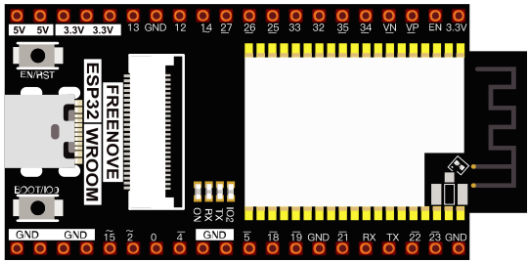
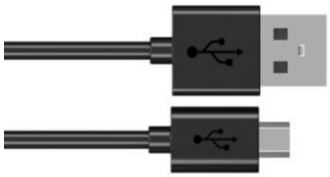
Chapter 1 LED

This chapter is the Start Point in the journey to build and explore ESP32 electronic projects. We will start with simple “Blink” project.

Project 1.1 Blink

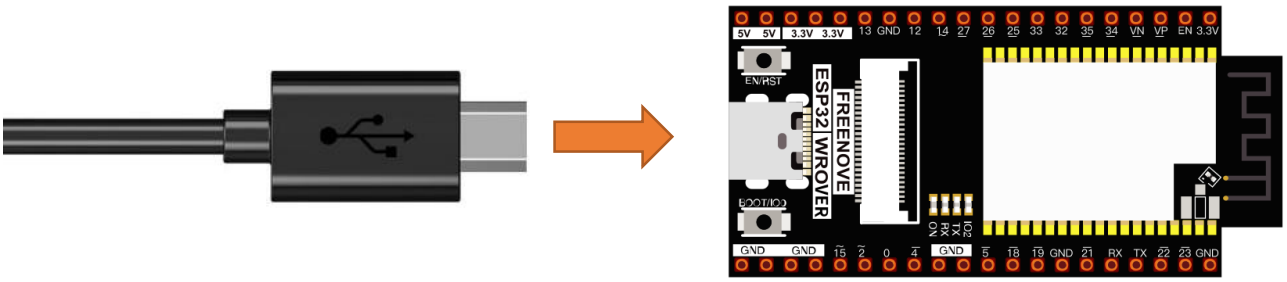
In this project, we will use ESP32 to control blinking a common LED.

Component List

ESP32-WROVER x1	USB cable
	

Power

ESP32-WROVER needs 5v power supply. In this tutorial, we need connect ESP32-WROVER to computer via USB cable to power it and program it. We can also use other 5v power source to power it.



In the following projects, we only use USB cable to power ESP32-WROVER by default.
In the whole tutorial, we don’t use T extension to power ESP32-WROVER. So 5V and 3.3V (includeing EXT 3.3V) on the extension board are provided by ESP32-WROVER.
We can also use DC jack of extension board to power ESP32-WROVER.In this way, 5v and EXT 3.3v on extension board are provided by external power resource.

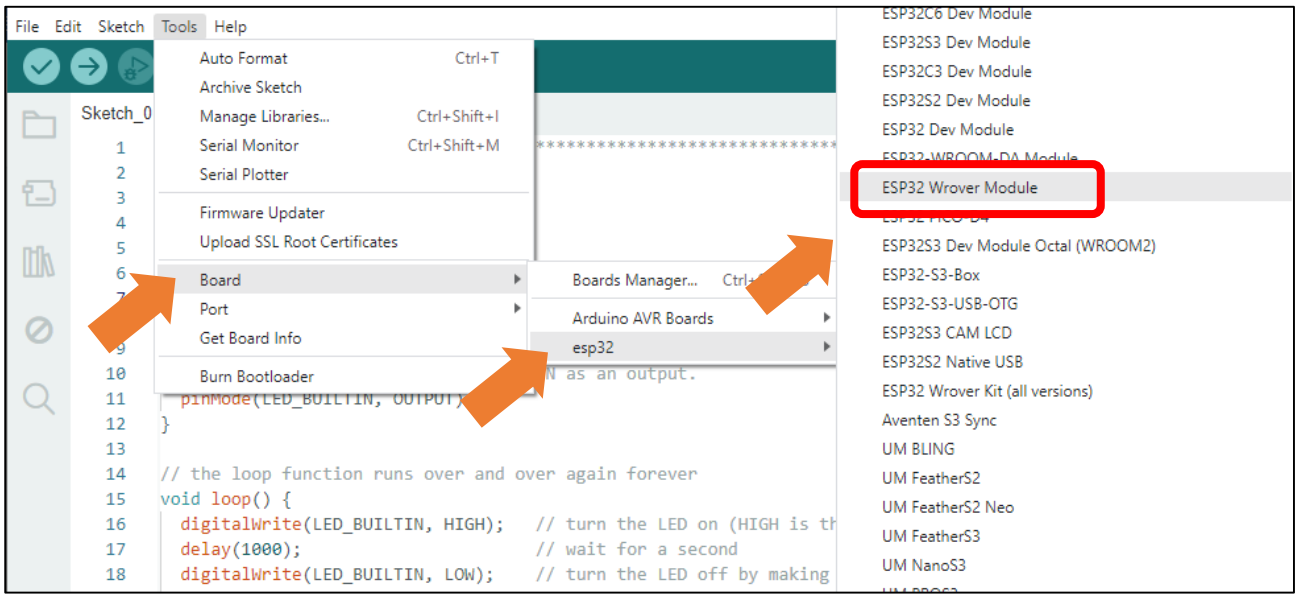
Sketch

According to the circuit, when the GPIO2 of ESP32-WROVER output level is low, the LED turns ON. Conversely, when the GPIO2 ESP32-WROVER output level is high, the LED turns OFF. Therefore, we can let GPIO2 circularly output high and low level to make the LED blink.

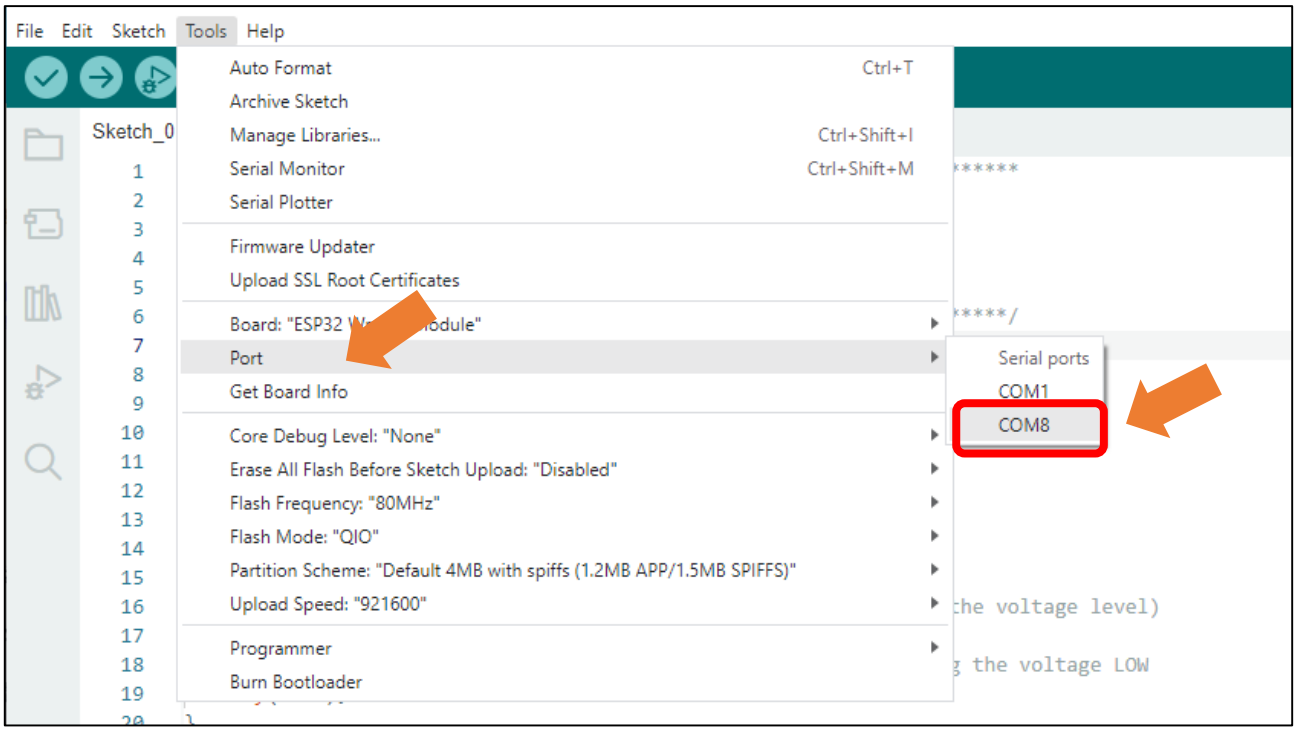
Upload the following Sketch:

Freenove_ESP32_WROVER_Board\Sketches\Sketch_01.1_Blink.

Before uploading the code, click "**Tools**", "**Board**" and select "**ESP32 Wrover Module**".

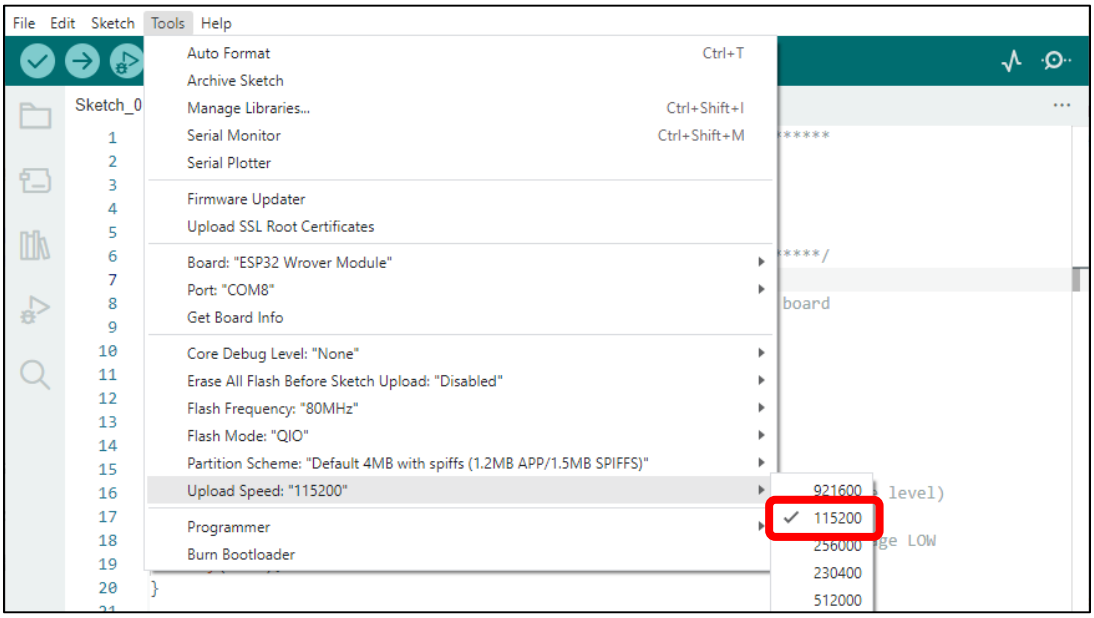


Select the serial port.

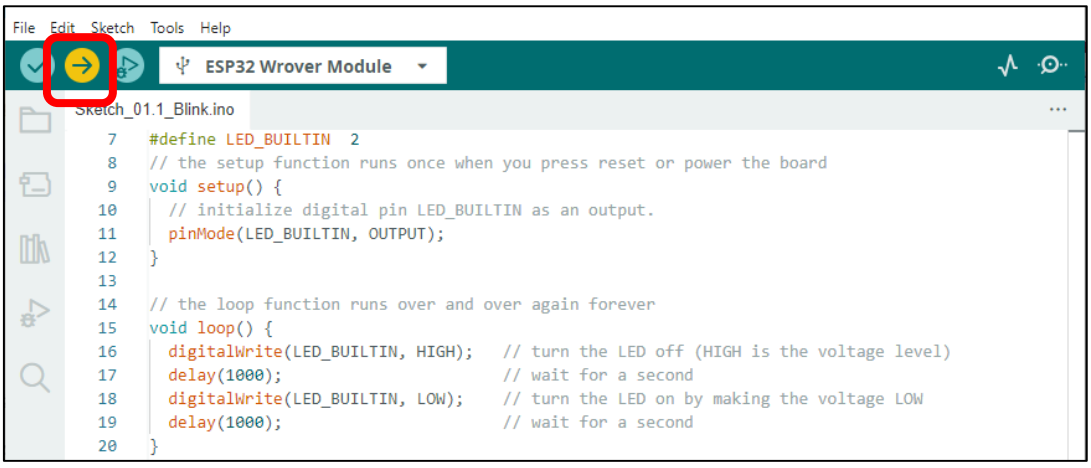


Note: For macOS users, if the uploading fails, please set the baud rate to 115200 before clicking

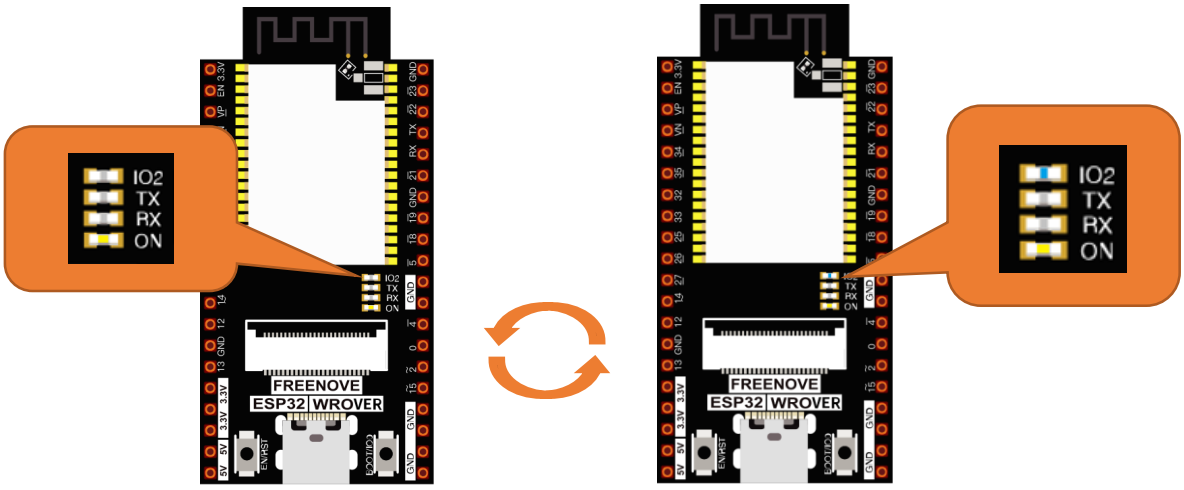
"Upload Speed".



Sketch_01.1_Blink



Click "Upload", Download the code to ESP32-WROVER and your LED in the circuit starts Blink.

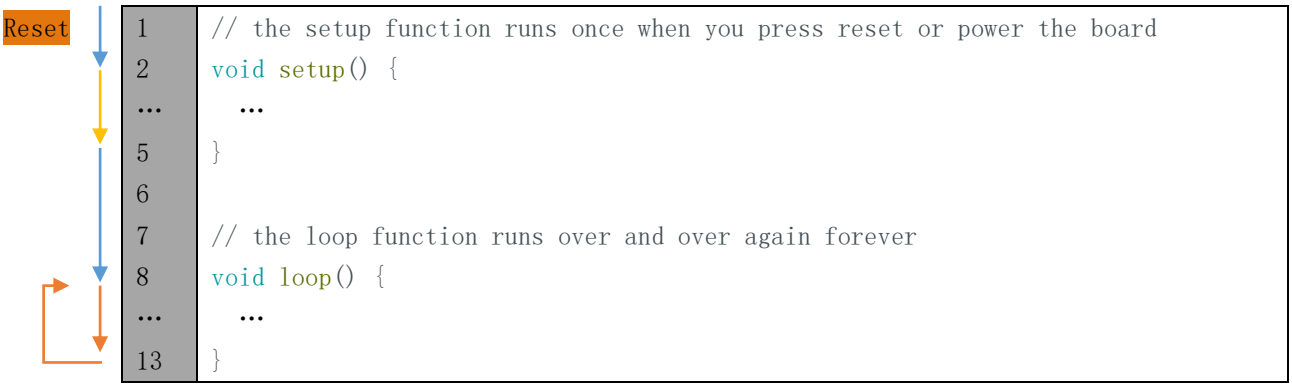


If you have any concerns, please contact us via: support@freenove.com

The following is the program code:

```
#define PIN_LED 2
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(PIN_LED, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(PIN_LED, HIGH);   // turn the LED off (HIGH is the voltage level)
  delay(1000);                   // wait for a second
  digitalWrite(PIN_LED, LOW);    // turn the LED on by making the voltage LOW
  delay(1000);                   // wait for a second
}
```

The Arduino IDE code usually contains two basic functions: void setup() and void loop().
After the board is reset, the setup() function will be executed firstly, and then the loop() function.
setup() function is generally used to write code to initialize the hardware. And loop() function is used to write code to achieve certain functions. loop() function is executed repeatedly. When the execution reaches the end of loop(), it will jump to the beginning of loop() to run again.



Reset

Reset operation will lead the code to be executed from the beginning. Switching on the power, finishing uploading the code and pressing the reset button will trigger reset operation.

In the circuit, ESP32-WROVER's GPIO2 is connected to the LED, so the LED pin is defined as 2.

```
#define PIN_LED 2
```

This means that after this line of code, all PIN_LED will be treated as 2.
In the setup () function, first, we set the PIN_LED as output mode, which can make the port output high level or low level.

```
  // initialize digital pin PIN_LED as an output.
  pinMode(PIN_LED, OUTPUT);
```

Then, in the loop () function, set the PIN_LED to output high level to make LED light off.

```
  digitalWrite(PIN_LED, HIGH);  // turn the LED off (HIGH is the voltage level)
```

Wait for 1000ms, that is 1s. Delay () function is used to make control board wait for a moment before executing the next statement. The parameter indicates the number of milliseconds to wait for.

```
11    delay(1000);                    // wait for a second
```

Then set the PIN_LED to output low level, and LED light up. One second later, the execution of loop () function will be completed.

```
12    digitalWrite(PIN_LED, LOW);    // turn the LED on by making the voltage LOW
13    delay(1000);                    // wait for a second
```

The loop() function is constantly being executed, so LED will keep blinking.

Reference

void pinMode(int pin, int mode);

Configures the specified pin to behave either as an input or an output.

Parameters

pin: the pin number to set the mode of.

mode: INPUT, OUTPUT, INPUT_PULLDOWM, or INPUT_PULLUP.

void digitalWrite (int pin, int value);

Writes the value HIGH or LOW (1 or 0) to the given pin which must have been previously set as an output.

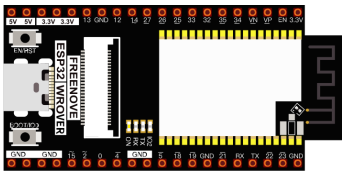
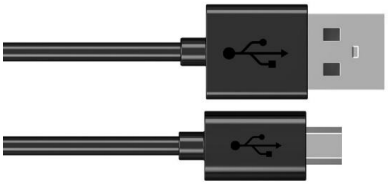
For more related functions, please refer to https://www.arduino.cc/reference/en/

Chapter 2 Bluetooth

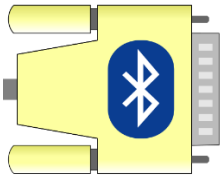
This chapter mainly introduces how to make simple data transmission through Bluetooth of ESP32-WROVER and mobile phones.
Project 2.1 is classic Bluetooth and Project 2.2 is low power Bluetooth.If you are an iPhone user, please start with Project 2.2.

Project 2.1 Bluetooth Passthrough

Component List

ESP32-WROVER x1 	Micro USB Wire x1 

In this tutorial we need to use a Bluetooth APP called Serial Bluetooth Terminal to assist in the experiment. If you've not installed it yet, please do so by clicking: https://www.appsapk.com/serial-bluetooth-terminal/ The following is its logo.



Component knowledge

ESP32's integrated Bluetooth function Bluetooth is a short-distance communication system, which can be divided into two types, namely Bluetooth Low Energy(BLE) and Classic Bluetooth. There are two modes for simple data transmission: master mode and slave mode.

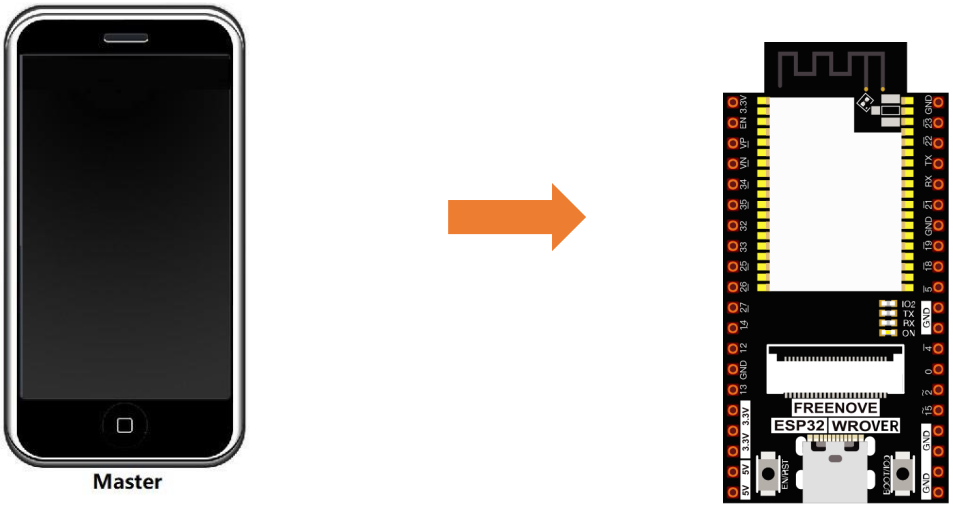
Master mode

In this mode, works are done in the master device and it can connect with a slave device. And we can search and select slave devices nearby to connect with. When a device initiates connection request in master mode, it requires information of the other Bluetooth devices including their address and pairing passkey. After finishing pairing, it can connect with them directly.

Slave mode

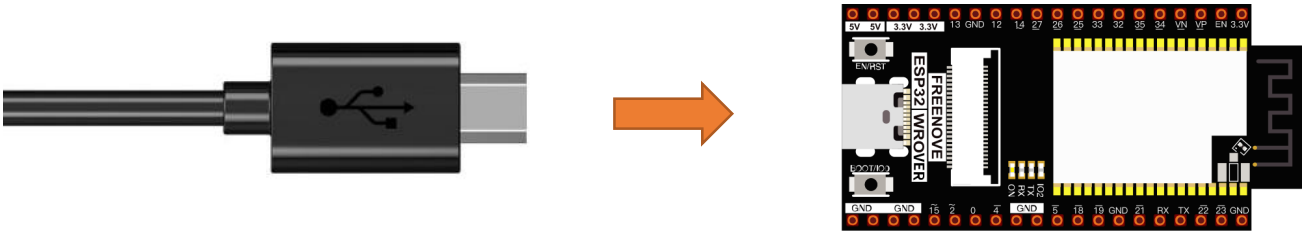
The Bluetooth module in slave mode can only accept connection request from a host computer, but cannot initiate a connection request. After connecting with a host device, it can send data to or receive from the host device.

Bluetooth devices can make data interaction with each other, as one is in master mode and the other in slave mode. When they are making data interaction, the Bluetooth device in master mode searches and selects devices nearby to connect to. When establishing connection, they can exchange data. When mobile phones exchange data with ESP32, they are usually in master mode and ESP32 in slave mode.



Circuit

Connect Freenove ESP32 to the computer using the USB cable.



Sketch

Sketch_02.1_SerialToSerialBT

File Edit Sketch Tools Help

ESP32 Wrover Module

Sketch_27.1_SerialToSerialBT.ino

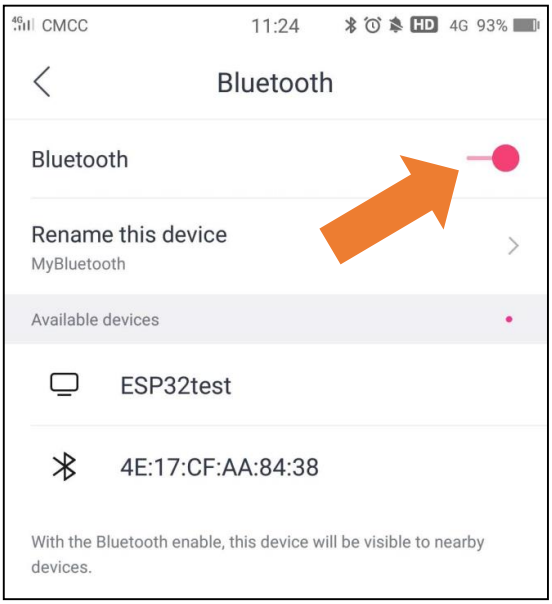
```
#include "BluetoothSerial.h"

BluetoothSerial SerialBT;
String buffer;
void setup() {
  Serial.begin(115200);
  SerialBT.begin("ESP32test"); //Bluetooth device name
  Serial.println("\nThe device started, now you can pair it with bluetooth!");
}

void loop() {
  if (Serial.available()) {
    SerialBT.write(Serial.read());
  }
  if (SerialBT.available()) {
    Serial.write(SerialBT.read());
  }
  delay(20);
}
```

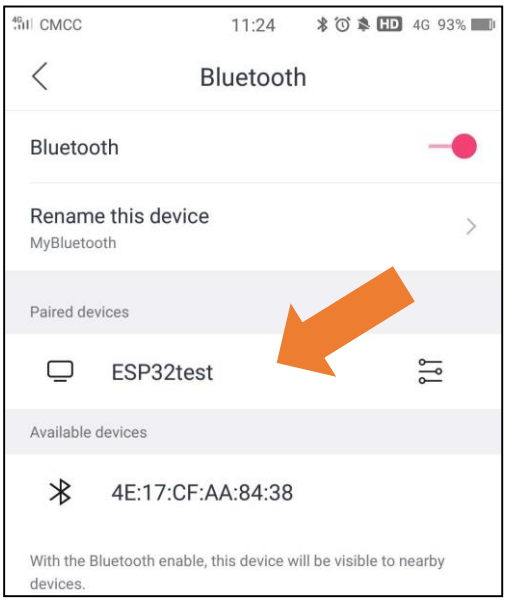
Compile and upload the code to the ESP32-WROVER, open the serial monitor, and set the baud rate to 115200. When you see the serial printing out the character string as below, it indicates that the Bluetooth of ESP32 is ready and waiting to connect with the mobile phone.

```
11:07:06.801 -> rst:0x1 (POWERON_RESET),boot:0x1b (SPI_FAST_FLASH_BOOT)
11:07:06.842 -> configsip: 0, SPIWP:0xee
11:07:06.842 -> clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
11:07:06.842 -> mode:DIO, clock div:1
11:07:06.843 -> load:0x3fff0030,len:1448
11:07:06.843 -> load:0x40078000,len:14844
11:07:06.843 -> ho 0 tail 12 room 4
11:07:06.843 -> load:0x40080400,len:4
11:07:06.843 -> load:0x40080404,len:3356
11:07:06.843 -> entry 0x4008059c
11:07:08.127 ->
11:07:08.127 -> The device started, now you can pair it with bluetooth!
```

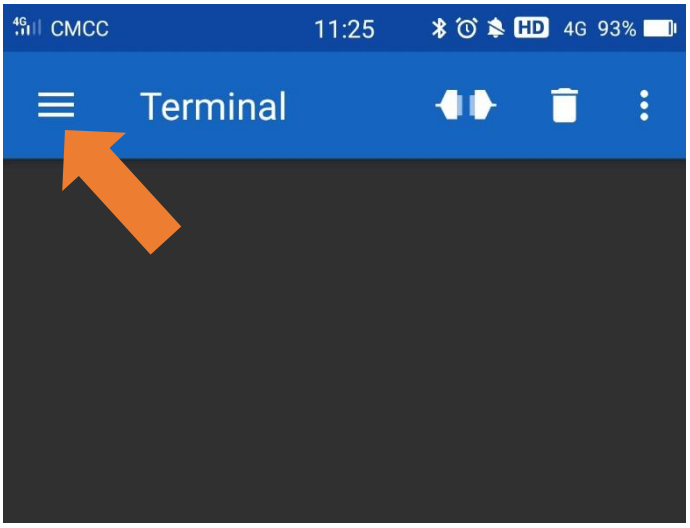
Make sure that the Bluetooth of your phone has been turned on and Serial Bluetooth Terminal has been installed.

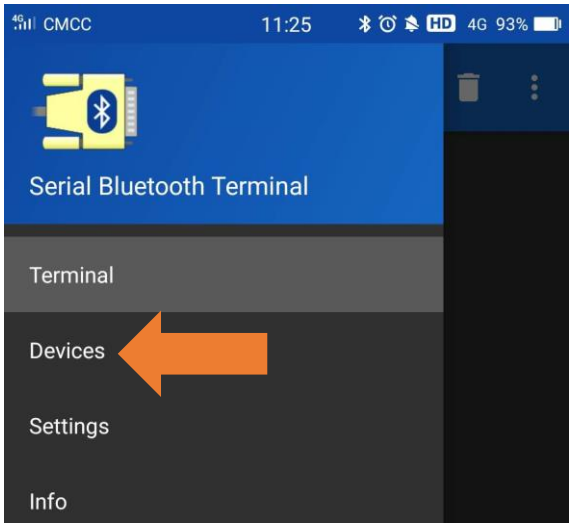
Click "Search" to search Bluetooth devices nearby and select "ESP32 test" to connect to.



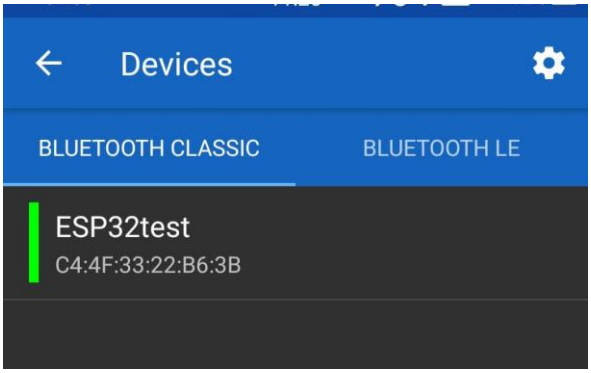


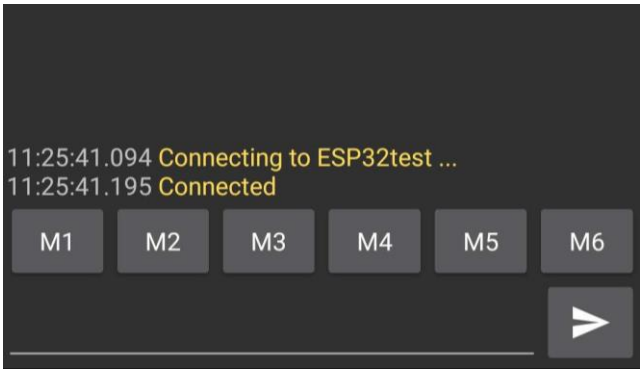
Turn on software APP, click the left of the terminal. Select "Devices"





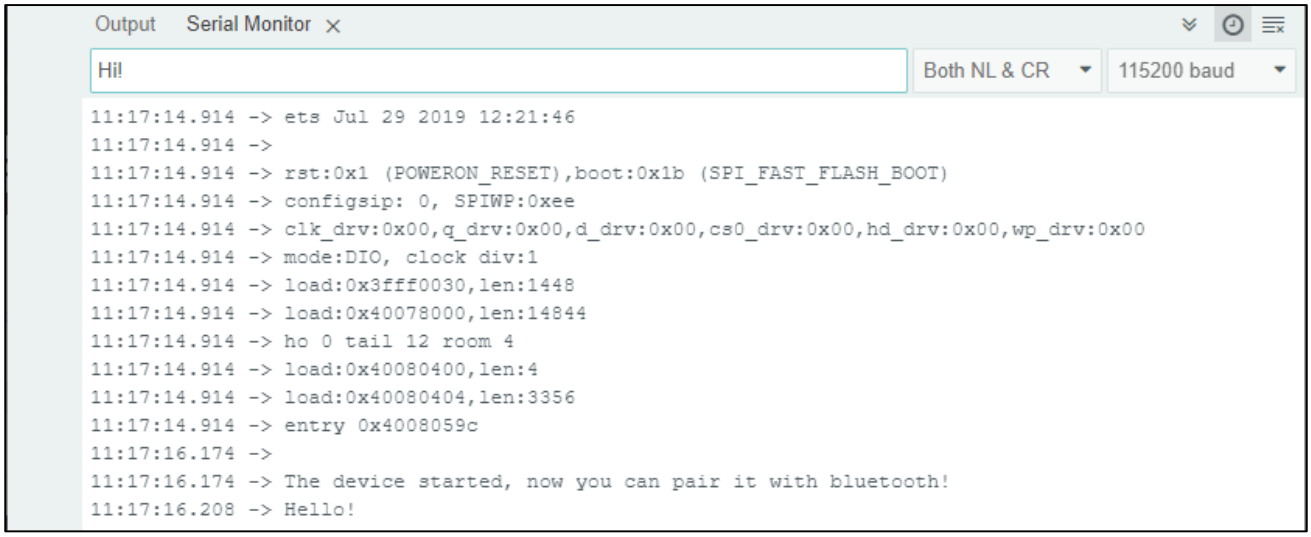
Select ESP32test in classic Bluetooth mode, and a successful connecting prompt will appear as shown on the right illustration.

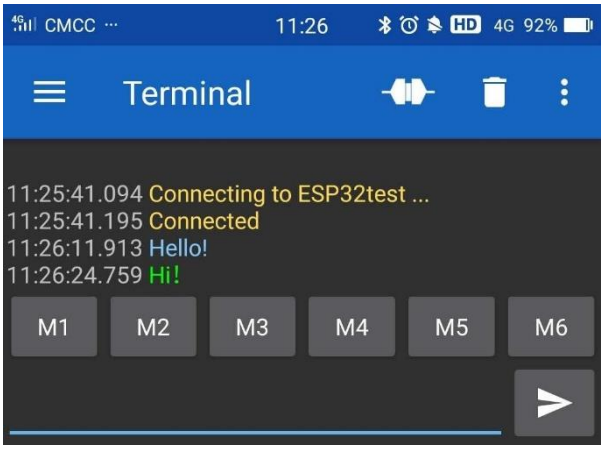




And now data can be transferred between your mobile phone and computer via ESP32-WROVER.

Send 'Hello!'' from your phone, when the computer receives it, reply "Hi" to your phone.





Reference

Class BluetoothSerial

This is a class library used to operate **BluetoothSerial**, which can directly read and set **BluetoothSerial**. Here are some member functions:

begin(localName, isMaster): Initialization function of the Bluetooth
name: name of Bluetooth module; Data type: String
isMaster: bool type, whether to set Bluetooth as Master. By default, it is false.

available(): acquire digits sent from the buffer, if not, return 0.

read(): read data from Bluetooth, data type of return value is int.

readString(): read data from Bluetooth, data type of return value is String.

write(val): send an int data val to Bluetooth.

write(str): send an Srtring data str to Bluetooth.

write(buf, len): Sends the first len data in the buf Array to Bluetooth.

setPin(const char *pin): set a four-digit Bluetooth pairing code. By default, it is 1234

connet(remoteName): connect a Bluetooth named remoteName, data type: String

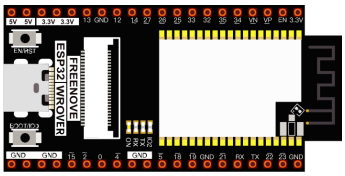
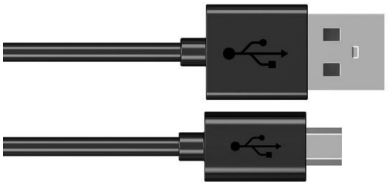
connect(remoteAddress[]): connect the physical address of Bluetooth, data type: uint8-t.

disconnect():disconnect all Bluetooth devices.

end(): disconnect all Bluetooth devices and turn off the Bluetooth, release all occupied space

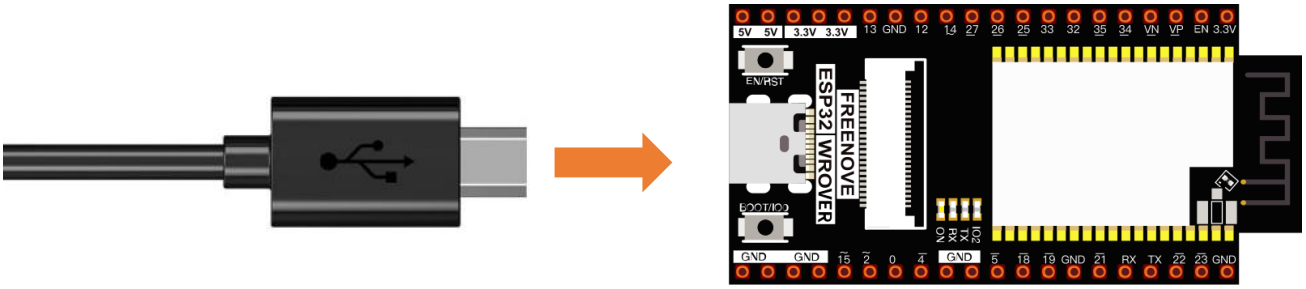
Project 2.2 Bluetooth Low Energy Data Passthrough

Component List

ESP32-WROVER x1 	Micro USB Wire x1 

Circuit

Connect Freenove ESP32 to the computer using the USB cable.



Sketch

Sketch_02.2_BLE

File Edit Sketch Tools Help

ESP32 Wrover Module

Sketch_27.2_BLE_USART.ino

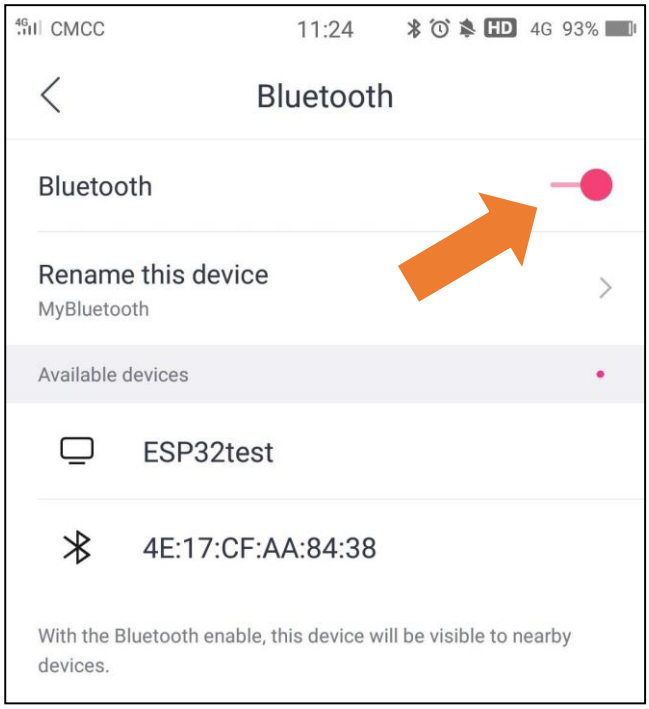
```
59  void setup() {
60    Serial.begin(115200);
61    setupBLE("ESP32_Bluetooth");
62  }
63
64  void loop() {
65    long now = millis();
66    if (now - lastMsg > 100) {
67      if (deviceConnected&&rxload.length()>0) {
68          Serial.println(rxload);
69          rxload="";
70      }
71      if(Serial.available()>0){
72          String str=Serial.readString();
73          const char *newValue=str.c_str();
74          pCharacteristic->setValue(newValue);
75          pCharacteristic->notify();
76      }
77      lastMsg = now;
78    }
79  }
80
```

Serial Bluetooth

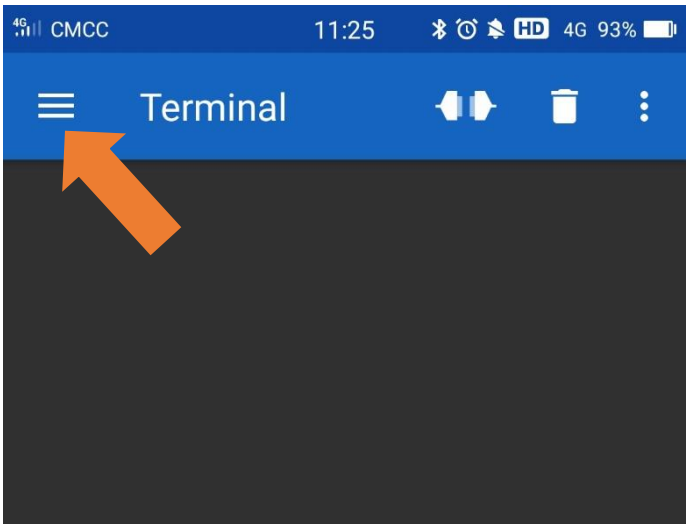
Compile and upload code to ESP32, the operation is similar to the last section.

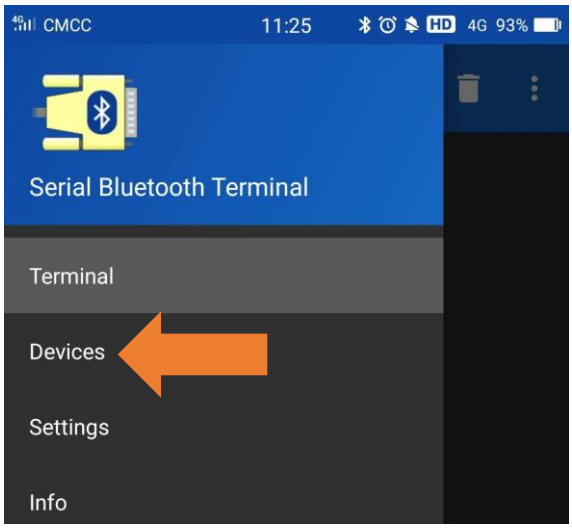
First, make sure you've turned on the mobile phone Bluetooth, and then open the software.

Click "Search" to search Bluetooth devices nearby and select "ESP32 test" to connect to.

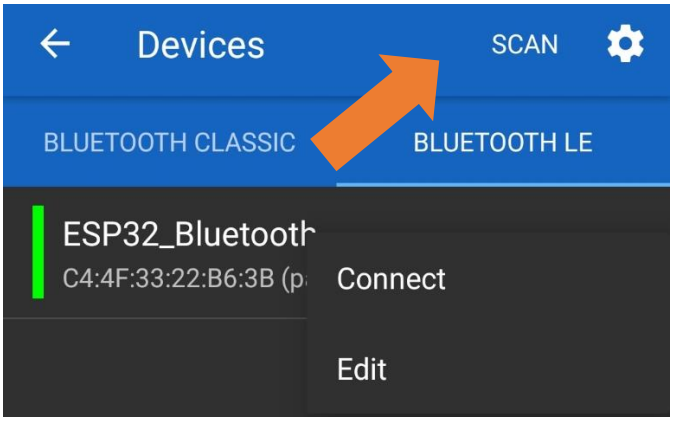


Turn on software APP, click the left of the terminal. Select "Devices"

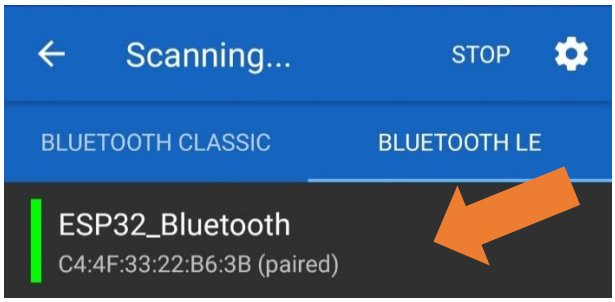




Select BLUETOOTHLE, click SCAN to scan Low Energy Bluetooth devices nearby.

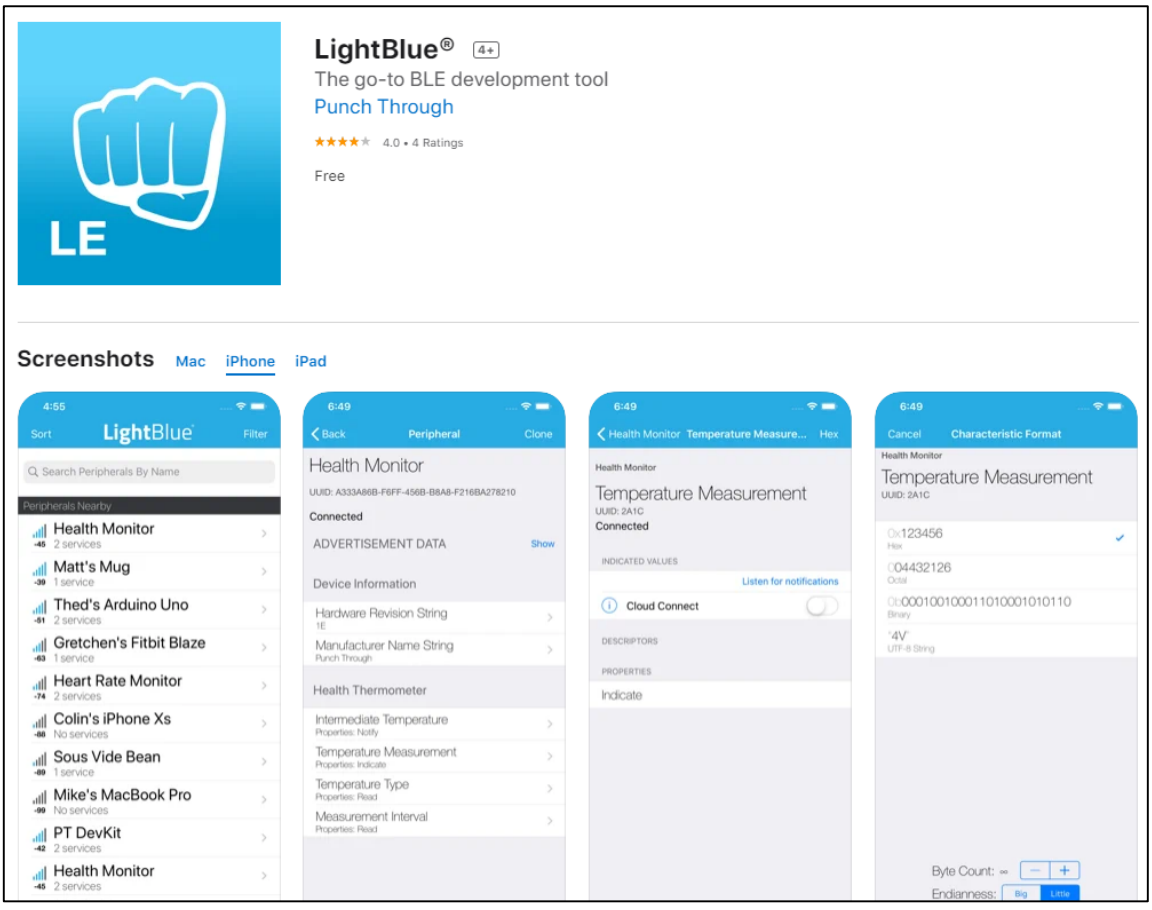


Select"ESP32-Bluetooth"



Lightblue

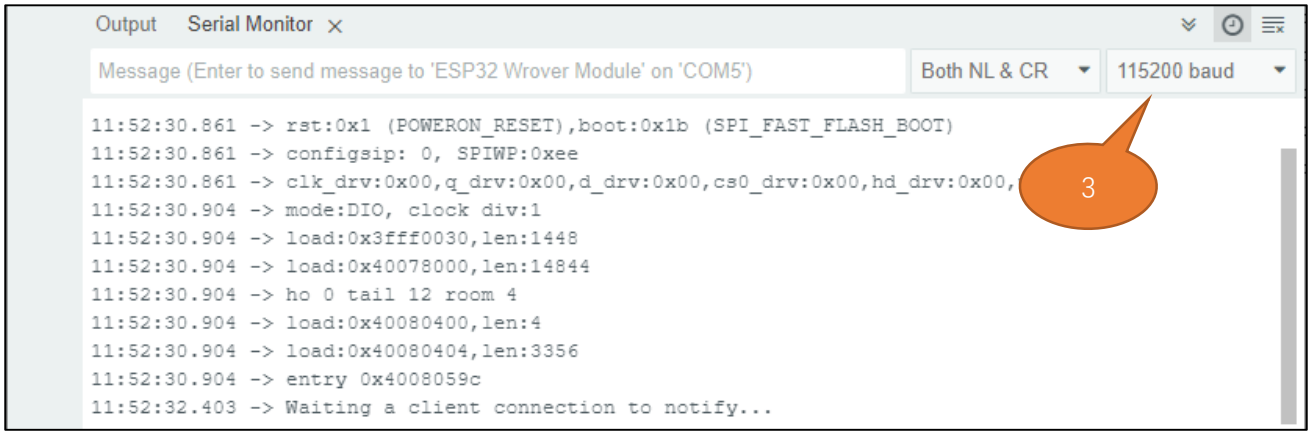
If you can't install Serial Bluetooth on your phone, try LightBlue.If you do not have this software installed on your phone, you can refer to this link:
https://apps.apple.com/us/app/lightblue/id557428110#?platform=iphone.



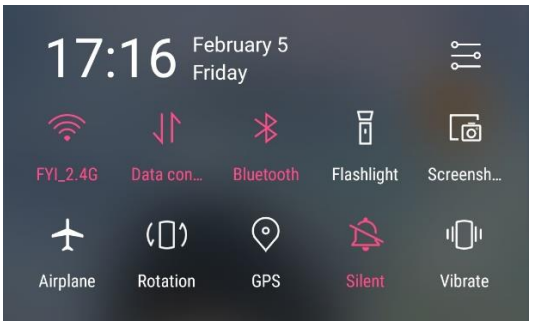
Step1. Upload the code of Project27.2 to ESP32.

Step2. Click on serial monitor.

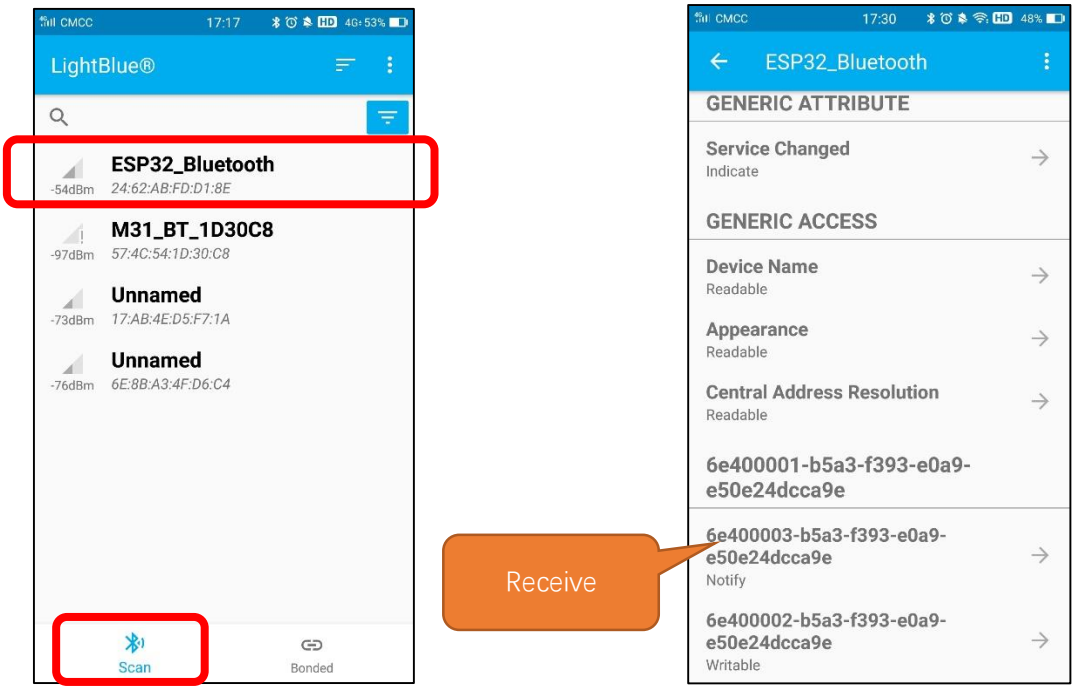
Step3. Set baud rate to 115200.



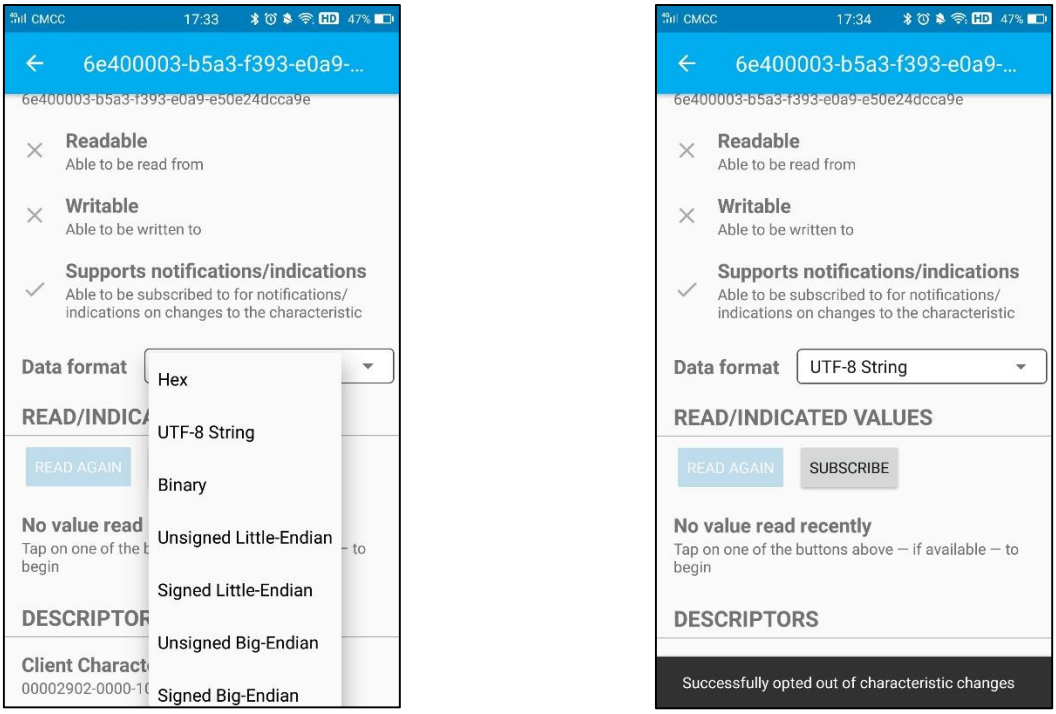
Turn ON Bluetooth on your phone, and open the Lightblue APP.



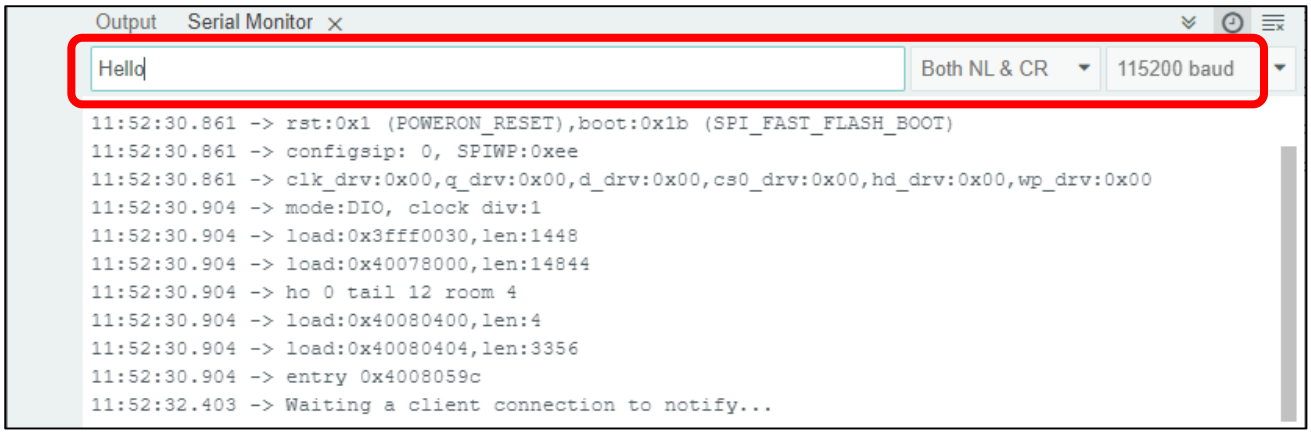
In the Scan page, swipe down to refresh the name of Bluetooth that the phone searches for. Click ESP32_Bluetooth.



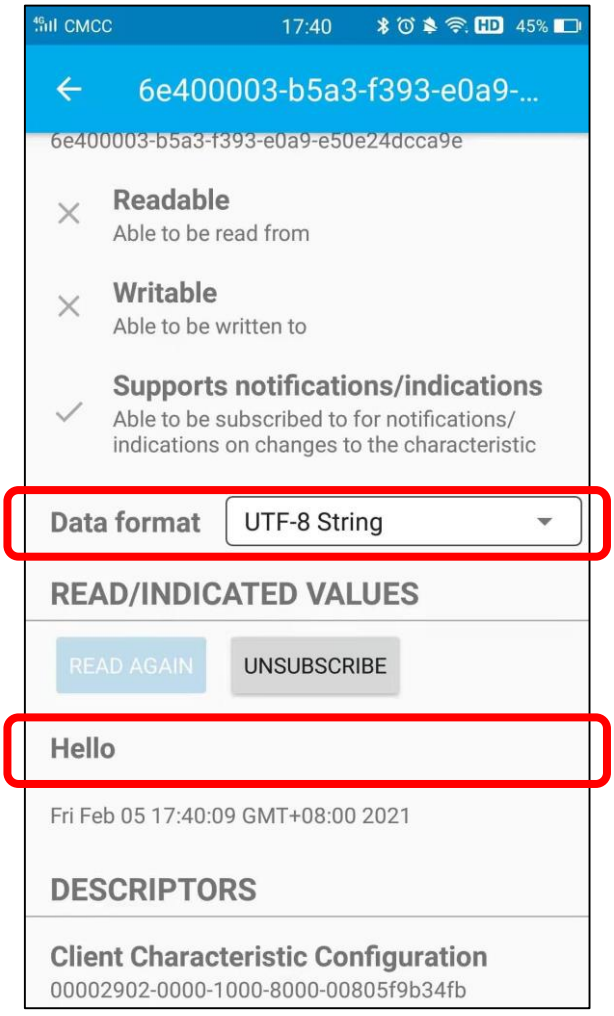
Click "Receive". Select the appropriate Data format in the box to the right of Data Format. For example, HEX for hexadecimal, utf-string for character, Binary for Binary, etc. Then click SUBSCRIBE.



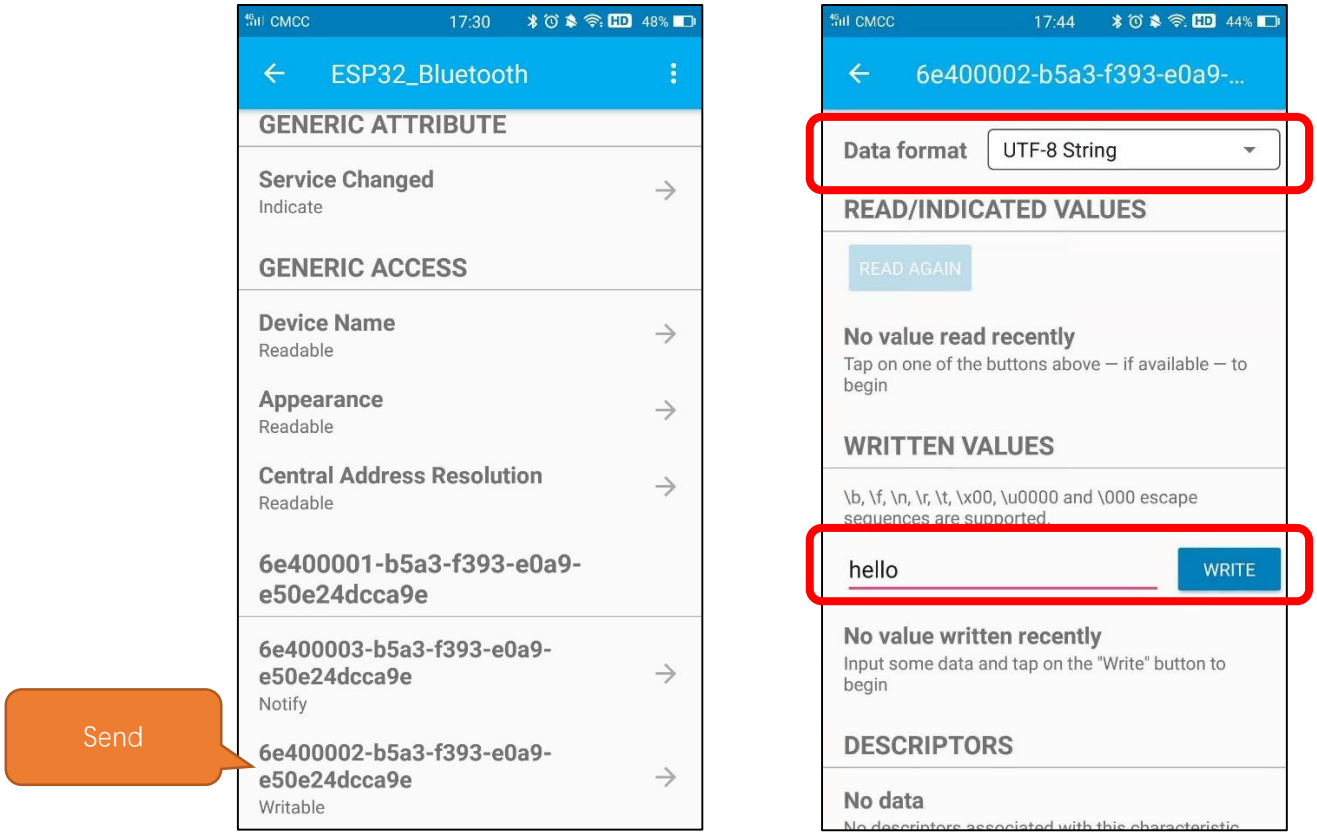
Back to the serial monitor on your computer. You can type anything in the left border of Send, and then click Send.



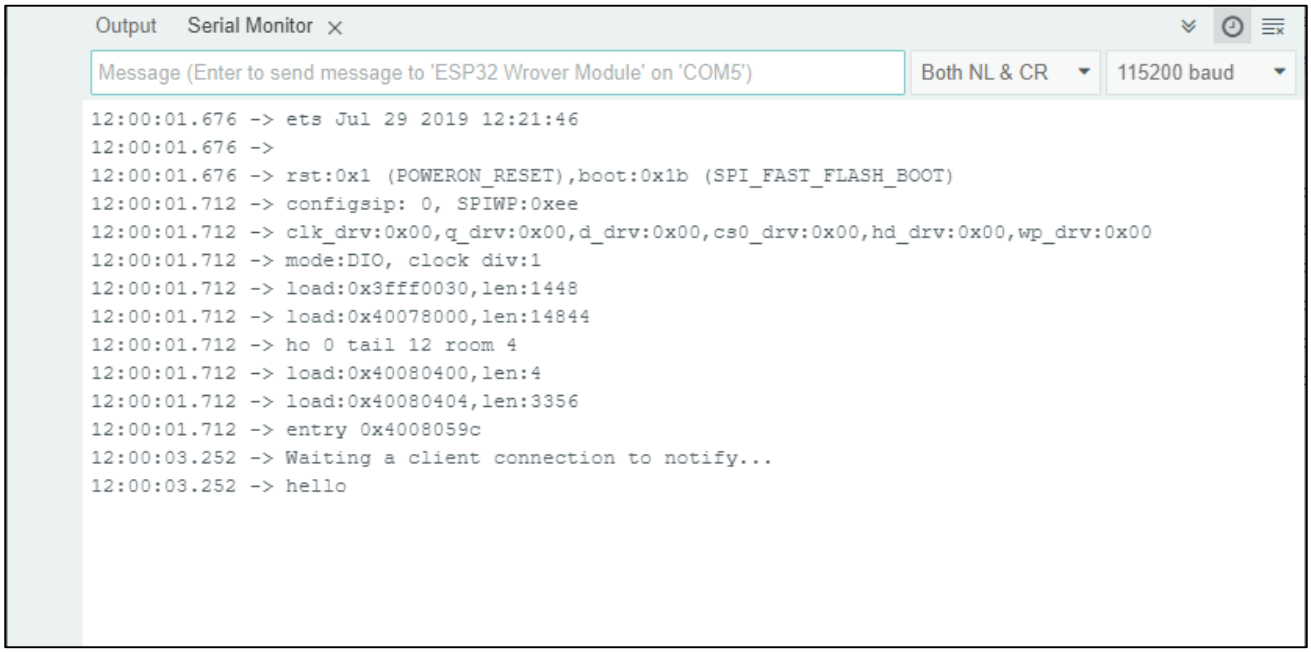
And then you can see the mobile Bluetooth has received the message.



Similarly, you can select "Send" on your phone. Set Data format, and then enter anything in the sending box and click Write to send.



And the computer will receive the message from the mobile Bluetooth.



And now data can be transferred between your mobile phone and computer via ESP32-WROVER.
The following is the program code:

```
#include <BLEDevice.h>
#include <BLEServer.h>
#include <BLEUtils.h>
#include <BLE2902.h>
#include <String.h>

BLECharacteristic *pCharacteristic;
bool deviceConnected = false;
uint8_t txValue = 0;
long lastMsg = 0;
String rxload="Test\n";

#define SERVICE_UUID           "6E400001-B5A3-F393-E0A9-E50E24DCCA9E"
#define CHARACTERISTIC_UUID_RX "6E400002-B5A3-F393-E0A9-E50E24DCCA9E"
#define CHARACTERISTIC_UUID_TX "6E400003-B5A3-F393-E0A9-E50E24DCCA9E"

class MyServerCallbacks: public BLEServerCallbacks {
    void onConnect(BLEServer* pServer) {
      deviceConnected = true;
    };
    void onDisconnect(BLEServer* pServer) {
      deviceConnected = false;
    }
};

class MyCallbacks: public BLECharacteristicCallbacks {
    void onWrite(BLECharacteristic *pCharacteristic) {
      std::string rxValue = pCharacteristic->getValue();
      if (rxValue.length() > 0) {
        rxload="";
        for (int i = 0; i < rxValue.length(); i++){
          rxload +=(char)rxValue[i];
        }
      }
    }
};

void setupBLE(String BLEName){
  const char *ble_name=BLEName.c_str();
  BLEDevice::init(ble_name);
  BLEServer *pServer = BLEDevice::createServer();
  pServer->setCallbacks(new MyServerCallbacks());
```

```
43    BLEService *pService = pServer->createService(SERVICE_UUID);
44    pCharacteristic=
      pService->createCharacteristic(CHARACTERISTIC_UUID_TX,BLECharacteristic::PROPERTY_NOTIFY);
45    pCharacteristic->addDescriptor(new BLE2902());
46    BLECharacteristic *pCharacteristic =
      pService->createCharacteristic(CHARACTERISTIC_UUID_RX,BLECharacteristic::PROPERTY_WRITE);
47    pCharacteristic->setCallbacks(new MyCallbacks());
48    pService->start();
49    pServer->getAdvertising()->start();
50    Serial.println("Waiting a client connection to notify...");
51  }
52
53  void setup() {
54    Serial.begin(9600);
55    setupBLE("ESP32_Bluetooth");
56  }
57
58  void loop() {
59    long now = millis();
60    if (now - lastMsg > 1000) {
61      if (deviceConnected&&rxload.length()>0) {
62          Serial.println(rxload);
63          rxload="";
64      }
65      if(Serial.available()>0){
66          String str=Serial.readString();
67          const char *newValue=str.c_str();
68          pCharacteristic->setValue(newValue);
69          pCharacteristic->notify();
70      }
71      lastMsg = now;
72    }
73  }
```

Define the specified UUID number for BLE vendor.

```
13  #define SERVICE_UUID           "6E400001-B5A3-F393-E0A9-E50E24DCCA9E"
14  #define CHARACTERISTIC_UUID_RX "6E400002-B5A3-F393-E0A9-E50E24DCCA9E"
15  #define CHARACTERISTIC_UUID_TX "6E400003-B5A3-F393-E0A9-E50E24DCCA9E"
```

Write a Callback function for BLE server to manage connection of BLE.

```
17  class MyServerCallbacks: public BLEServerCallbacks {
18      void onConnect(BLEServer* pServer) {
19        deviceConnected = true;
20      };
21      void onDisconnect(BLEServer* pServer) {
22        deviceConnected = false;
23      }
24  };
```

Write Callback function with BLE features. When it is called, as the mobile terminal send data to ESP32, it will store them into reload.

```
26  class MyCallbacks: public BLECharacteristicCallbacks {
27      void onWrite(BLECharacteristic *pCharacteristic) {
28        std::string rxValue = pCharacteristic->getValue();
29        if (rxValue.length() > 0) {
30          rxload="";
31          for (int i = 0; i < rxValue.length(); i++){
32            rxload +=(char)rxValue[i];
33          }
34        }
35      }
36  };
```

Initialize the BLE function and name it.

```
55  setupBLE("ESP32_Bluetooth");
```

When the mobile phone send data to ESP32 via BLE Bluetooth, it will print them out with serial port; When the serial port of ESP32 receive data, it will send them to mobile via BLE Bluetooth.

```
59    long now = millis();
60    if (now - lastMsg > 1000) {
61      if (deviceConnected&&rxload.length()>0) {
62          Serial.println(rxload);
63          rxload="";
64      }
65      if(Serial.available()>0){
66          String str=Serial.readString();
67          const char *newValue=str.c_str();
68          pCharacteristic->setValue(newValue);
69          pCharacteristic->notify();
70      }
71      lastMsg = now;
72    }
```

The design for creating the BLE server is:

1. Create a BLE Server
2. Create a BLE Service
3. Create a BLE Characteristic on the Service
4. Create a BLE Descriptor on the characteristic
5. Start the service.
6. Start advertising.

```
38  void setupBLE(String BLEName){
39    const char *ble_name=BLEName.c_str();
40    BLEDevice::init(ble_name);
41    BLEServer *pServer = BLEDevice::createServer();
42    pServer->setCallbacks(new MyServerCallbacks());
43    BLEService *pService = pServer->createService(SERVICE_UUID);
44    pCharacteristic=
      pService->createCharacteristic(CHARACTERISTIC_UUID_TX,BLECharacteristic::PROPERTY_NOTIFY);
45    pCharacteristic->addDescriptor(new BLE2902());
46    BLECharacteristic *pCharacteristic =
      pService->createCharacteristic(CHARACTERISTIC_UUID_RX,BLECharacteristic::PROPERTY_WRITE);
47    pCharacteristic->setCallbacks(new MyCallbacks());
48    pService->start();
49    pServer->getAdvertising()->start();
50    Serial.println("Waiting a client connection to notify...");
51  }
```

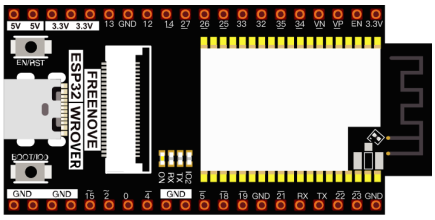
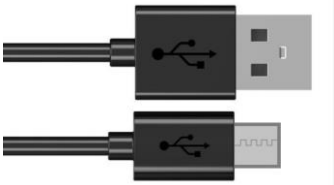
Chapter 3 Read and Write the Sdcard

Note: The SD card chapter only applies to the ESP32 WROVER development board with an SD card slot on the back. If your ESP32 WROVER does not have an SD card slot on the back, please skip this chapter.

An SDcard slot is integrated on the back of the ESP32 WROVER. In this chapter we learn how to use ESP32 to read and write SDcard.

Project 3.1 SDMMC Test

Component List

ESP32 WROVER x1 	USB cable x1 	SDcard x1 

Component knowledge

SD card read and write method

ESP32 has two ways to use SD card, one is to use the SPI interface to access the SD card, and the other is to use the SDMMC interface to access the SD card. SPI mode uses 4 IOs to access SD card. The SDMMC has one-bit bus mode and four-bit bus mode. In one-bit bus mode, SDMMC use 3 IOs to access SD card. In four-bit bus mode, SDMMC uses 6 IOs to access the SD card.

The above three methods can all be used to access the SD card, the difference is that the access speed is different.

In the four-bit bus mode of SDMMC, the reading and writing speed of accessing the SD card is the fastest. In the one-bit bus mode of SDMMC, the access speed is about 80% of the four-bit bus mode. The access speed of SPI is the slowest, which is about 50% of the four-bit bus mode of SDMMC.

Usually, we recommend using the one-bit bus mode to access the SD card, because in this mode, we only need to use the least pin IO to access the SD card with good performance and speed.

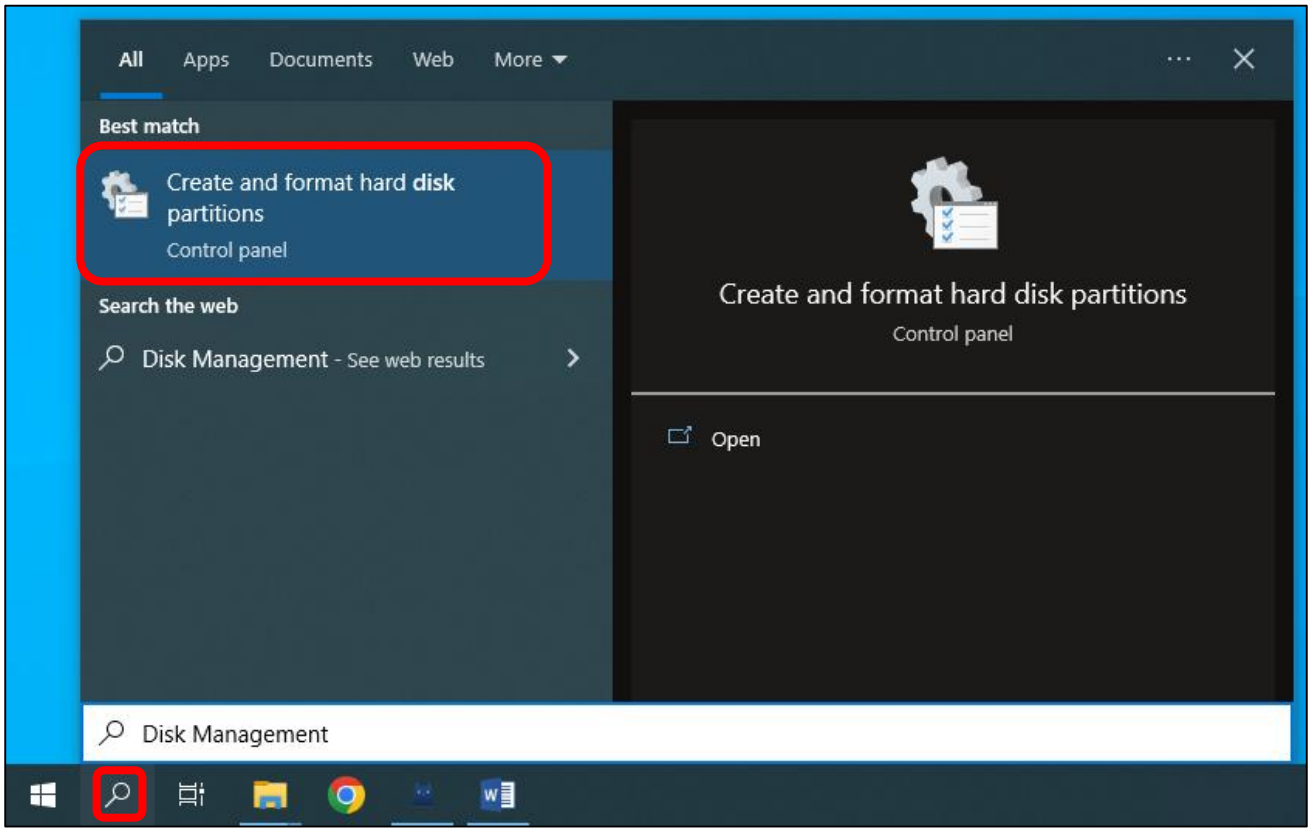
Format SD card

Before starting the tutorial, we need to create a drive letter for the blank SD card and format it. This step requires a card reader and SD card. Please prepare them in advance. Below we will guide you to do it on different computer systems. You can choose the guide that matches your computer.

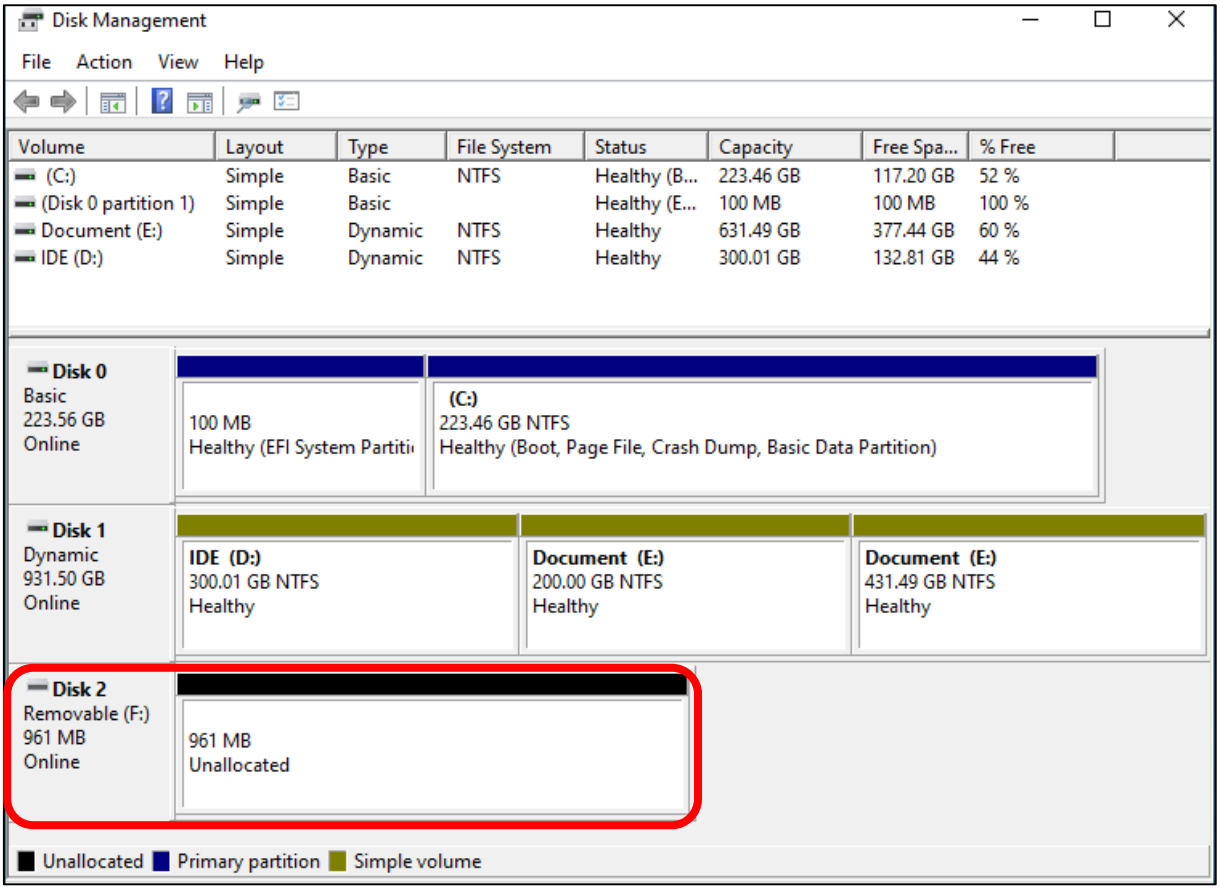
Windows

Insert the SD card into the card reader, then insert the card reader into the computer.

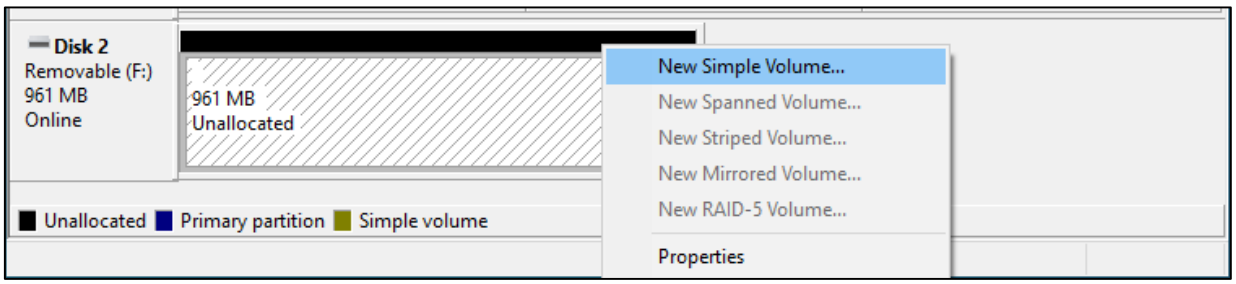
In the Windows search box, enter "Disk Management" and select "Create and format hard disk partitions".



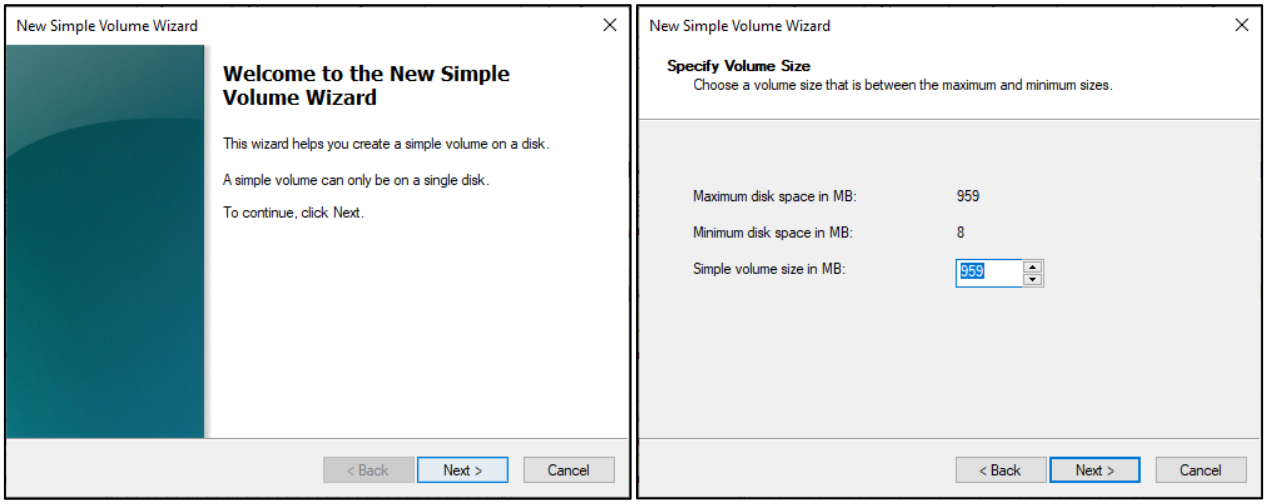
In the new pop-up window, find an unallocated volume close to 1G in size.



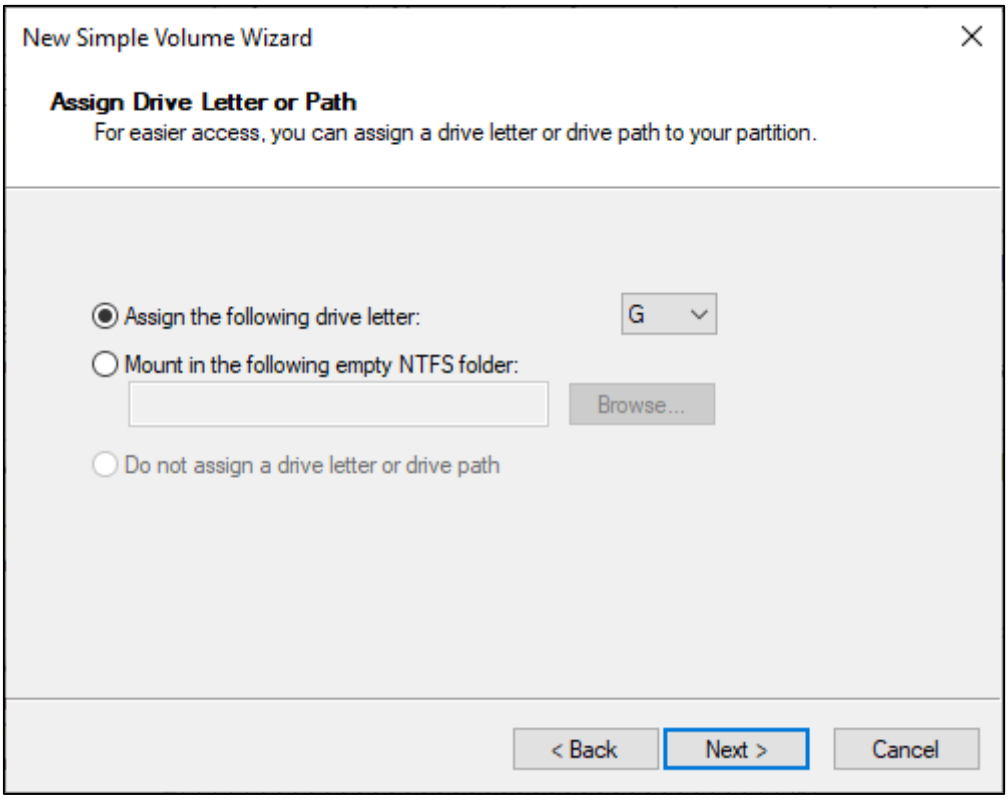
Click to select the volume, right-click and select "New Simple Volume".



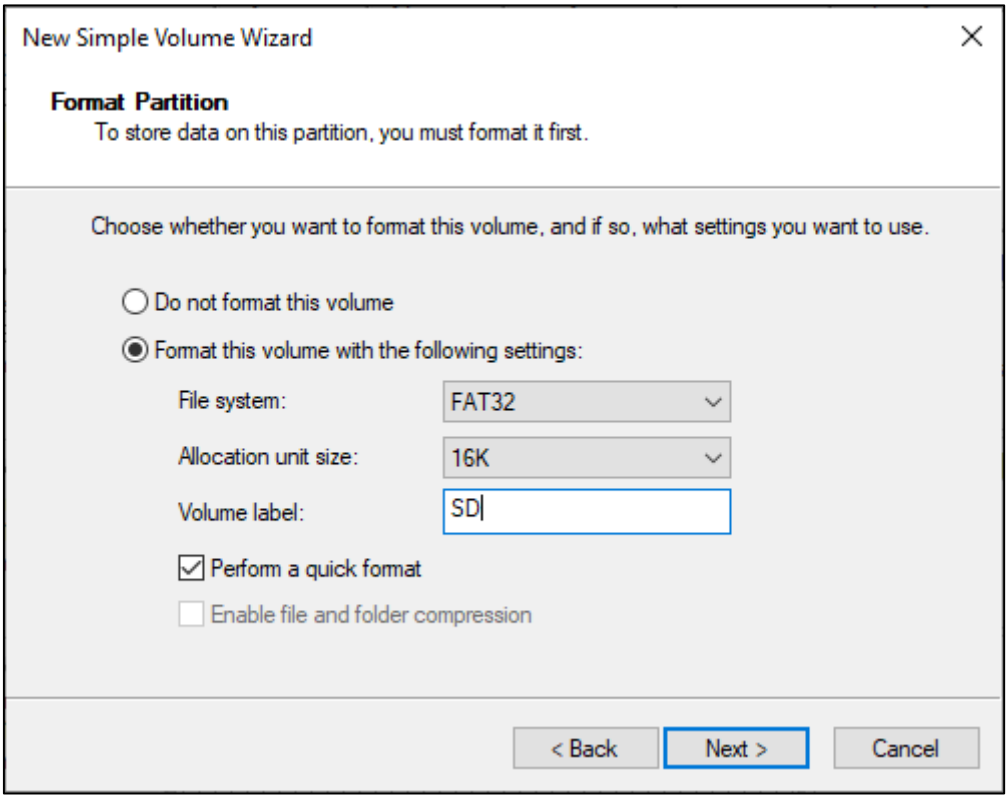
Click Next.



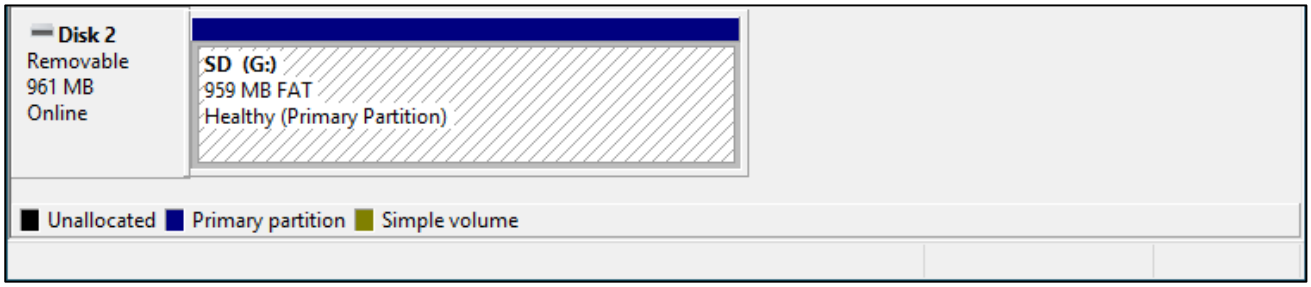
You can choose the drive letter on the right, or you can choose the default. By default, just click Next.



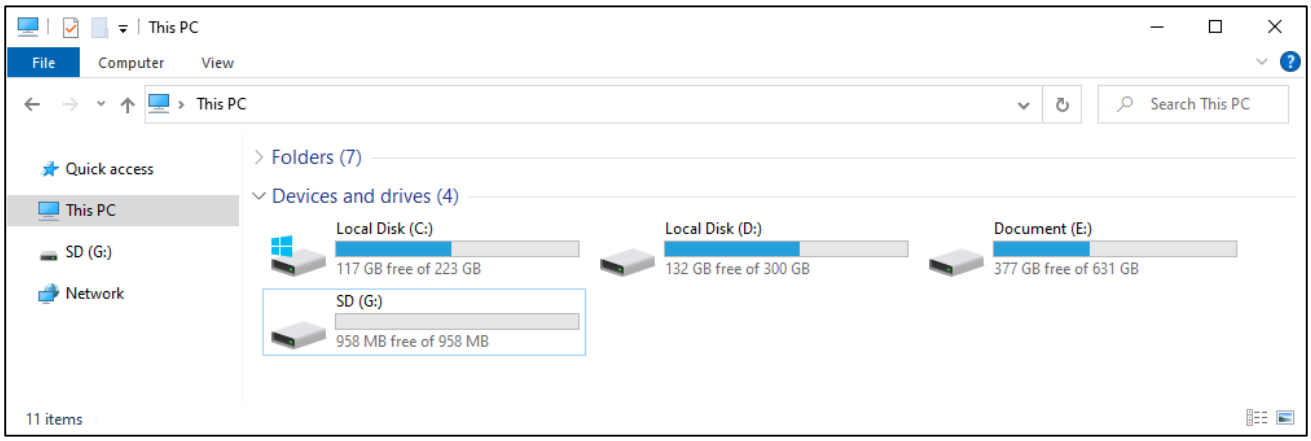
File system is FAT(or FAT32). The Allocation unit size is 16K, and the Volume label can be set to any name. After setting, click Next.



Click Finish. Wait for the SD card initialization to complete.

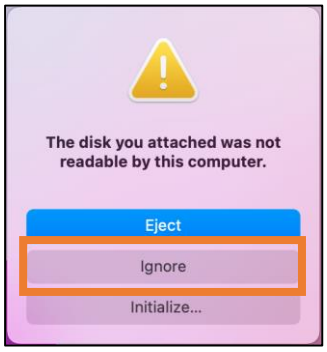


At this point, you can see the SD card in This PC.

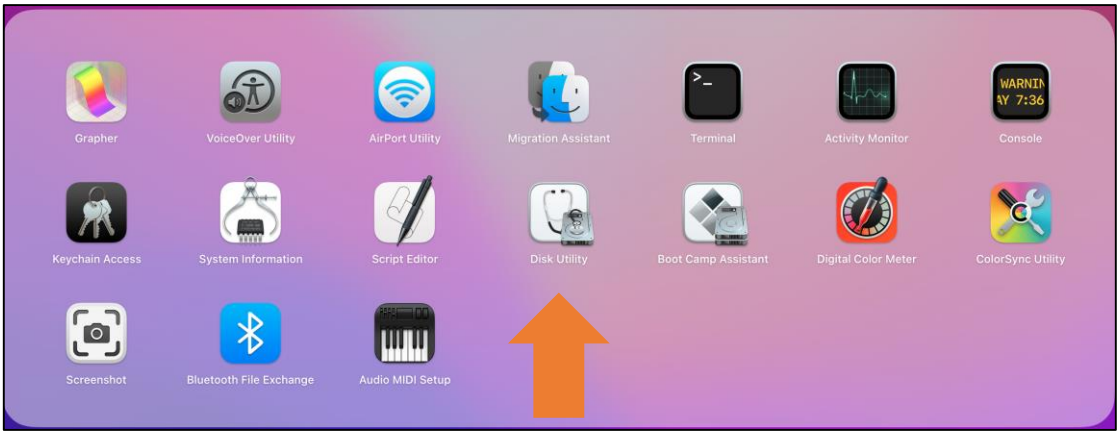


MAC

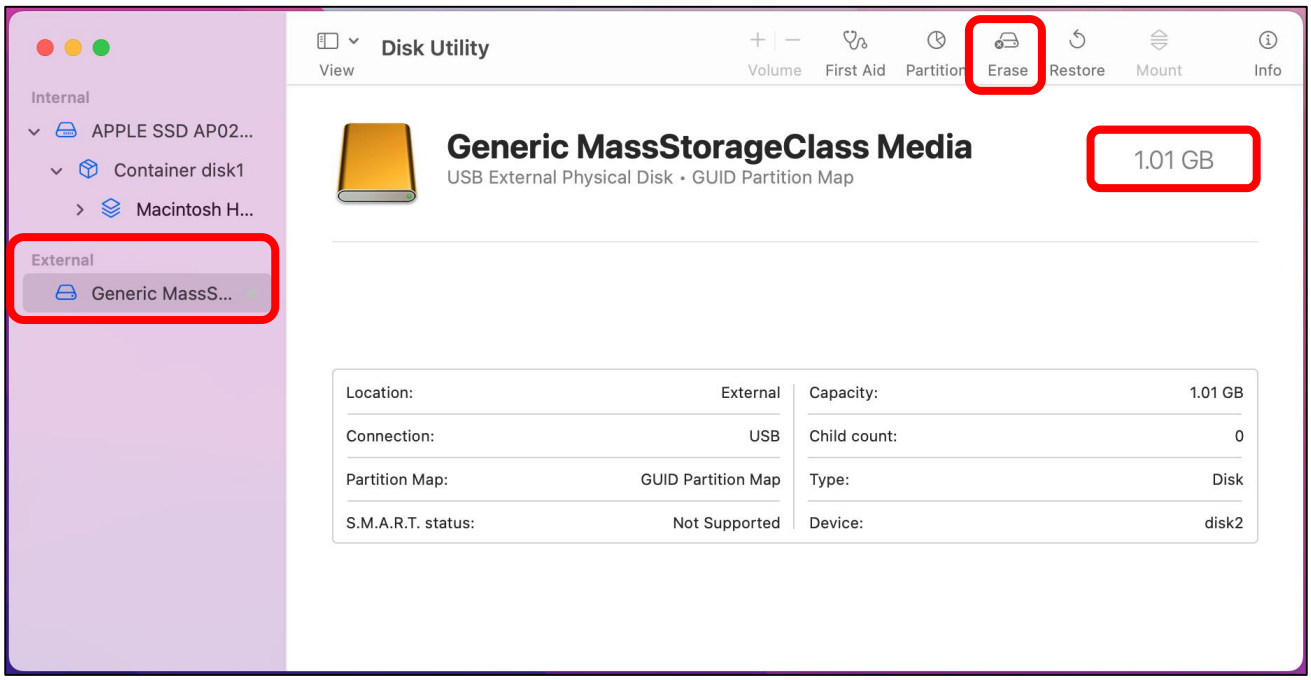
Insert the SD card into the card reader, then insert the card reader into the computer. Some computers will prompt the following information, please click to ignore it.



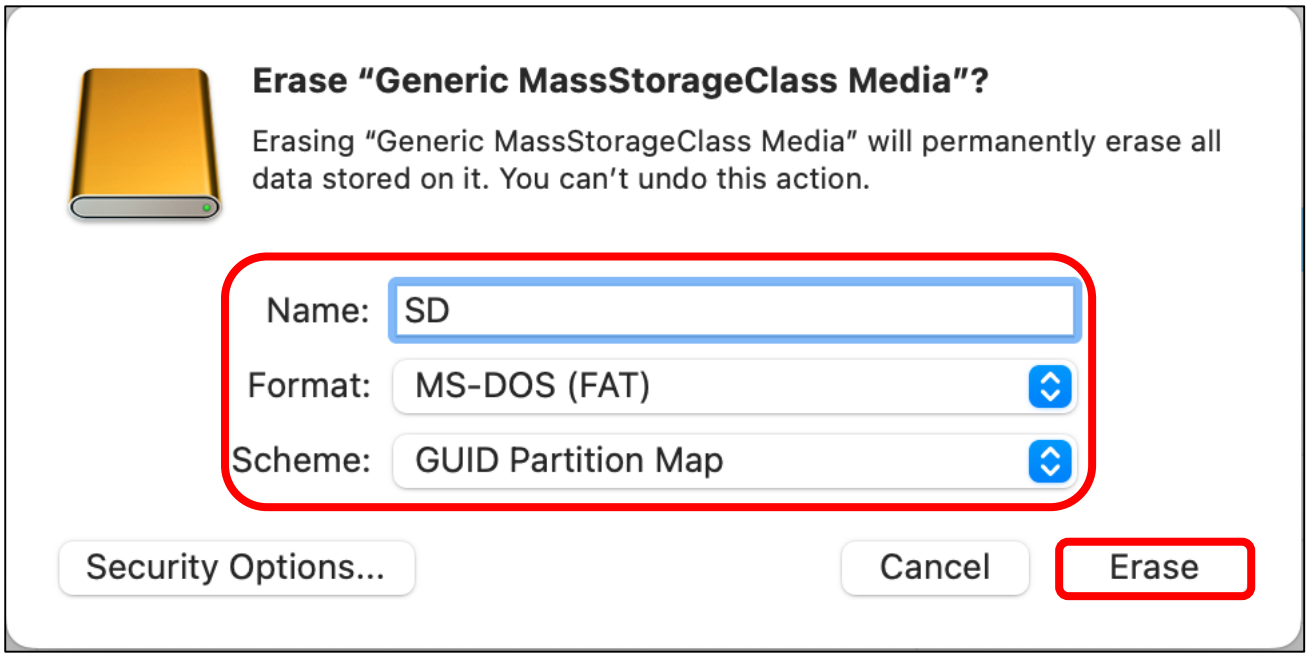
Find "Disk Utility" in the MAC system and click to open it.



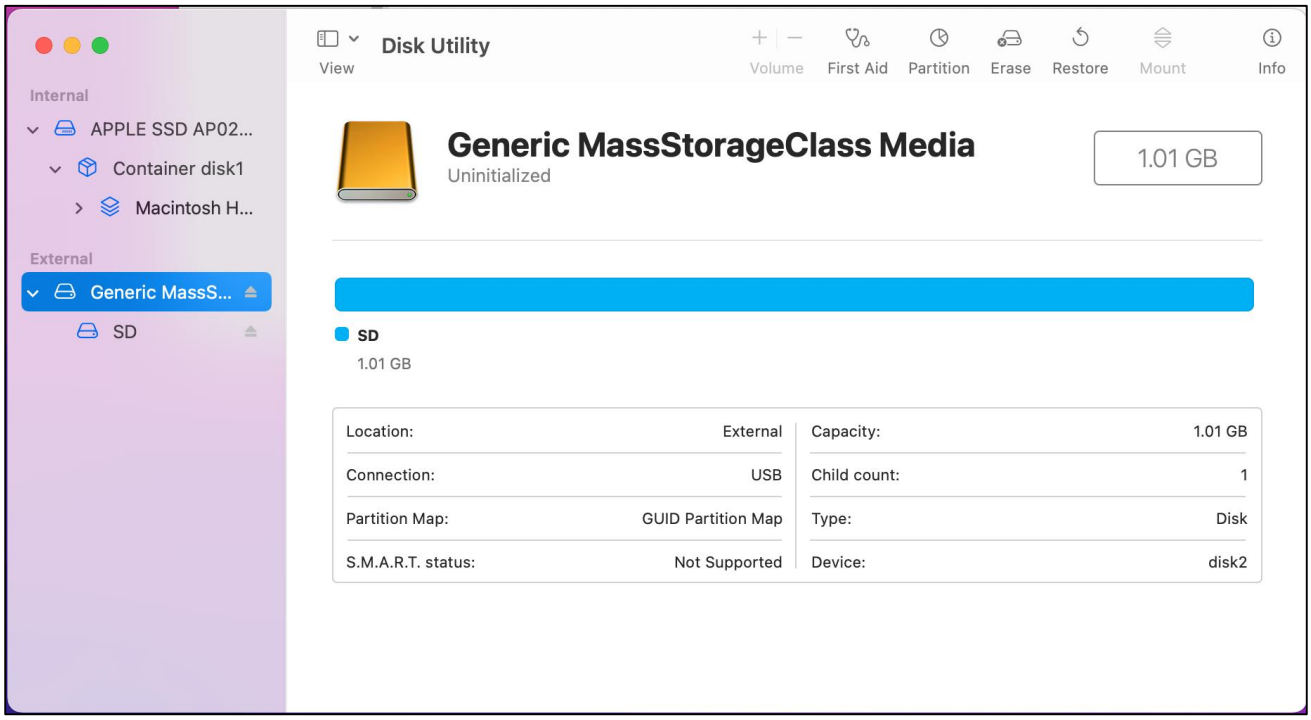
Select "Generic MassStorageClass Media", note that its size is about 1G. Please do not choose wrong item. Click "Erase".



Select the configuration as shown in the figure below, and then click "Erase".

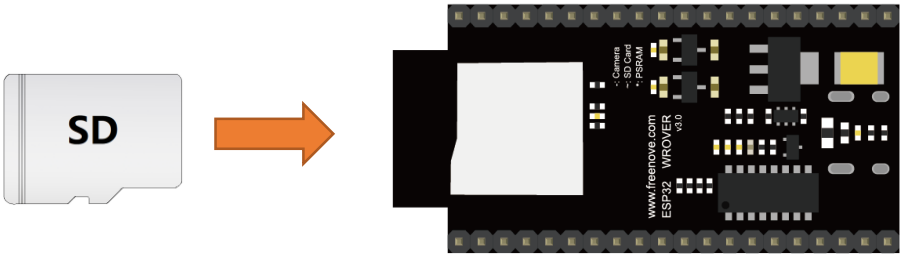


Wait for the formatting to complete. When finished, it will look like the picture below. At this point, you can see a new disk on the desktop named "SD".

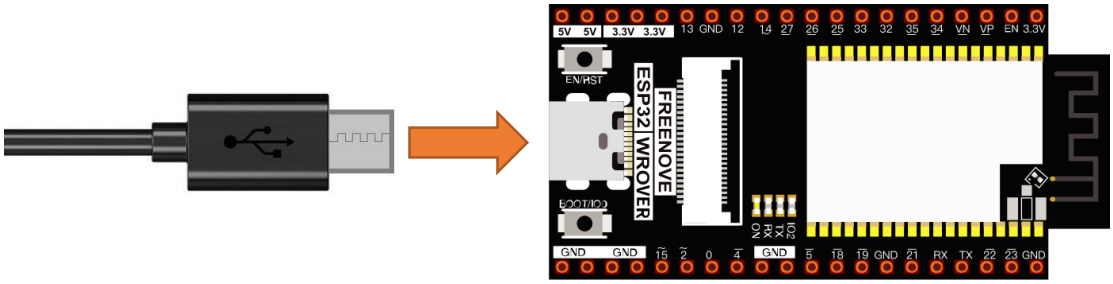


Circuit

Before connecting the USB cable, insert the SD card into the SD card slot on the back of the ESP32.

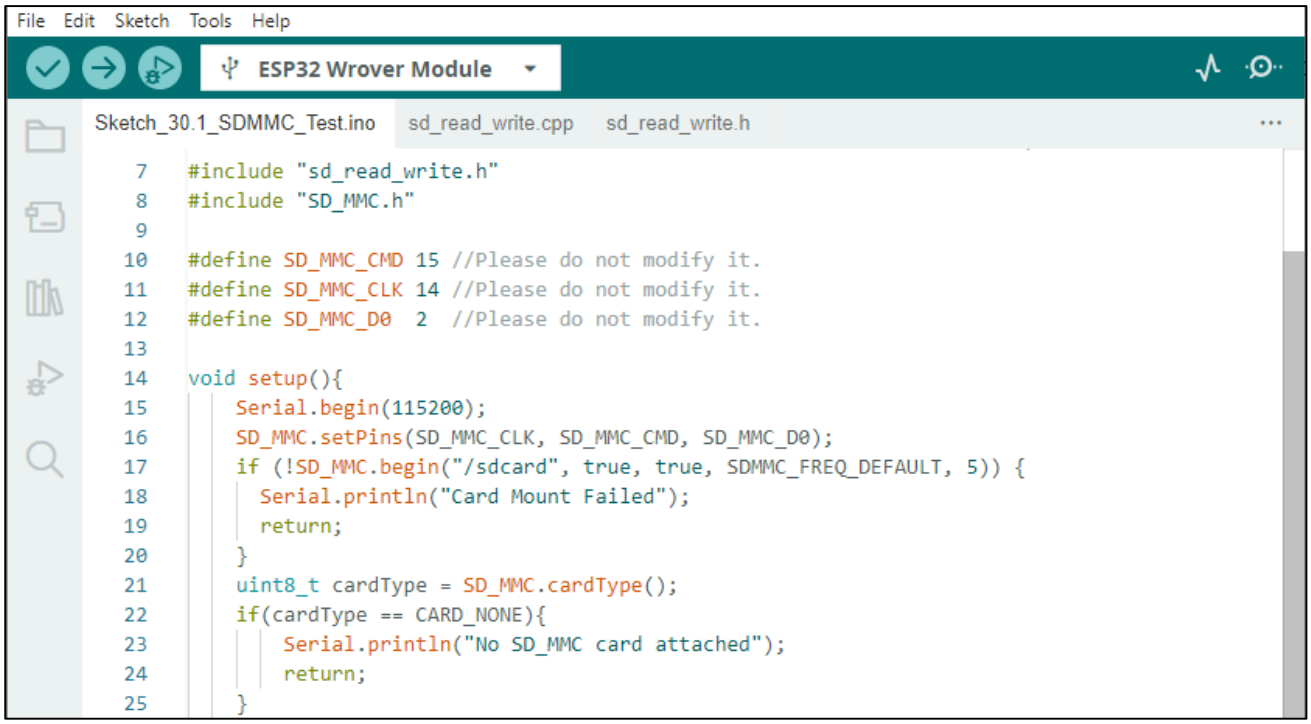


Connect Freenove ESP32 to the computer using the USB cable.



Sketch

Sketch_03.1_SDMMC_Test




```
#include "sd_read_write.h"
#include "SD_MMC.h"

#define SD_MMC_CMD 15 //Please do not modify it.
#define SD_MMC_CLK 14 //Please do not modify it.
#define SD_MMC_D0  2  //Please do not modify it.

void setup(){
    Serial.begin(115200);
    SD_MMC.setPins(SD_MMC_CLK, SD_MMC_CMD, SD_MMC_D0);
    if (!SD_MMC.begin("/sdcard", true, true, SDMMC_FREQ_DEFAULT, 5)) {
      Serial.println("Card Mount Failed");
      return;
    }
    uint8_t cardType = SD_MMC.cardType();
    if(cardType == CARD_NONE){
        Serial.println("No SD_MMC card attached");
        return;
    }
```

Compile and upload the code to ESP32, open the serial monitor, and press the RST button on the board. You can see the printout as shown below.

```
14:47:21.289 -> Listing directory: /System Volume Information
14:47:21.289 ->   FILE: WPSettings.dat  SIZE: 12
14:47:21.289 ->   FILE: IndexerVolumeGuid  SIZE: 76
14:47:21.289 ->   FILE: test.txt  SIZE: 1048576
14:47:21.289 ->   FILE: foo.txt  SIZE: 13
14:47:21.289 ->   DIR : music
14:47:21.289 -> Listing directory: /music
14:47:21.289 ->   FILE: 01.mp3  SIZE: 3528199
14:47:21.289 ->   FILE: Good Time.mp3  SIZE: 3291708
14:47:21.289 ->   FILE: Jingle Bells.mp3  SIZE: 8004409
14:47:21.289 -> Writing file: /hello.txt
14:47:21.289 -> File written
14:47:21.416 -> Appending to file: /hello.txt
14:47:21.416 -> Message appended
14:47:21.461 -> Reading file: /hello.txt
14:47:21.461 -> Read from file: Hello World!
14:47:21.461 -> Deleting file: /foo.txt
14:47:21.502 -> File deleted
14:47:21.502 -> Renaming file /hello.txt to /foo.txt
14:47:21.535 -> File renamed
14:47:21.535 -> Reading file: /foo.txt
14:47:21.535 -> Read from file: Hello World!
14:47:22.054 -> 1048576 bytes read for 547 ms
14:47:22.758 -> 1048576 bytes written for 687 ms
14:47:22.804 -> Total space: 953MB
14:47:22.804 -> Used space: 15MB
```

The following is the program code:

```
#include "sd_read_write.h"
#include "SD_MMC.h"

#define SD_MMC_CMD 15 //Please do not modify it.
#define SD_MMC_CLK 14 //Please do not modify it.
#define SD_MMC_D0  2  //Please do not modify it.

void setup(){
    Serial.begin(115200);
    SD_MMC.setPins(SD_MMC_CLK, SD_MMC_CMD, SD_MMC_D0);
    if (!SD_MMC.begin("/sdcard", true, true, SDMMC_FREQ_DEFAULT, 5)) {
      Serial.println("Card Mount Failed");
      return;
    }
    uint8_t cardType = SD_MMC.cardType();
    if(cardType == CARD_NONE){
        Serial.println("No SD_MMC card attached");
        return;
    }
    Serial.print("SD_MMC Card Type: ");
    if(cardType == CARD_MMC){
        Serial.println("MMC");
    } else if(cardType == CARD_SD){
        Serial.println("SDSC");
    } else if(cardType == CARD_SDHC){
        Serial.println("SDHC");
    } else {
        Serial.println("UNKNOWN");
    }

    uint64_t cardSize = SD_MMC.cardSize() / (1024 * 1024);
    Serial.printf("SD_MMC Card Size: %lluMB\n", cardSize);

    listDir(SD_MMC, "/", 0);

    createDir(SD_MMC, "/mydir");
    listDir(SD_MMC, "/", 0);

    removeDir(SD_MMC, "/mydir");
    listDir(SD_MMC, "/", 2);

    writeFile(SD_MMC, "/hello.txt", "Hello ");
    appendFile(SD_MMC, "/hello.txt", "World!\n");
```

```
    readFile(SD_MMC, "/hello.txt");

    deleteFile(SD_MMC, "/foo.txt");
    renameFile(SD_MMC, "/hello.txt", "/foo.txt");
    readFile(SD_MMC, "/foo.txt");

    testFileIO(SD_MMC, "/test.txt");

    Serial.printf("Total space: %lluMB\r\n", SD_MMC.totalBytes() / (1024 * 1024));
    Serial.printf("Used space: %lluMB\r\n", SD_MMC.usedBytes() / (1024 * 1024));
}

void loop(){
  delay(10000);
}
```

Add the SD card drive header file.

```
#include "sd_read_write.h"
#include "SD_MMC.h"
```

Defines the drive pins of the SD card. Please do not modify it. Because these pins are fixed.

```
#define SD_MMC_CMD 15 //Please do not modify it.
#define SD_MMC_CLK 14 //Please do not modify it.
#define SD_MMC_D0  2  //Please do not modify it.
```

Initialize the serial port function. Sets the drive pin for SDMMC one-bit bus mode.

```
    Serial.begin(115200);
    SD_MMC.setPins(SD_MMC_CLK, SD_MMC_CMD, SD_MMC_D0);
```

Set the mount point of the SD card, set SDMMC to one-bit bus mode, and set the read and write speed to 20MHz.

```
    if (!SD_MMC.begin("/sdcard", true, true, SDMMC_FREQ_DEFAULT, 5)) {
      Serial.println("Card Mount Failed");
      return;
    }
```

Get the type of SD card and print it out through the serial port.

```
    uint8_t cardType = SD_MMC.cardType();
    if(cardType == CARD_NONE){
        Serial.println("No SD_MMC card attached");
        return;
    }
    Serial.print("SD_MMC Card Type: ");
    if(cardType == CARD_MMC){
        Serial.println("MMC");
    } else if(cardType == CARD_SD){
        Serial.println("SDSC");
    } else if(cardType == CARD_SDHC){
        Serial.println("SDHC");
```

```
27      } else {
28          Serial.println("UNKNOWN");
29      }
```

Call the listDir() function to read the folder and file names in the SD card, and print them out through the serial port. This function can be found in "sd_read_write.cpp".

```
34  listDir(SD_MMC, "/", 0);
```

Call createDir() to create a folder, and call removeDir() to delete a folder.

```
36  createDir(SD_MMC, "/mydir");
39  removeDir(SD_MMC, "/mydir");
```

Call writeFile() to write any content to the txt file. If there is no such file, create this file first.

Call appendFile() to append any content to txt.

Call readFile() to read the content in txt and print it via the serial port.

```
42  writeFile(SD_MMC, "/hello.txt", "Hello ");
43  appendFile(SD_MMC, "/hello.txt", "World!\n");
44  readFile(SD_MMC, "/hello.txt");
```

Call deleteFile() to delete a specified file.

Call renameFile() to copy a file and rename it.

```
46      deleteFile(SD_MMC, "/foo.txt");
47      renameFile(SD_MMC, "/hello.txt", "/foo.txt");
```

Call the testFileIO() function to test the time it takes to read 512 bytes and the time it takes to write 2048*512 bytes of data.

```
50  testFileIO(SD_MMC, "/test.txt");
```

Print the total size and used size of the SD card via the serial port.

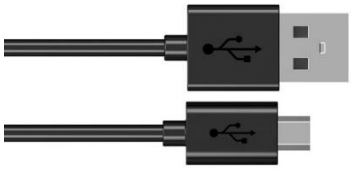
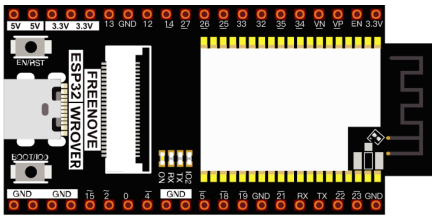
```
52      Serial.printf("Total space: %lluMB\r\n", SD_MMC.totalBytes() / (1024 * 1024));
53      Serial.printf("Used space: %lluMB\r\n", SD_MMC.usedBytes() / (1024 * 1024));
```

Chapter 4 WiFi Working Modes

In this chapter, we'll focus on the WiFi infrastructure for ESP32-WROVER.
ESP32-WROVER has 3 different WiFi operating modes: station mode, AP mode and AP+station mode. All WiFi programming projects must be configured with WiFi operating mode before using WiFi, otherwise WiFi cannot be used.

Project 4.1 Station mode

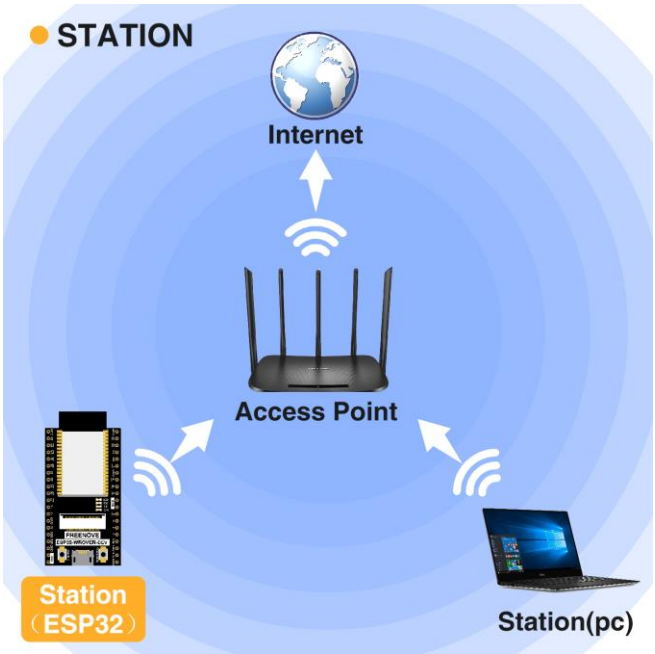
Component List

Micro USB Wire x1 	ESP32-WROVER x1 

Component knowledge

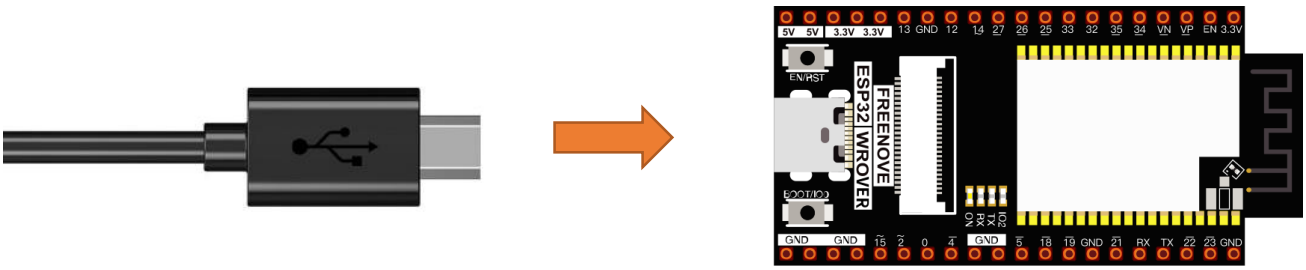
Station mode

When ESP32 selects Station mode, it acts as a WiFi client. It can connect to the router network and communicate with other devices on the router via WiFi connection. As shown below, the PC is connected to the router, and if ESP32 wants to communicate with the PC, it needs to be connected to the router.



Circuit

Connect Freenove ESP32 to the computer using the USB cable.

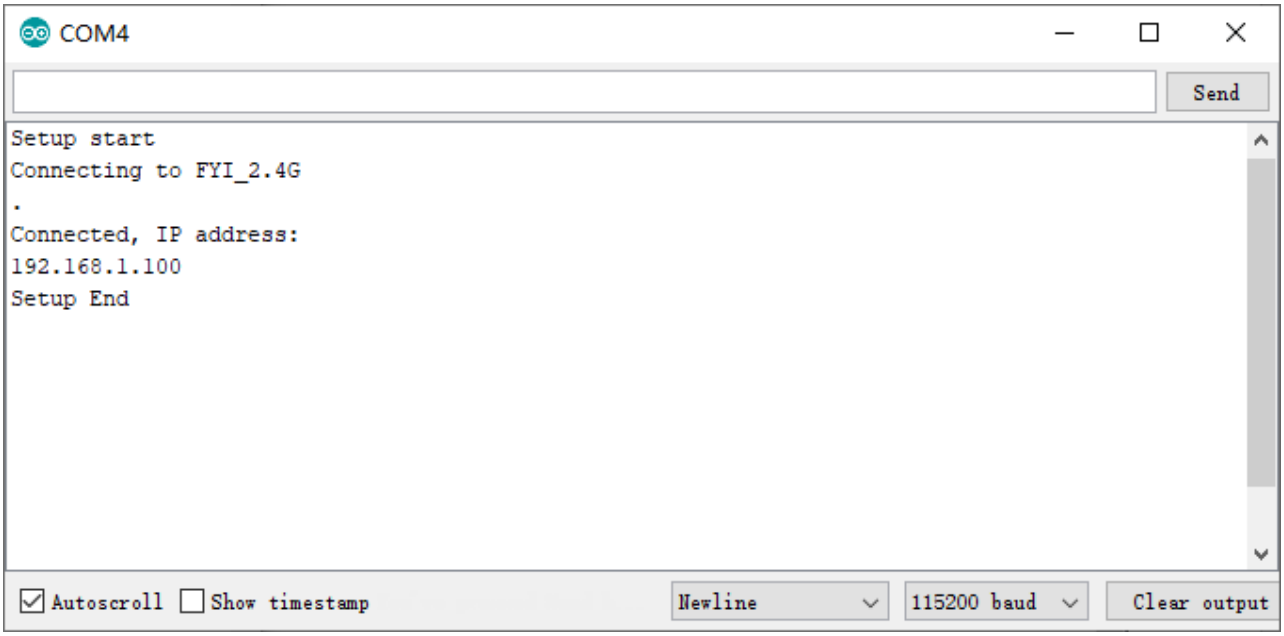


Sketch

Sketch_04.1_Station_mode

Because the names and passwords of routers in various places are different, before the Sketch runs, users need to enter the correct router's name and password in the box as shown in the illustration above.

After making sure the router name and password are entered correctly, compile and upload codes to ESP32-WROVER, open serial monitor and set baud rate to 115200. And then it will display as follows:



When ESP32-WROVER successfully connects to "ssid_Router", serial monitor will print out the IP address assigned to ESP32-WROVER by the router.

The following is the program code:

```
#include <WiFi.h>

const char *ssid_Router     = "********"; //Enter the router name
const char *password_Router = "********"; //Enter the router password

void setup(){
  Serial.begin(115200);
  delay(2000);
  Serial.println("Setup start");
  WiFi.begin(ssid_Router, password_Router);
  Serial.println(String("Connecting to ")+ssid_Router);
  while (WiFi.status() != WL_CONNECTED){
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nConnected, IP address: ");
  Serial.println(WiFi.localIP());
  Serial.println("Setup End");
}

void loop() {
}
```

Include the WiFi Library header file of ESP32.

```
#include <WiFi.h>
```

Enter correct router name and password.

```
3  const char *ssid_Router     = "********"; //Enter the router name
4  const char *password_Router = "********"; //Enter the router password
```

Set ESP32 in Station mode and connect it to your router.

```
10   WiFi.begin(ssid_Router, password_Router);
```

Check whether ESP32 has connected to router successfully every 0.5s.

```
12   while (WiFi.status() != WL_CONNECTED){
13     delay(500);
14     Serial.print(".");
15   }
```

Serial monitor prints out the IP address assigned to ESP32-WROVER

```
17   Serial.println(WiFi.localIP());
```

Reference

Class Station

Every time when using WiFi, you need to include header file "WiFi.h.".

begin(ssid, password,channel, bssid, connect): ESP32 is used as Station to connect hotspot.

ssid: WiFi hotspot name

password: WiFi hotspot password

channel: WiFi hotspot channel number; communicating through specified channel; optional parameter

bssid: mac address of WiFi hotspot, optional parameter

connect: blloean optional parameter, defaulting to true. If set as false, then ESP32 won't connect WiFi.

config(local_ip, gateway, subnet, dns1, dns2): set static local IP address.

local_ip: station fixed IP address.

subnet: subnet mask

dns1,dns2: optional parameter. define IP address of domain name server

status: obtain the connection status of WiFI

local IP(): obtian IP address in Station mode

disconnect(): disconnect wifi

setAutoConnect(boolen): set automatic connection Every time ESP32 is power on, it will connect WiFi aitomatically.

setAutoReconnect(boolen): set automatic reconnection Every time ESP32 disconnects WiFi, it will reconnect to WiFi automatically.

Project 4.2 AP mode

Component List & Circuit

Component List & Circuit are the same as in Section 30.1.

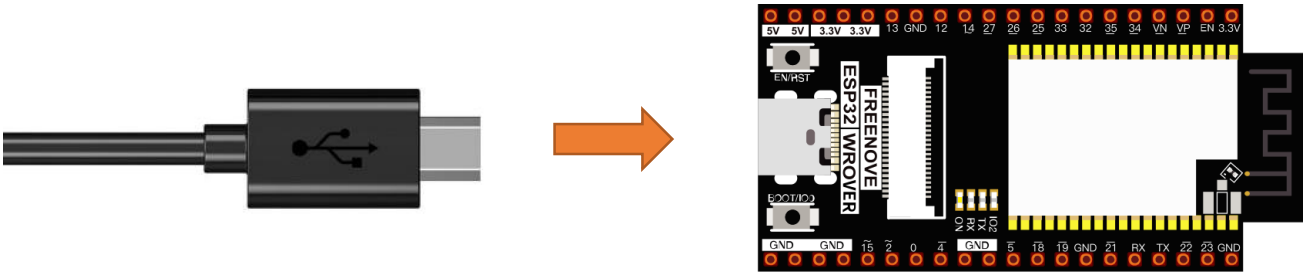
Component knowledge

AP mode

When ESP32 selects AP mode, it creates a hotspot network that is separate from the Internet and waits for other WiFi devices to connect. As shown in the figure below, ESP32 is used as a hotspot. If a mobile phone or PC wants to communicate with ESP32, it must be connected to the hotspot of ESP32. Only after a connection is established with ESP32 can they communicate.

Circuit

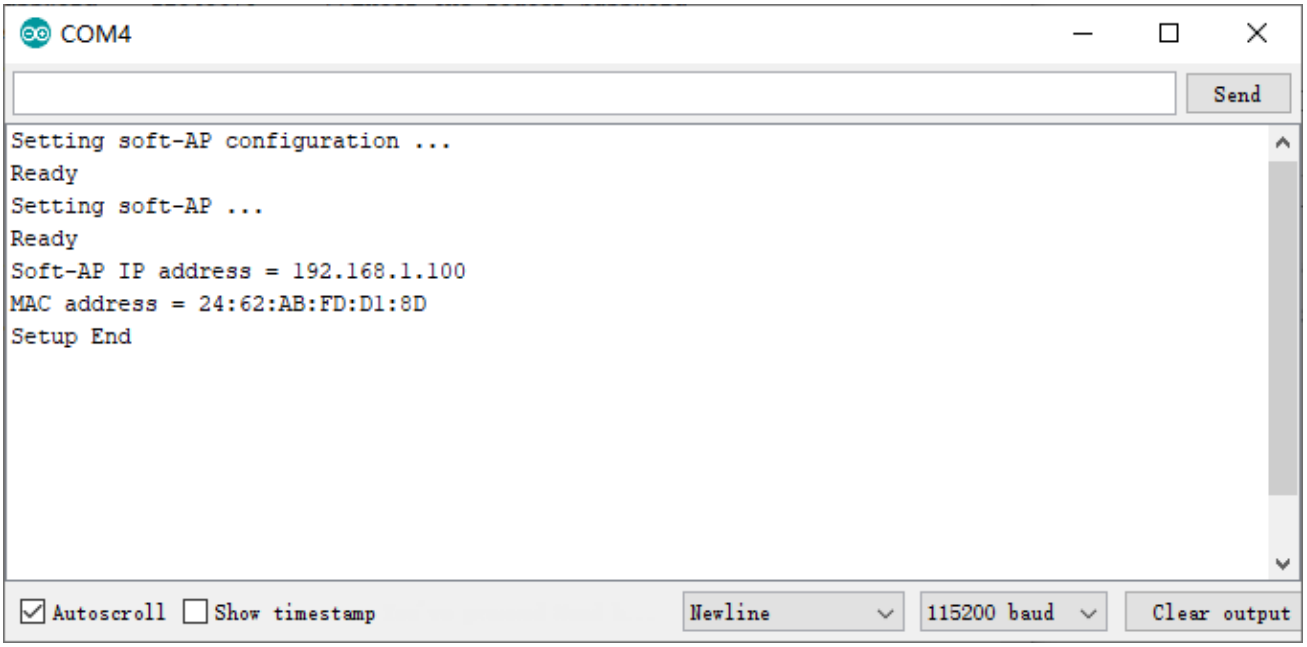
Connect Freenove ESP32 to the computer using the USB cable.



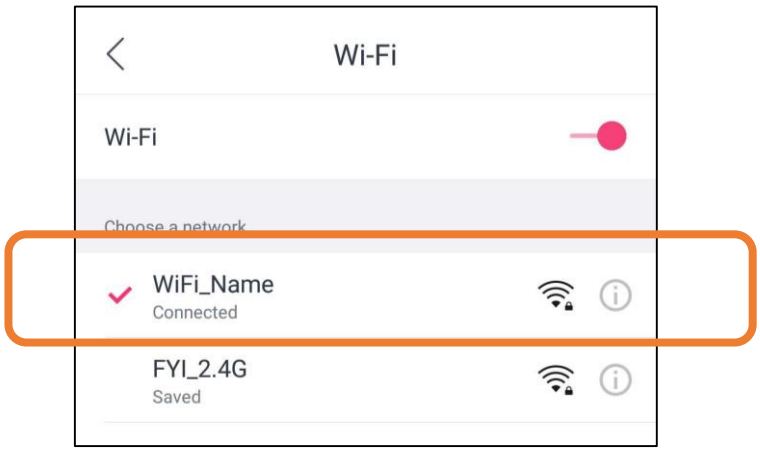
Sketch

Before the Sketch runs, you can make any changes to the AP name and password for ESP32 in the box as shown in the illustration above. Of course, you can leave it alone by default.

Compile and upload codes to ESP32-WROVER, open the serial monitor and set the baud rate to 115200. And then it will display as follows.



When observing the print information of the serial monitor, turn on the WiFi scanning function of your phone, and you can see the ssid_AP on ESP32, which is called "WiFi_Name" in this Sketch. You can enter the password "12345678" to connect it or change its AP name and password by modifying Sketch.



Sketch_04.2_AP_mode

The following is the program code:

```
#include <WiFi.h>

const char *ssid_AP     = "WiFi_Name"; //Enter the router name
const char *password_AP = "12345678"; //Enter the router password

IPAddress local_IP(192,168,1,100);//Set the IP address of ESP32 itself
IPAddress gateway(192,168,1,10);   //Set the gateway of ESP32 itself
IPAddress subnet(255,255,255,0);  //Set the subnet mask for ESP32 itself

void setup(){
  Serial.begin(115200);
  delay(2000);
  Serial.println("Setting soft-AP configuration ... ");
  WiFi.disconnect();
  WiFi.mode(WIFI_AP);
  Serial.println(WiFi.softAPConfig(local_IP, gateway, subnet) ? "Ready" : "Failed!");
  Serial.println("Setting soft-AP ... ");
  boolean result = WiFi.softAP(ssid_AP, password_AP);
  if(result){
    Serial.println("Ready");
    Serial.println(String("Soft-AP IP address = ") + WiFi.softAPIP().toString());
    Serial.println(String("MAC address = ") + WiFi.softAPmacAddress().c_str());
  }else{
    Serial.println("Failed!");
  }
  Serial.println("Setup End");
}

void loop() {
}
```

Include WiFi Library header file of ESP32.

```
#include <WiFi.h>
```

Enter correct AP name and password.

```
const char *ssid_AP     = "WiFi_Name"; //Enter the router name
const char *password_AP = "12345678"; //Enter the router password
```

Set ESP32 in AP mode.

```
WiFi.mode(WIFI_AP);
```

Configure IP address, gateway and subnet mask for ESP32.

```
WiFi.softAPConfig(local_IP, gateway, subnet)
```

Turn on an AP in ESP32, whose name is set by ssid_AP and password is set by password_AP.

```
WiFi.softAP(ssid_AP, password_AP);
```

Check whether the AP is turned on successfully. If yes, print out IP and MAC address of AP established by ESP32. If no, print out the failure prompt.

```
19  if(result){
20      Serial.println("Ready");
21      Serial.println(String("Soft-AP IP address = ") + WiFi.softAPIP().toString());
22      Serial.println(String("MAC address = ") + WiFi.softAPmacAddress().c_str());
23    }else{
24      Serial.println("Failed!");
25    }
26    Serial.println("Setup End");
```

Reference

Class AP

Every time when using WiFi, you need to include header file "WiFi.h.".

softAP(ssid, password, channel, ssid_hidden, max_connection):

ssid: WiFi hotspot name

password: WiFi hotspot password

channel: Number of WiFi connection channels, range 1-13. The default is 1.

ssid_hidden: Whether to hide WiFi name from scanning by other devices. The default is not hide.

max_connection: Maximum number of WiFi connected devices. The range is 1-4. The default is 4.

softAPConfig(local_ip, gateway, subnet): set static local IP address.

local_ip: station fixed IP address.

Gateway: gateway IP address

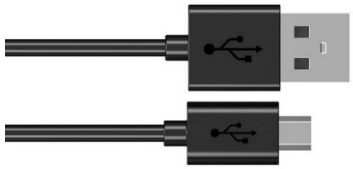
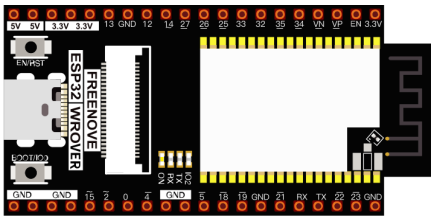
subnet: subnet mask

softAP(): obtian IP address in AP mode

softAPdisconnect (): disconnect AP mode.

Project 3.3 AP+Station mode

Component List

Micro USB Wire x1 	ESP32-WROVER x1 

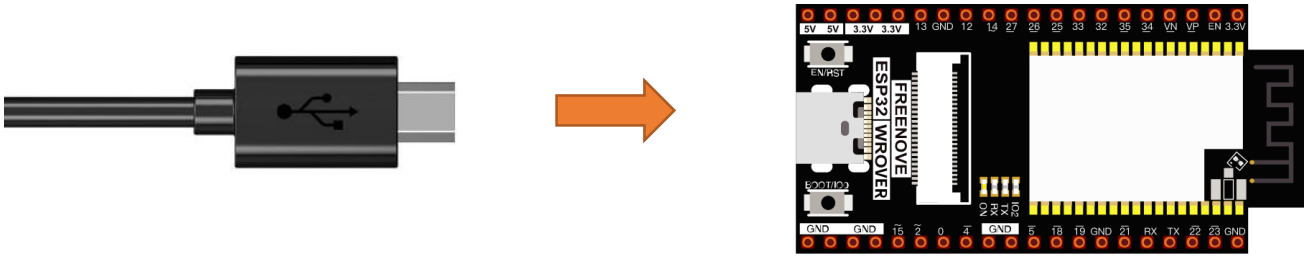
Component knowledge

AP+Station mode

In addition to AP mode and station mode, ESP32 can also use AP mode and station mode at the same time. This mode contains the functions of the previous two modes. Turn on ESP32's station mode, connect it to the router network, and it can communicate with the Internet via the router. At the same time, turn on its AP mode to create a hotspot network. Other WiFi devices can choose to connect to the router network or the hotspot network to communicate with ESP32.

Circuit

Connect Freenove ESP32 to the computer using the USB cable.

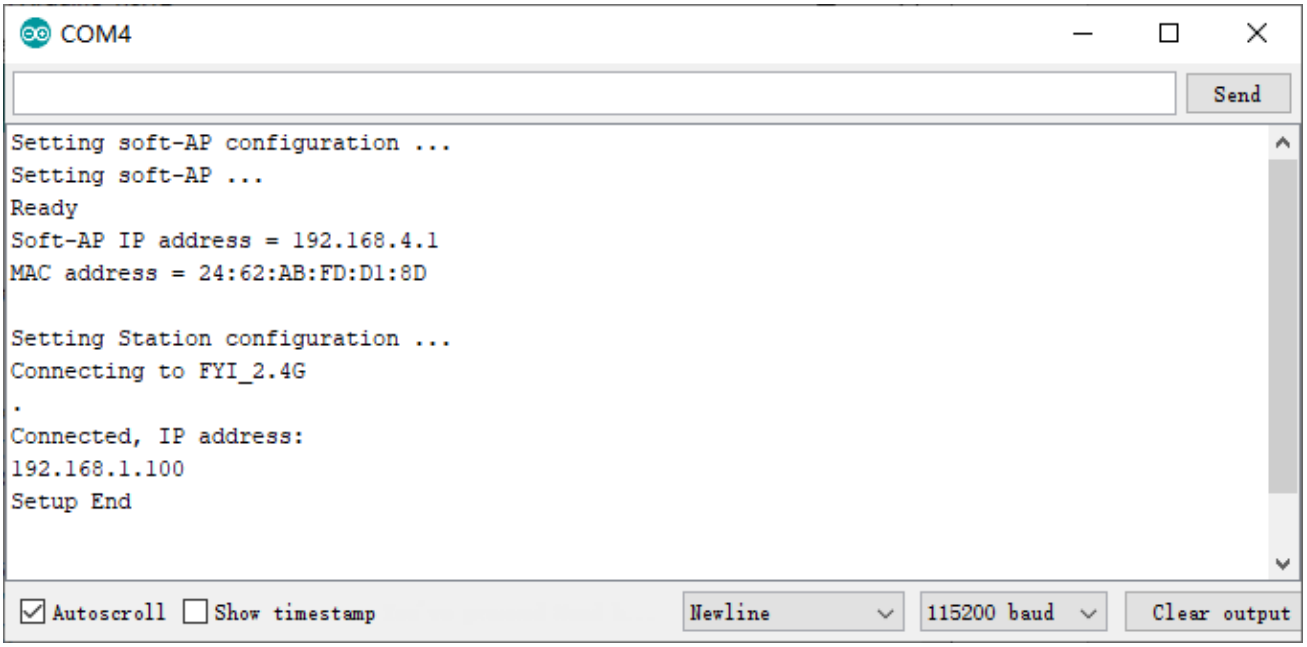


Sketch

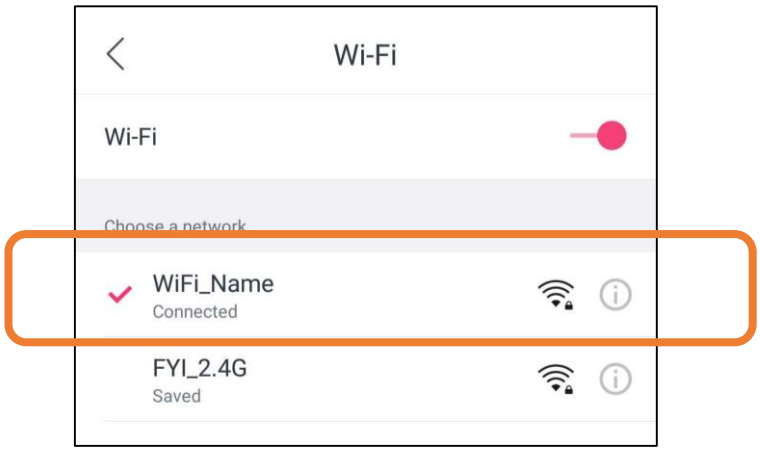
Sketch_04.3_AP_Station_mode

It is analogous to Project 4.1 and Project 4.2. Before running the Sketch, you need to modify ssid_Router, password_Router, ssid_AP and password_AP shown in the box of the illustration above.

After making sure that Sketch is modified correctly, compile and upload codes to ESP32-WROVER, open serial monitor and set baud rate to 115200. And then it will display as follows:



When observing the print information of the serial monitor, turn on the WiFi scanning function of your phone, and you can see the ssid_AP on ESP32.



The following is the program code:

```
#include <WiFi.h>

const char *ssid_Router     = "********";  //Enter the router name
const char *password_Router = "********";  //Enter the router password
const char *ssid_AP         = "WiFi_Name"; //Enter the AP name
const char *password_AP     = "12345678";  //Enter the AP password

void setup(){
  Serial.begin(115200);
  Serial.println("Setting soft-AP configuration ... ");
  WiFi.disconnect();
```

```
12    WiFi.mode(WIFI_AP);
13    Serial.println("Setting soft-AP ... ");
14    boolean result = WiFi.softAP(ssid_AP, password_AP);
15    if(result){
16      Serial.println("Ready");
17      Serial.println(String("Soft-AP IP address = ") + WiFi.softAPIP().toString());
18      Serial.println(String("MAC address = ") + WiFi.softAPmacAddress().c_str());
19    }else{
20      Serial.println("Failed!");
21    }
22
23    Serial.println("\nSetting Station configuration ... ");
24    WiFi.begin(ssid_Router, password_Router);
25    Serial.println(String("Connecting to ")+ ssid_Router);
26    while (WiFi.status() != WL_CONNECTED){
27      delay(500);
28      Serial.print(".");
29    }
30    Serial.println("\nConnected, IP address: ");
31    Serial.println(WiFi.localIP());
32    Serial.println("Setup End");
33  }
34
35  void loop() {
36  }
```

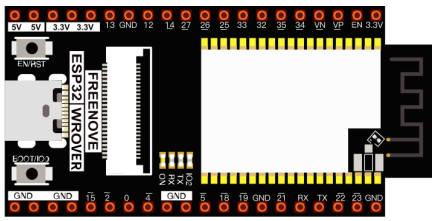
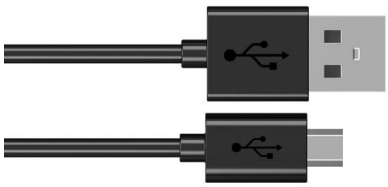
Chapter 5 TCP/IP

In this chapter, we will introduce how ESP32 implements network communications based on TCP/IP protocol. There are two roles in TCP/IP communication, namely Server and Client, which will be implemented respectively with two projects in this chapter.

Project 5.1 As Client

In this section, ESP32 is used as Client to connect Server on the same LAN and communicate with it.

Component List

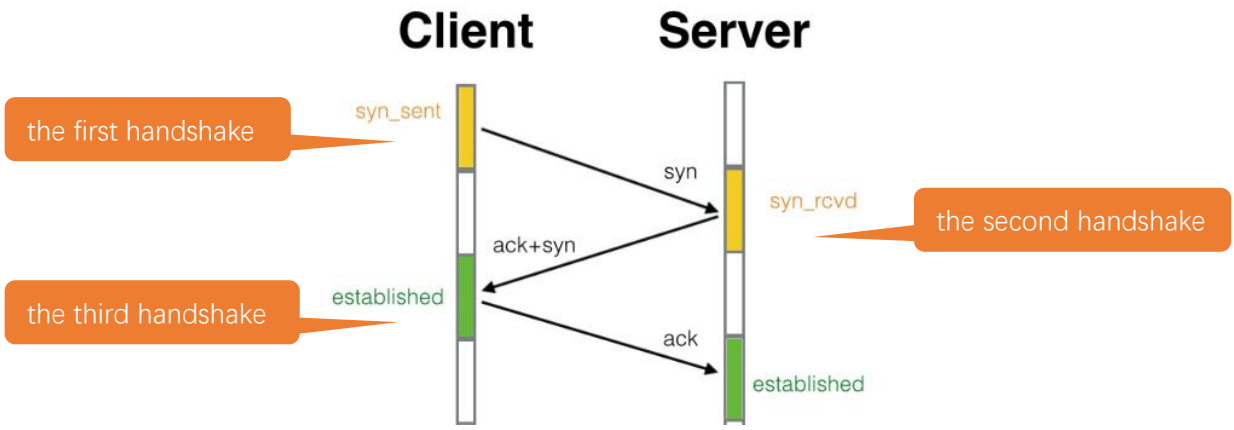
ESP32-WROVER x1 	Micro USB Wire x1 

Component knowledge

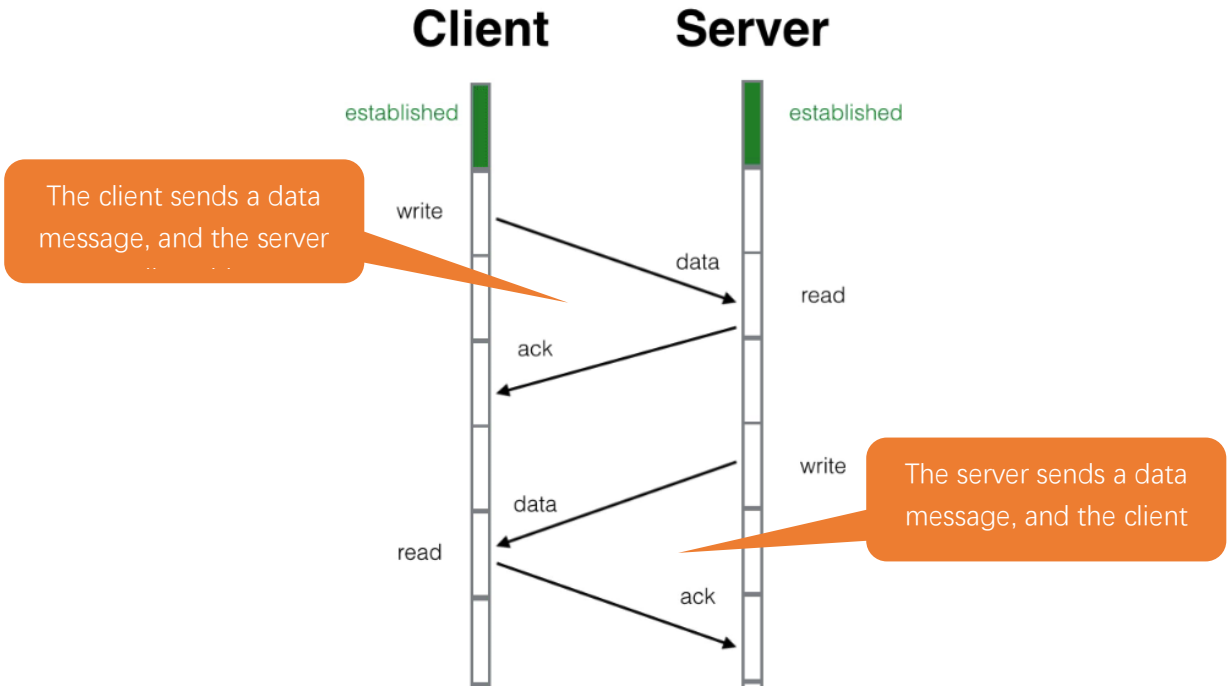
TCP connection

Before transmitting data, TCP needs to establish a logical connection between the sending end and the receiving end. It provides reliable and error-free data transmission between the two computers. In the TCP connection, the client and the server must be clarified. The client sends a connection request to the server, and each time such a request is proposed, a "three-times handshake" is required.

Three-times handshake: In the TCP protocol, during the preparation phase of sending data, the client and the server interact three times to ensure the reliability of the connection, which is called "three-times handshake".
The first handshake, the client sends a connection request to the server and waits for the server to confirm.
The second handshake, the server sends a response back to the client informing that it has received the connection request.
The third handshake, the client sends a confirmation message to the server again to confirm the connection.



TCP is a connection-oriented, low-level transmission control protocol. After TCP establishes a connection, the client and server can send and receive messages to each other, and the connection will always exist as long as the client or server does not initiate disconnection. Each time one party sends a message, the other party will reply with an ack signal.



Install Processing

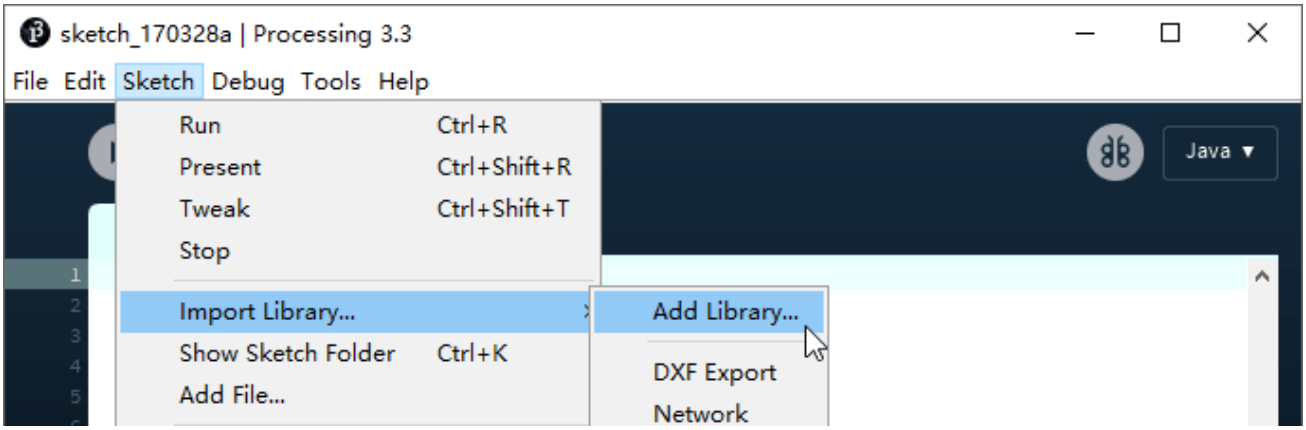
In this tutorial, we use Processing to build a simple TCP/IP communication platform.

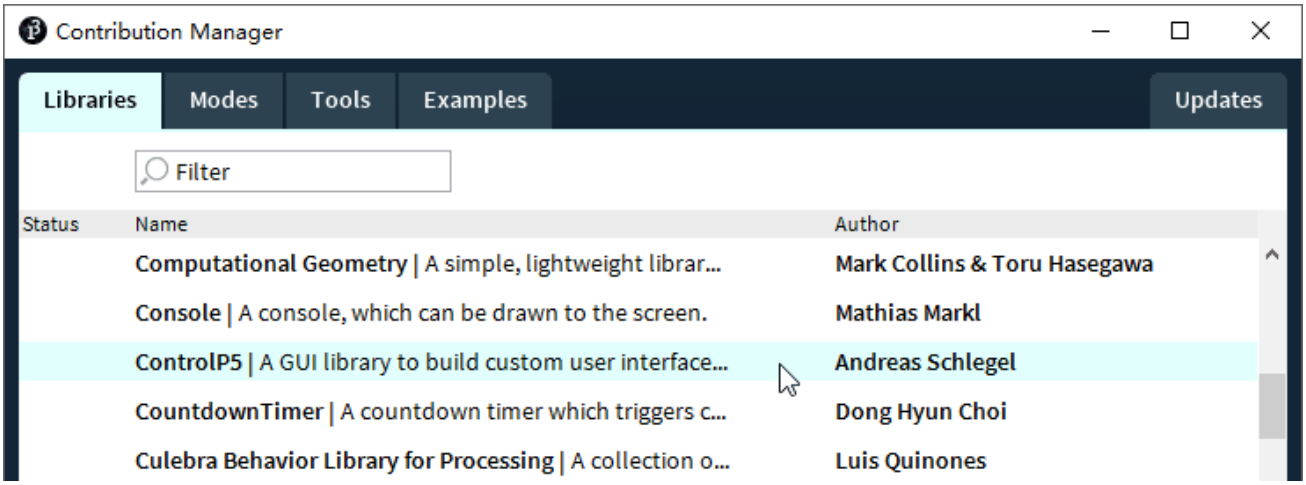
If you've not installed Processing, you can download it by clicking https://processing.org/download/. You can choose an appropriate version to download according to your PC system.

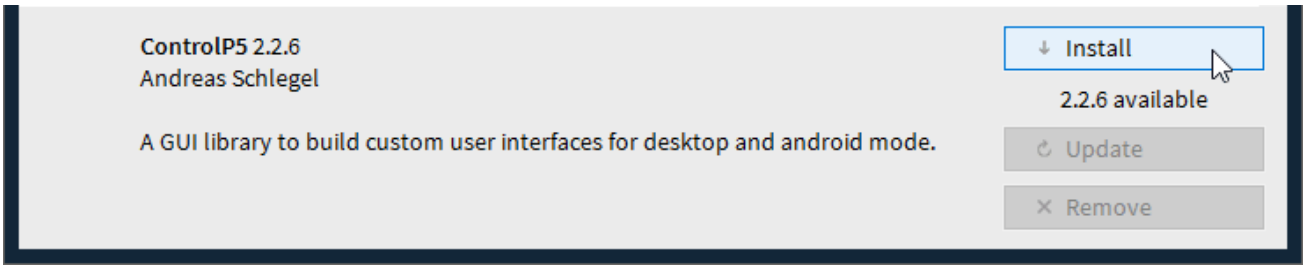
Unzip the downloaded file to your computer. Click "processing.exe" as the figure below to run this software.

Use Server mode for communication

Install ControlP5.





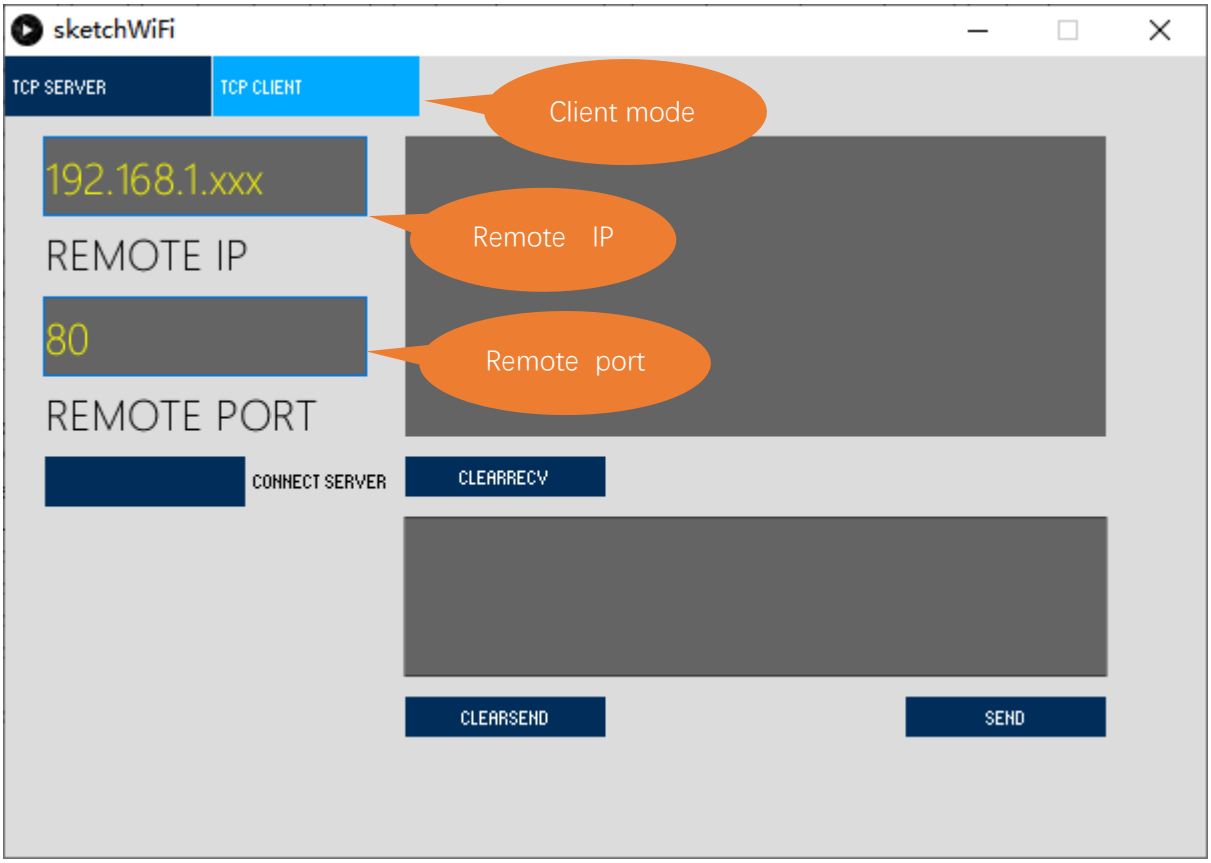


Open the "**Freenove_ESP32_WROVER_Board\Sketches\Sketches\Sketch_05.1_WiFiClient\sketchWiFi\sketchWiFi.pde**", and click "Run".

The new pop-up interface is as follows. If ESP32 is used as client, select TCP SERVER mode for sketchWiFi.

When sketchWiFi selects TCP SERVER mode, ESP32 Sketch needs to be changed according to sketchWiFi's displaying of LOCAL IP or LOCAL PORT.

If ESP32 serves as server, select TCP CLIENT mode for sketchWiFi.



When sketchWiFi selects TCP CLIENT mode, the LOCAL IP and LOCAL PORT of sketchWiFi need to be changed according to the IP address and port number printed by the serial monitor.

Mode selection: select **Server mode/Client mode**.

IP address: In server mode, this option does not need to be filled in, and the computer will automatically obtain the IP address.

In client mode, fill in the remote IP address to be connected.

Port number: In server mode, fill in a port number for client devices to make an access connection.

In client mode, fill in port number given by the Server devices to make an access connection.

Start button: In server mode, push the button, then the computer will serve as server and open a port number for client to make access connection. During this period, the computer will keep monitoring.

In client mode, before pushing the button, please make sure the server is on, remote IP address and remote port number is correct; push the button, and the computer will make access connection to the remote port number of the remote IP as a client.

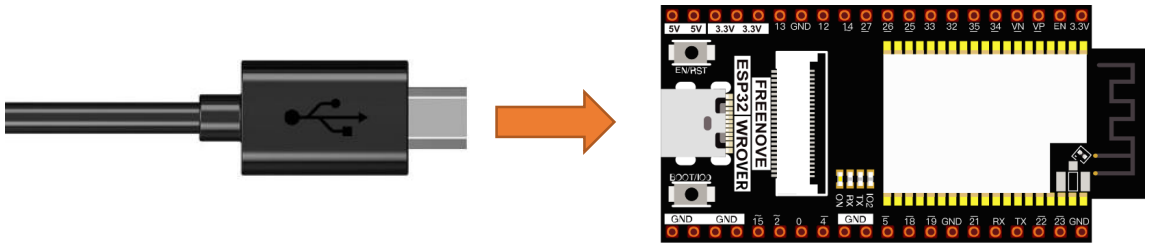
clear receive: clear out the content in the receiving text box

clear send: clear out the content in the sending text box

Sending button: push the sending button, the computer will send the content in the text box to others.

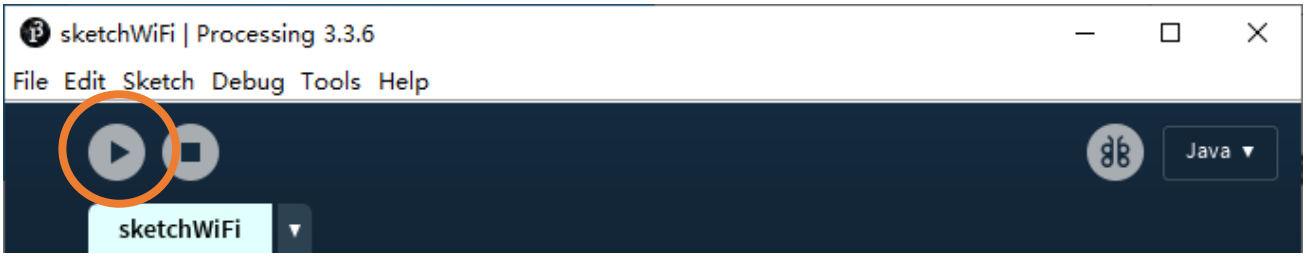
Circuit

Connect Freenove ESP32 to the computer using USB cable.

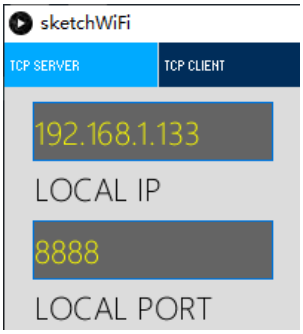


Sketch

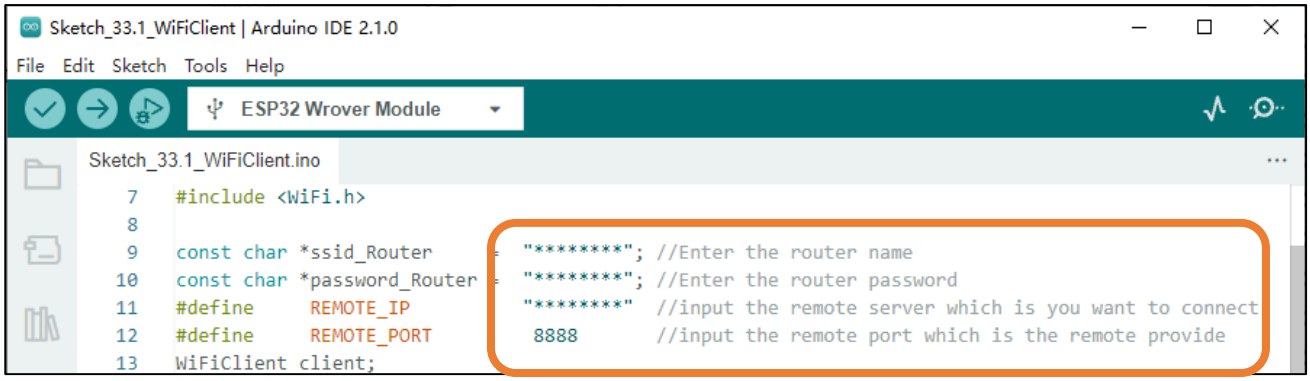
Before running the Sketch, please open “sketchWiFi.pde.” first, and click “Run”.



The newly pop up window will use the computer’s IP address by default and open a data monitor port.

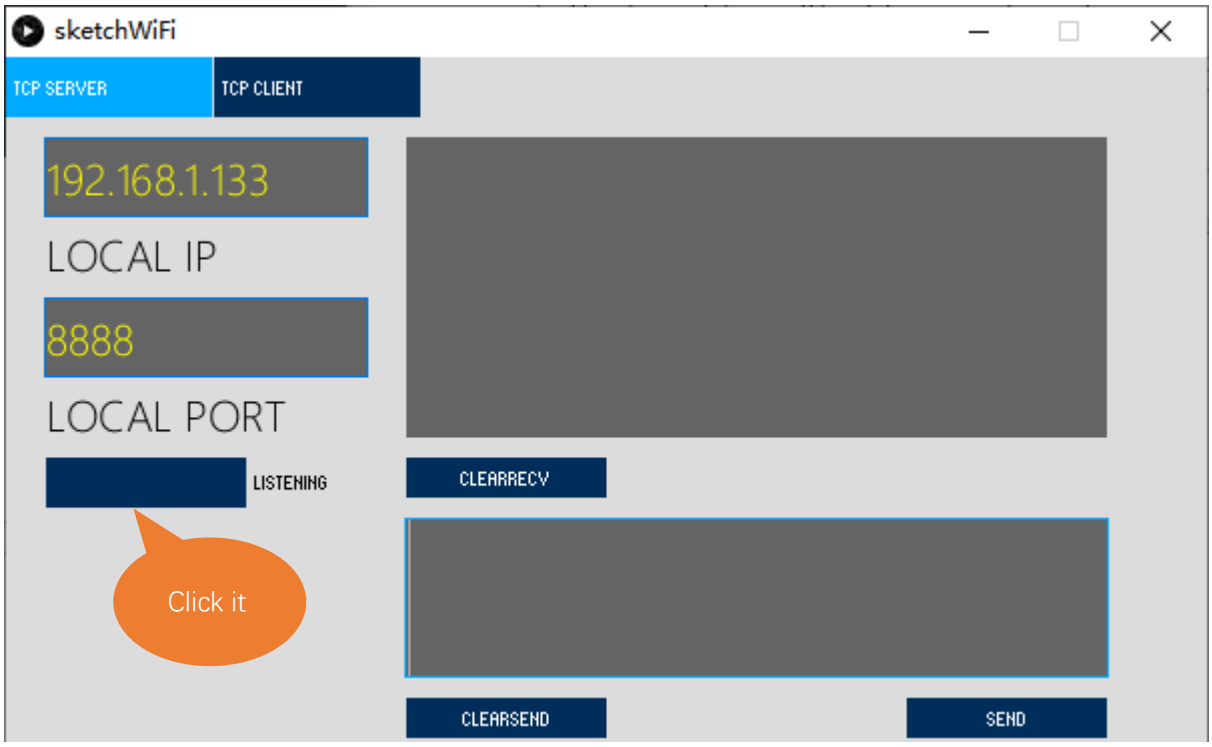


Next, open Sketch_05.1_WiFiClient.ino. Before running it, please change the following information based on "LOCAL IP" and "LOCAL PORT" in the figure above.

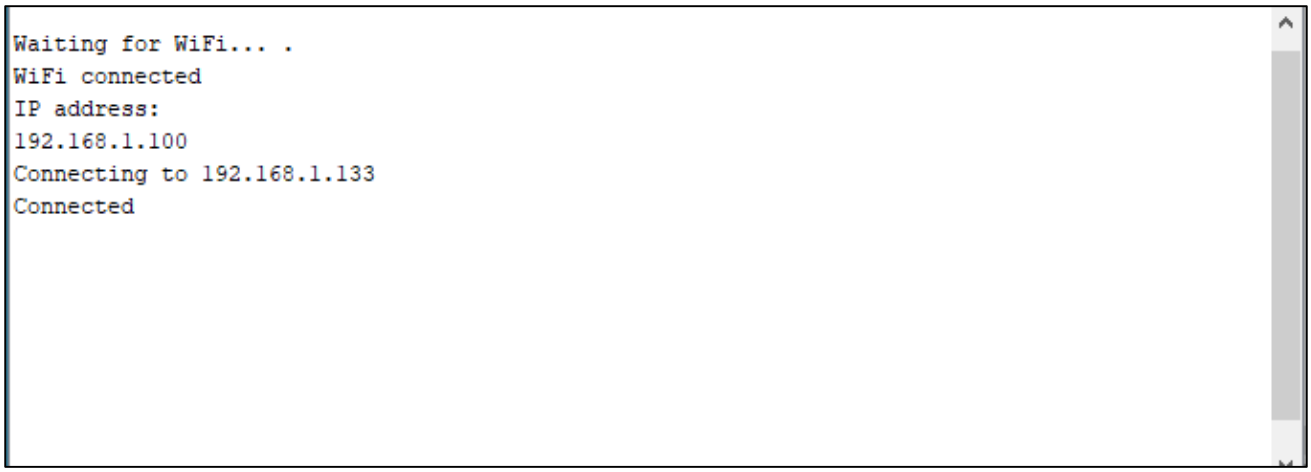


REMOTE_IP needs to be filled in according to the interface of sketchWiFi.pde. Taking this tutorial as an example, its REMOTE_IP is “192.168.1.133”. Generally, by default, the ports do not need to change its value.

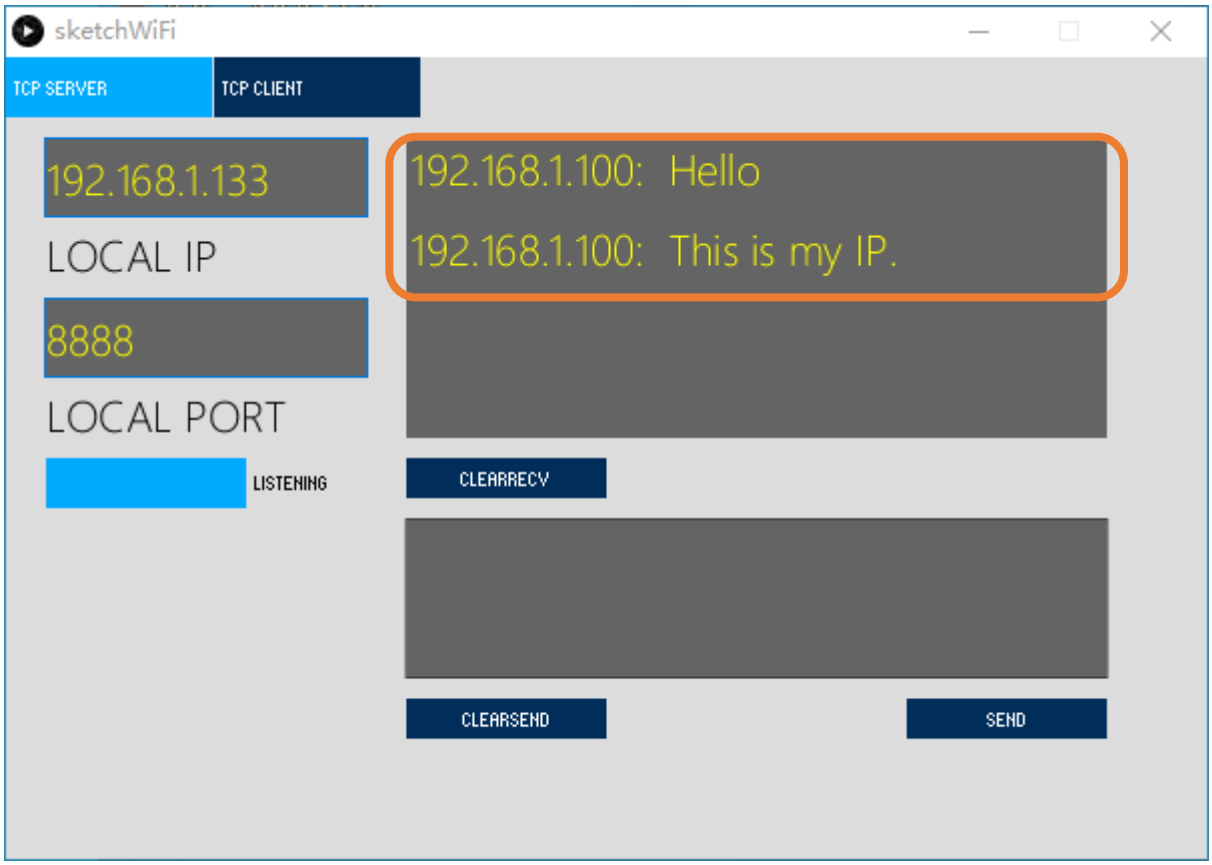
Click LISTENING, turn on TCP SERVER's data listening function and wait for ESP32 to connect.



Compile and upload code to ESP32-WROVER, open the serial monitor and set the baud rate to 115200. ESP32 connects router, obtains IP address and sends access request to server IP address on the same LAN till the connection is successful. When connect successfully, ESP32 can send messages to server.



ESP32 connects with TCP SERVER, and TCP SERVER receives messages from ESP32, as shown in the figure below.



Sketch_05.1_As_Client

The following is the program code:

```
#include <WiFi.h>

const char *ssid_Router     =  "********"; //Enter the router name
const char *password_Router =  "********"; //Enter the router password
#define     REMOTE_IP       "********"  //input the remote server which is you want to connect
#define     REMOTE_PORT      8888       //input the remote port which is the remote provide
WiFiClient client;

void setup() {
  Serial.begin(115200);
  delay(10);

  WiFi.begin(ssid_Router, password_Router);
  Serial.print("\nWaiting for WiFi... ");
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  delay(500);

  Serial.print("Connecting to ");
  Serial.println(REMOTE_IP);

  while (!client.connect(REMOTE_IP, REMOTE_PORT)) {
    Serial.println("Connection failed.");
    Serial.println("Waiting a moment before retrying...");
  }
  Serial.println("Connected");
  client.print("Hello\n");
  client.print("This is my IP.\n");
}

void loop() {
  if (client.available() > 0) {
    delay(20);
    //read back one line from the server
    String line = client.readString();
    Serial.println(REMOTE_IP + String(":") + line);
```

```
42    }
43    if (Serial.available() > 0) {
44      delay(20);
45      String line = Serial.readString();
46      client.print(line);
47    }
48    if (client.connected () == 0) {
49      client.stop();
50      WiFi.disconnect();
51    }
52  }
```

Add WiFi function header file.

```
1   #include <WiFi.h>
```

Enter the actual router name, password, remote server IP address, and port number.

```
3   const char *ssid_Router     =  "********"; //Enter the router name
4   const char *password_Router =  "********"; //Enter the router password
5   #define     REMOTE_IP       "********"  //input the remote server which is you want to connect
6   #define     REMOTE_PORT      8888       //input the remote port which is the remote provide
```

Apply for the method class of WiFiClient.

```
7   WiFiClient client;
```

Connect specified WiFi until it is successful. If the name and password of WiFi are correct but it still fails to connect, please push the reset key.

```
13    WiFi.begin(ssid_Router, password_Router);
14    Serial.print("\nWaiting for WiFi... ");
15    while (WiFi.status() ! = WL_CONNECTED) {
16      Serial.print(".");
17      delay(500);
18    }
```

Send connection request to remote server until connect successfully. When connect successfully, print out the connecting prompt on the serial monitor and send messages to remote server.

```
28    while (!client.connect(REMOTE_IP, REMOTE_PORT)) {//Connect to Server
29      Serial.println("Connection failed.");
30      Serial.println("Waiting a moment before retrying...");
31    }
32    Serial.println("Connected");
33    client.print("Hello\n");
```

When ESP32 receive messages from servers, it will print them out via serial port; Users can also send messages to servers from serial port.

```
37  if (client.available() > 0) {
38      delay(20);
39      //read back one line from the server
40      String line = client.readString();
41      Serial.println(REMOTE_IP + String(":") + line);
42    }
```

```
43    if (Serial.available() > 0) {
44      delay(20);
45      String line = Serial.readString();
46      client.print(line);
47    }
```

If the server is disconnected, turn off WiFi of ESP32.

```
48    if (client.connected () == false) {
49      client.stop();
50      WiFi.disconnect();
51    }
```

Reference

Class Client

Every time when using Client, you need to include header file "WiFi.h."

connect(ip, port, timeout)/connect(*host, port, timeout): establish a TCP connection.

ip, *host: ip address of target server

port: port number of target server

timeout: connection timeout

connected(): judge whether client is connecting. If return value is 1, then connect successfully; If return value is 0, then fail to connect.

stop(): stop tcp connection

print(): send data to server connecting to client

available(): return to the number of bytes readable in receive buffer, if no, return to 0 or -1.

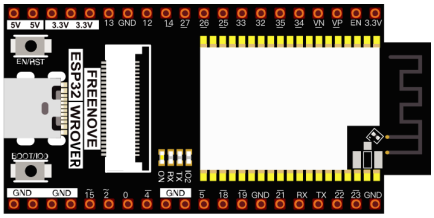
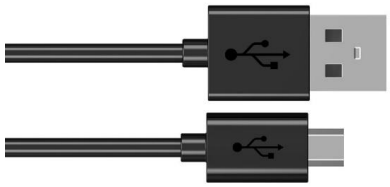
read(): read one byte of data in receive buffer

readString(): read string in receive buffer

Project 5.2 As Server

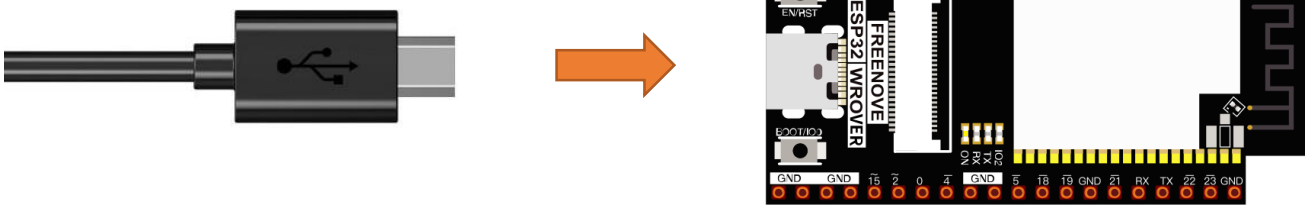
In this section, ESP32 is used as a server to wait for the connection and communication of client on the same LAN.

Component List

ESP32-WROVER x1 	Micro USB Wire x1 

Circuit

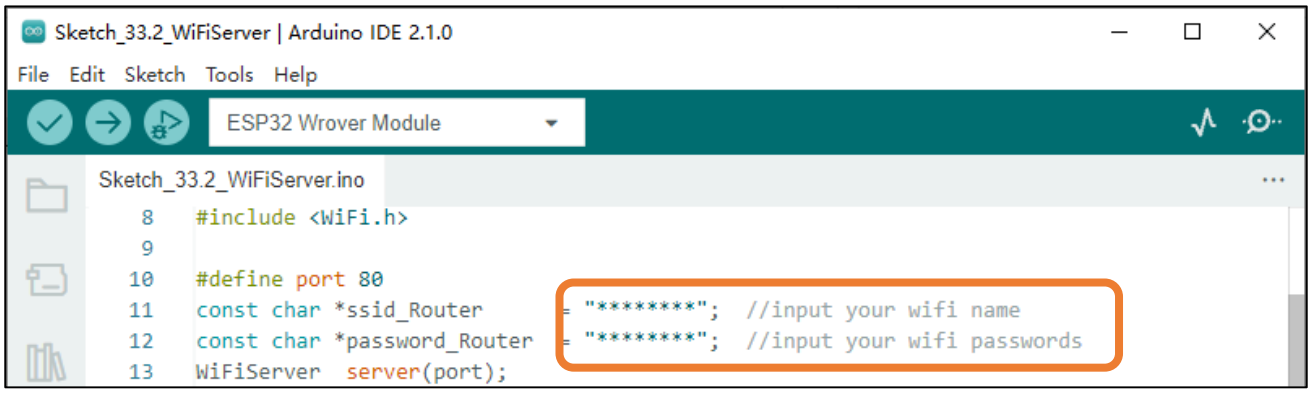
Connect Freenove ESP32 to the computer using a USB cable.



Sketch

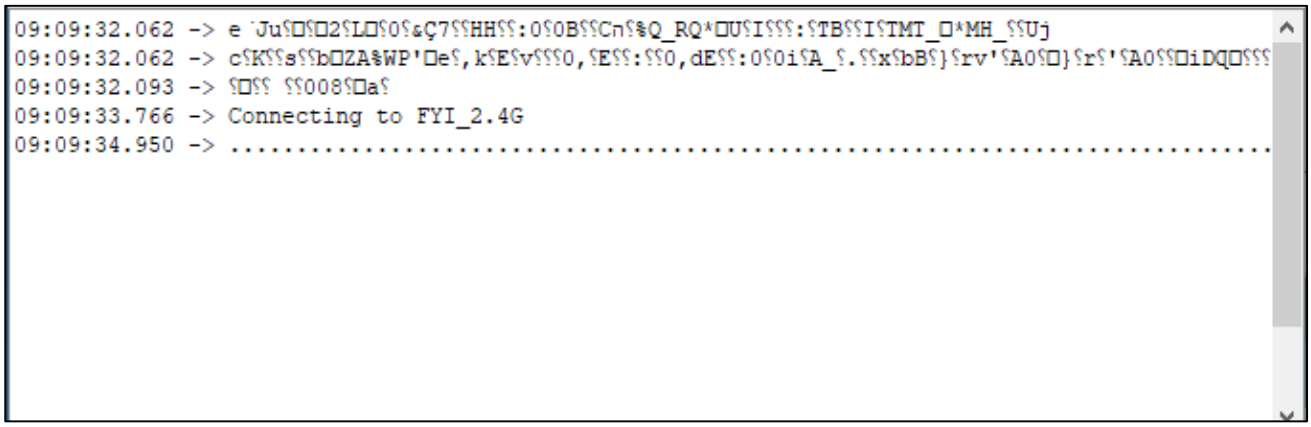
Before running Sketch, please modify the contents of the box below first.

Sketch_05.2_As_Server

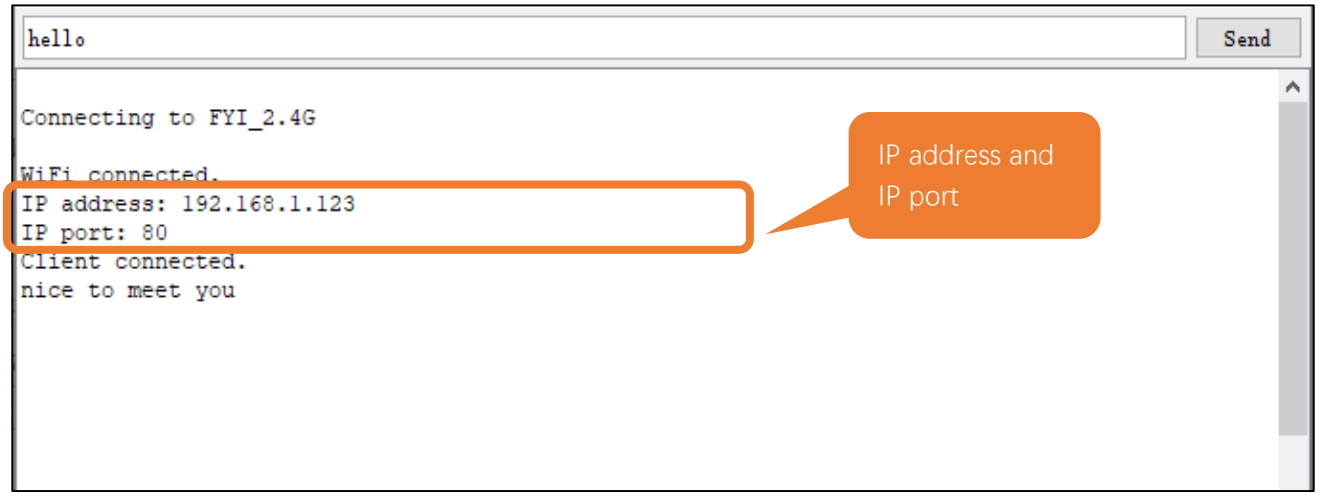


Compile and upload code to ESP32-WROVER board, open the serial monitor and set the baud rate to 115200. Turn on server mode for ESP32, waiting for the connection of other devices on the same LAN. Once a device connects to server successfully, they can send messages to each other.

If the ESP32 fails to connect to router, press the reset button as shown below and wait for ESP32 to run again.



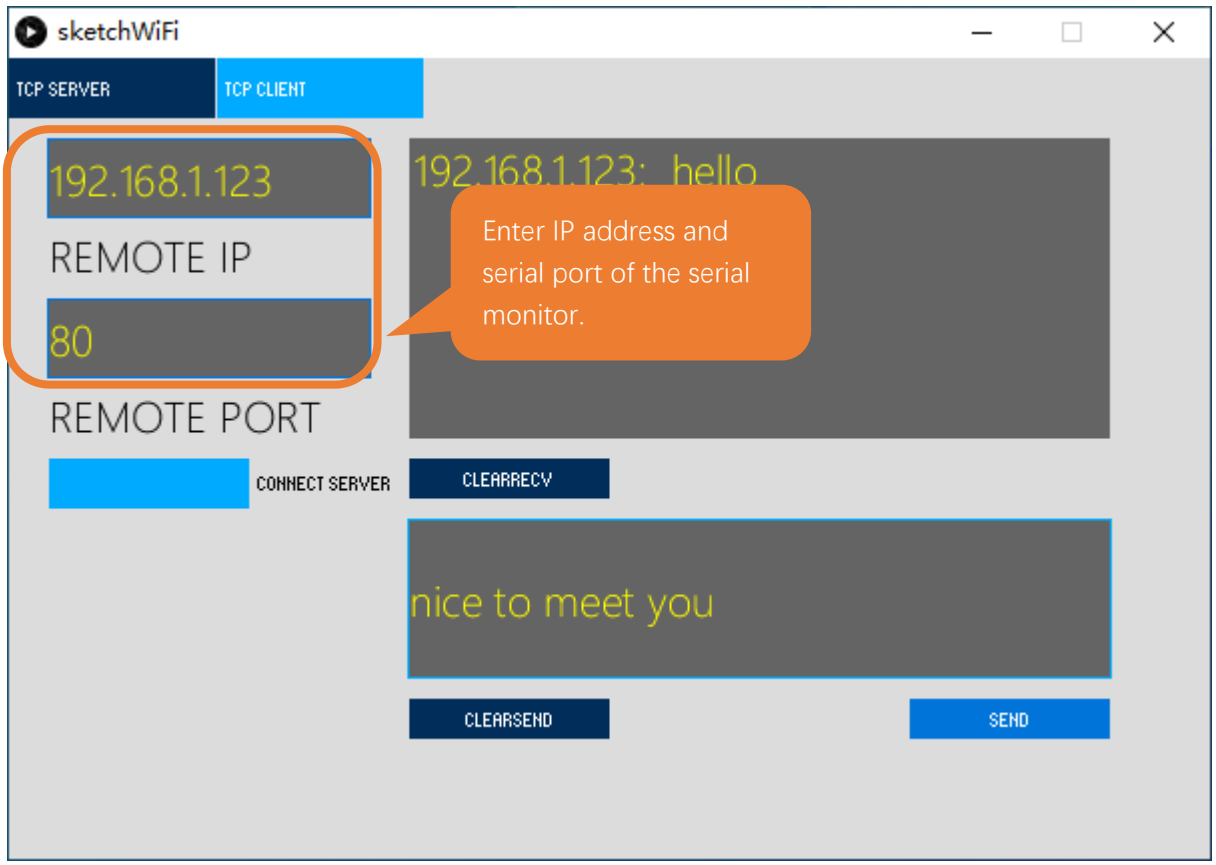
Serial Monitor



Processing:

Open the "**Freenove_ESP32_WROVER_Board\Sketches\Sketches\Sketch_05.2_WiFiServer\sketchWiFi\sketchWiFi.pde**".

Based on the messages printed by the serial monitor, enter correct IP address and serial port in Processing to establish connection and make communication.



The following is the program code:

```
#include <WiFi.h>

#define port 80
const char *ssid_Router      = "********";  //input your wifi name
const char *password_Router  = "********";  //input your wifi passwords
WiFiServer  server(port);

void setup()
{
    Serial.begin(115200);
    Serial.printf("\nConnecting to ");
    Serial.println(ssid_Router);
    WiFi.disconnect();
    WiFi.begin(ssid_Router, password_Router);
    delay(1000);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("");
    Serial.println("WiFi connected.");
    Serial.print("IP address: ");
    Serial.println(WiFi.localIP());
    Serial.printf("IP port: %d\n",port);
    server.begin(port);
    WiFi.setAutoReconnect(true);
}

void loop(){
 WiFiClient client = server.accept();            // listen for incoming clients
  if (client) {                                  // if you get a client
    Serial.println("Client connected.");
    while (client.connected()) {                 // loop while the client's connected
      if (client.available()) {                  // if there's bytes to read from the
client
        Serial.println(client.readStringUntil('\n')); // print it out the serial monitor
        while(client.read()>0);                  // clear the wifi receive area cache
      }
      if(Serial.available()){                    // if there's bytes to read from the
serial monitor
        client.print(Serial.readStringUntil('\n')); // print it out the client.
        while(Serial.read()>0);                  // clear the wifi receive area cache
      }
```

```
42      }
43      client.stop();                                  // stop the client connecting.
44      Serial.println("Client Disconnected.");
45    }
46  }
```

Apply for method class of WiFiServer.

```
6   WiFiServer server(port);          //Apply for a Server object whose port number is 80
```

Connect specified WiFi until it is successful. If the name and password of WiFi are correct but it still fails to connect, please push the reset key.

```
13      WiFi.disconnect();
14      WiFi.begin(ssid_Router, password_Router);
15      delay(1000);
16      while (WiFi.status() != WL_CONNECTED) {
17          delay(500);
18          Serial.print(".");
19      }
20      Serial.println("");
21      Serial.println("WiFi connected.");
```

Print out the IP address and port number of ESP32.

```
22      Serial.print("IP address: ");
23      Serial.println(WiFi.localIP());                 //print out IP address of ESP32
24      Serial.printf("IP port: %d\n",port);            //Print out ESP32's port number
```

Turn on server mode of ESP32, turn on automatic reconnection.

```
25      server.begin();                                 //Turn ON ESP32 as Server mode
26      WiFi.setAutoReconnect(true);
```

When ESP32 receive messages from servers, it will print them out via serial port; Users can also send messages to servers from serial port.

```
34      if (client.available()) {                       // if there's bytes to read from the
    client
35        Serial.println(client.readStringUntil('\n')); // print it out the serial monitor
36        while(client.read()>0);                       // clear the wifi receive area cache
37      }
38      if(Serial.available()){                         // if there's bytes to read from the
    serial monitor
39        client.print(Serial.readStringUntil('\n')); // print it out the client.
40        while(Serial.read()>0);                       // clear the wifi receive area cache
41      }
```

Reference

Class Server

Every time use Server functionality, we need to include header file"WiFi.h".

WiFiServer(uint16_t port=80, uint8_t max_clients=4): create a TCP Server.

port: ports of Server; range from 0 to 65535 with the default number as 80.

max_clients: maximum number of clients with default number as 4.

begin(port): start the TCP Server.

port: ports of Server; range from 0 to 65535 with the default number as 0.

setNoDelay(bool nodelay): whether to turn off the delay sending functionality.

nodelay: true stands for forbidden Nagle algorithm.

close(): close tcp connection.

stop(): stop tcp connection.

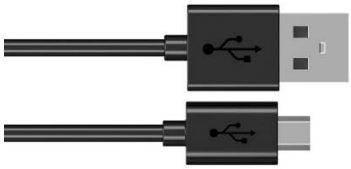
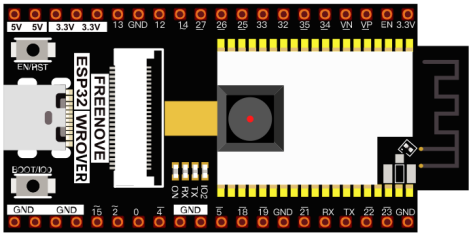
Chapter 6 Camera Web Server

In this section, we'll use ESP32's video function as an example to study.

Project 6.1 Camera Web Server

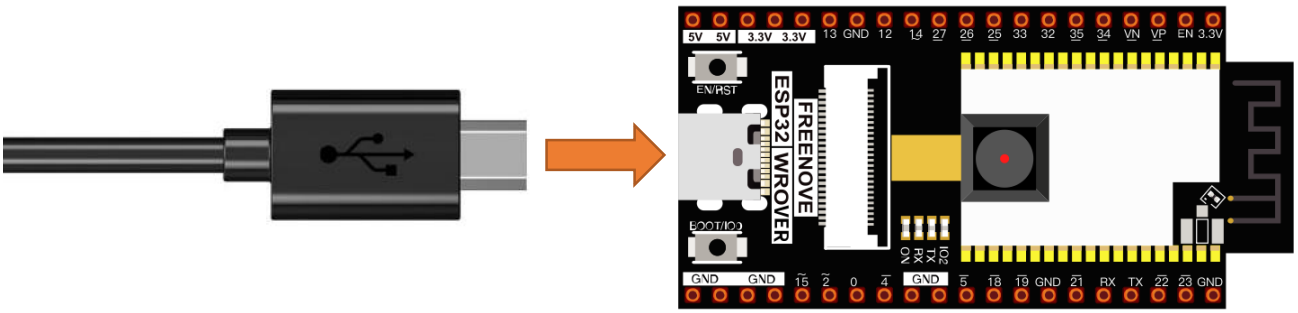
Connect ESP32 using USB and check its IP address through serial monitor. Use web page to access IP address to obtain video and image data.

Component List

Micro USB Wire x1 	ESP32-WROVER x1 

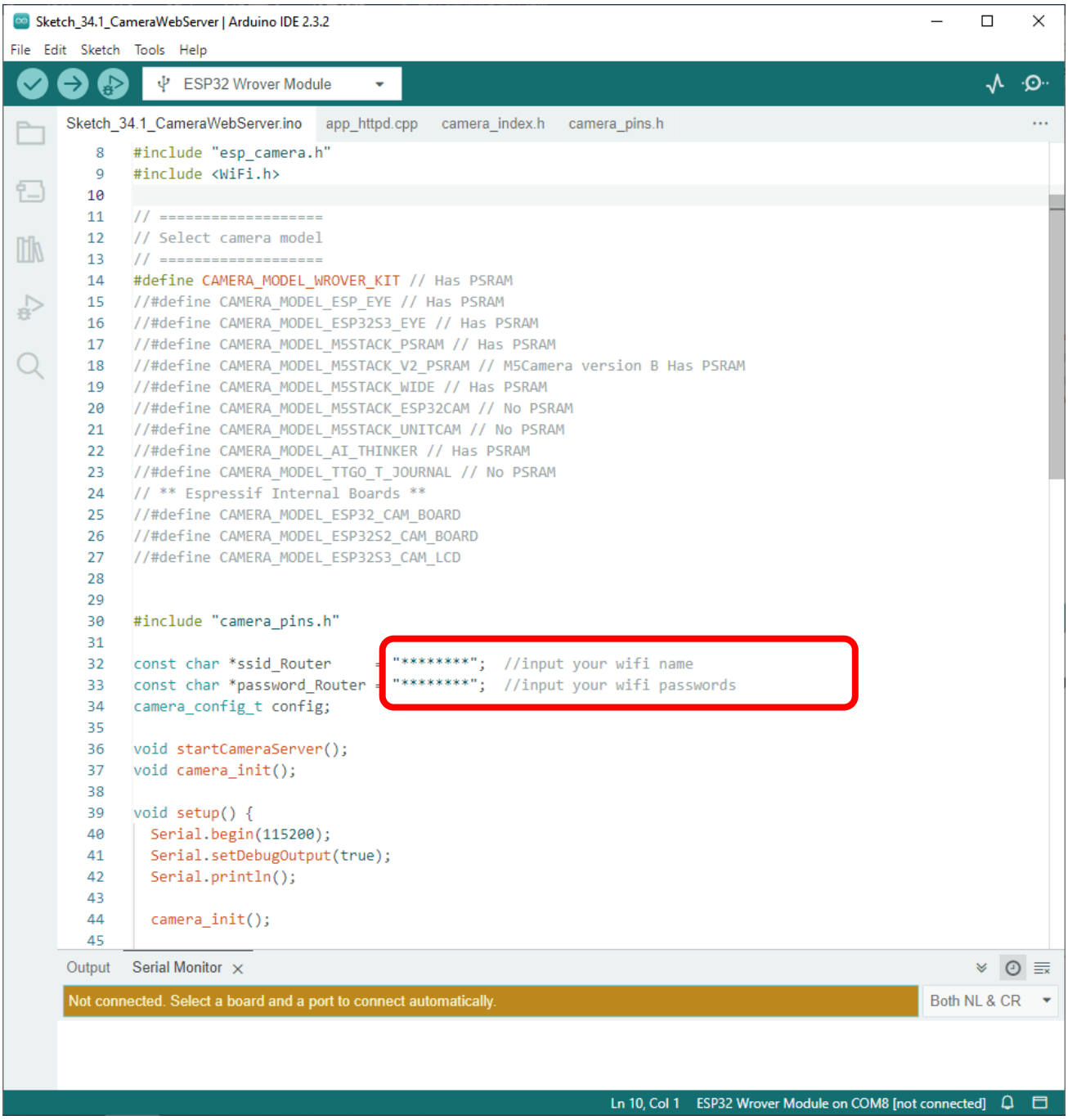
Circuit

Connect Freenove ESP32 to the computer using USB cable.



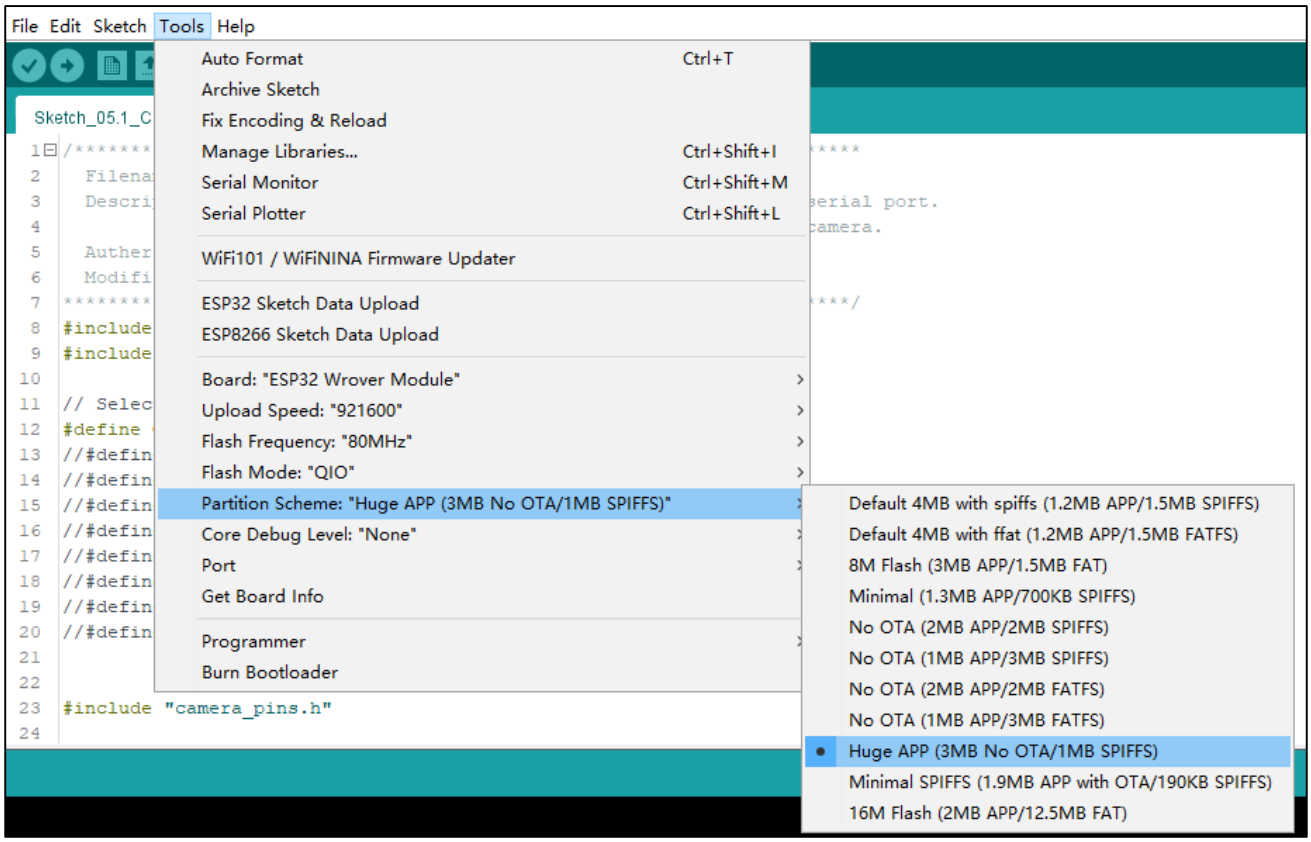
Sketch

Sketch_06.1_As_CameraWebServer

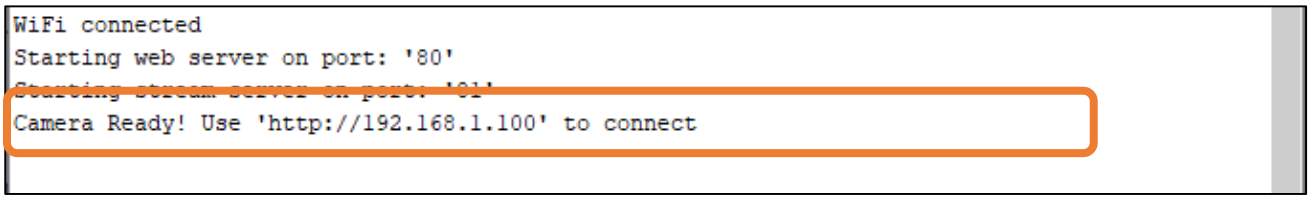


Before running the program, please modify your router's name and password in the box shown in the illustration above to make sure that your Sketch can compile and work successfully.

If your Arduino IDE prompts you that your sketch is out of your project's storage space, compile the code again as configured below.



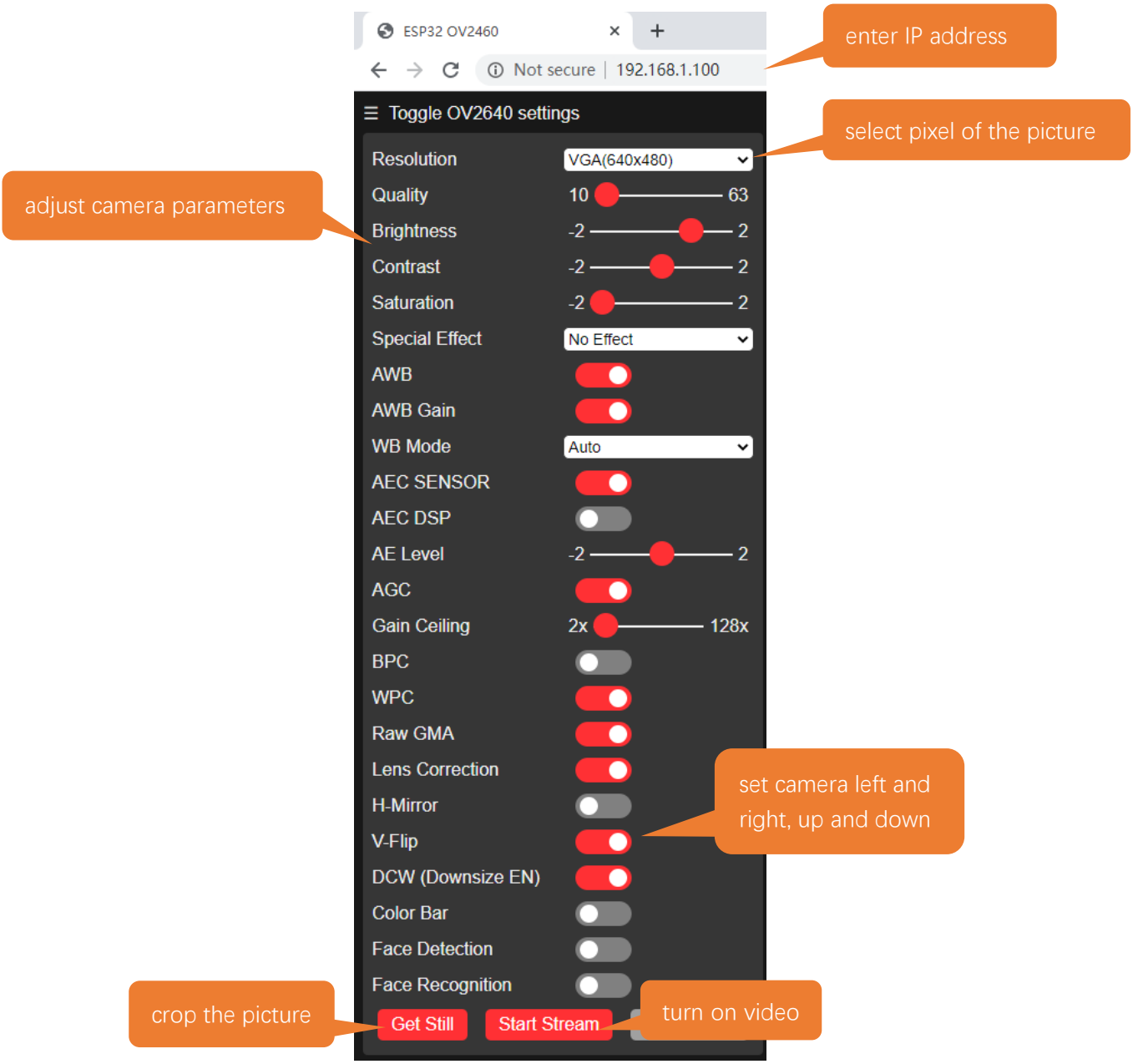
Compile and upload codes to ESP32, open the serial monitor and set the baud rate to 115200, and the serial monitor will print out a network link address.



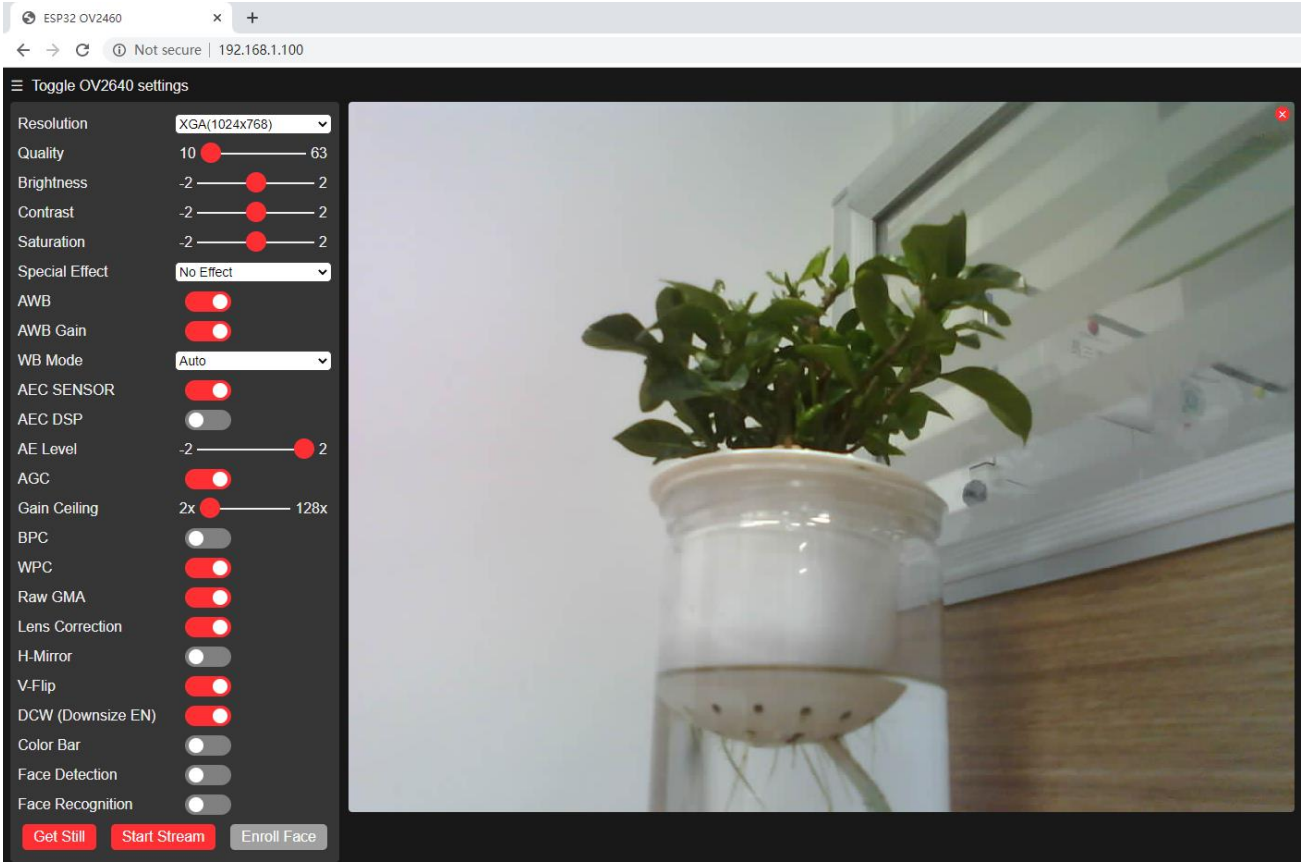
If your ESP32 has been in the process of connecting to router, but the information above has not been printed out, please re-check whether the router name and password have been entered correctly and press the reset key on ESP32-WROVER to wait for a successful connection prompt.

Open a web browser, enter the IP address printed by the serial monitor in the address bar, and access it. Taking the Google browser as an example, here's what the browser prints out after successful access to ESP32's IP.

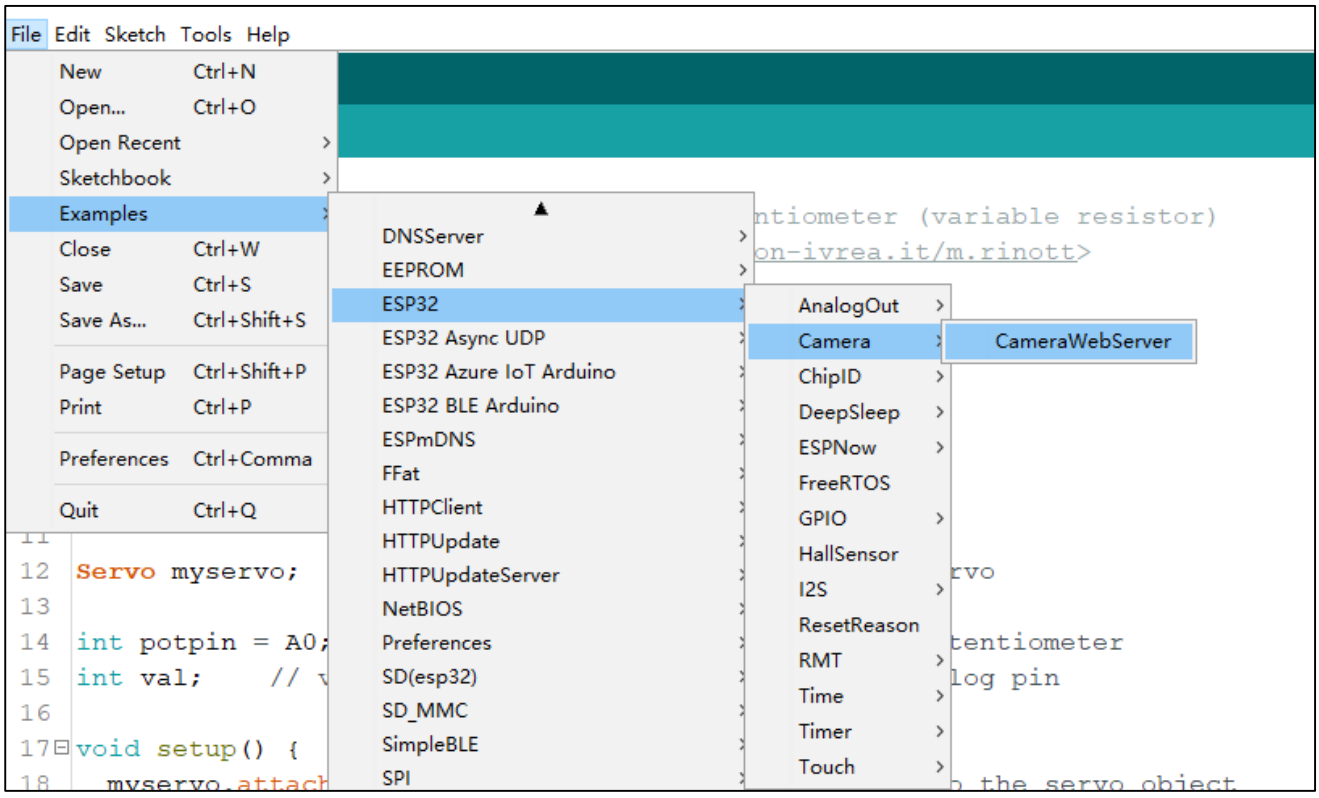
We recommend that the resolution not exceed VGA(640x480).



Click on Start Stream. The effect is shown in the image below.



Note: If sketch compilation fails due to ESP32 support package, follow the steps of the image to open the CameraWebServer. This sketch is the same as described in the tutorial above.



The following is the main program code. You need include other code files in the same folder when write your own code.

```
#include "esp_camera.h"
#include <WiFi.h>

// Select camera model
#define CAMERA_MODEL_WROVER_KIT
//#define CAMERA_MODEL_ESP_EYE
//#define CAMERA_MODEL_M5STACK_PSRAM
//#define CAMERA_MODEL_M5STACK_WIDE
//#define CAMERA_MODEL_AI_THINKER

#include "camera_pins.h"

const char *ssid_Router     = "********";  //input your wifi name
const char *password_Router = "********";  //input your wifi passwords
camera_config_t config;

void startCameraServer();
void camera_init();

void setup() {
  Serial.begin(115200);
  Serial.setDebugOutput(true);
  Serial.println();

  camera_init();
  config.frame_size = FRAMESIZE_VGA;
  config.jpeg_quality = 10;

  // camera init
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
  }

  sensor_t * s = esp_camera_sensor_get();
  s->set_vflip(s, 1);        //flip it back
  s->set_brightness(s, 1);   //up the blightness just a bit
  s->set_saturation(s, -1);  //lower the saturation

  WiFi.begin(ssid_Router, password_Router);
  while (WiFi.status() != WL_CONNECTED) {
```

```
43      delay(500);
44      Serial.print(".");
45    }
46    Serial.println("");
47    Serial.println("WiFi connected");
48
49    startCameraServer();
50
51    Serial.print("Camera Ready! Use 'http://");
52    Serial.print(WiFi.localIP());
53    Serial.println("' to connect");
54  }
55
56  void loop() {
57    ;
58  }
59
60  void camera_init(){
61    config.ledc_channel = LEDC_CHANNEL_0;
62    config.ledc_timer = LEDC_TIMER_0;
63    config.pin_d0 = Y2_GPIO_NUM;
64    config.pin_d1 = Y3_GPIO_NUM;
65    config.pin_d2 = Y4_GPIO_NUM;
66    config.pin_d3 = Y5_GPIO_NUM;
67    config.pin_d4 = Y6_GPIO_NUM;
68    config.pin_d5 = Y7_GPIO_NUM;
69    config.pin_d6 = Y8_GPIO_NUM;
70    config.pin_d7 = Y9_GPIO_NUM;
71    config.pin_xclk = XCLK_GPIO_NUM;
72    config.pin_pclk = PCLK_GPIO_NUM;
73    config.pin_vsync = VSYNC_GPIO_NUM;
74    config.pin_href = HREF_GPIO_NUM;
75    config.pin_sccb_sda = SIOD_GPIO_NUM;
76    config.pin_sccb_scl = SIOC_GPIO_NUM;
77    config.pin_pwdn = PWDN_GPIO_NUM;
78    config.pin_reset = RESET_GPIO_NUM;
79    config.xclk_freq_hz = 20000000;
80    config.pixel_format = PIXFORMAT_JPEG;
81    config.fb_count = 1;
82  }
```

Add procedure files and API interface files related to ESP32 camera.

```
#include "esp_camera.h"
#include <WiFi.h>

// Select camera model
#define CAMERA_MODEL_WROVER_KIT
//#define CAMERA_MODEL_ESP_EYE
//#define CAMERA_MODEL_M5STACK_PSRAM
//#define CAMERA_MODEL_M5STACK_WIDE
//#define CAMERA_MODEL_AI_THINKER

#include "camera_pins.h"
```

Enter the name and password of the router

```
const char *ssid_Router     = "********";  //input your wifi name
const char *password_Router = "********";  //input your wifi passwords
```

Initialize serial port, set baud rate to 115200; open the debug and output function of the serial.

```
  Serial.begin(115200);
  Serial.setDebugOutput(true);
  Serial.println();
```

Configure parameters including interface pins of the camera. Note: It is generally not recommended to change them.

```
void camera_init(){
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sccb_sda = SIOD_GPIO_NUM;
  config.pin_sccb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;
  config.fb_count = 1;
}
```

ESP32 connects to the router and prints a successful connection prompt. If it has not been successfully connected, press the reset key on the ESP32-WROVER.

```
41    WiFi.begin(ssid_Router, password_Router);
42    while (WiFi.status() ! = WL_CONNECTED) {
43      delay(500);
44      Serial.print(".");
45    }
46    Serial.println("");
47    Serial.println("WiFi connected");
```

Open the video streams server function of the camera and print its IP address via serial port.

```
49    startCameraServer();
50
51    Serial.print("Camera Ready!  Use 'http://");
52    Serial.print(WiFi.localIP());
53    Serial.println("' to connect");
```

Configure the display image information of the camera.

The set_vflip() function sets whether the image is flipped 180°, with 0 for no flip and 1 for flip 180°.

The set_brightness() function sets the brightness of the image, with values ranging from -2 to 2.

The set_saturation() function sets the color saturation of the image, with values ranging from -2 to 2.

```
36    sensor_t * s = esp_camera_sensor_get();
37    s->set_vflip(s, 1);        //flip it back
38    s->set_brightness(s, 1);   //up the blightness just a bit
39    s->set_saturation(s, -1);  //lower the saturation
```

Modify the resolution and sharpness of the images captured by the camera. The sharpness ranges from 10 to 63, and the smaller the number, the sharper the picture. The larger the number, the blurrier the picture. Please refer to the table below.

```
26    config.frame_size = FRAMESIZE_VGA;
27    config.jpeg_quality = 10;
```

Reference

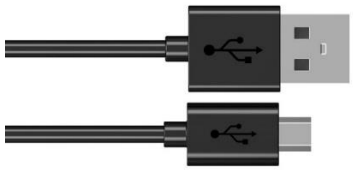
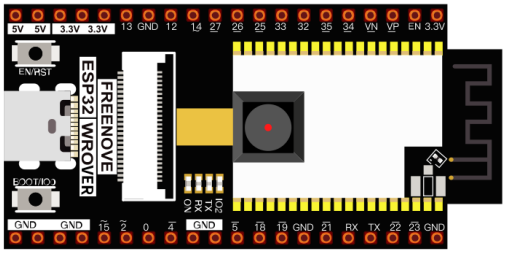
Image resolution	Sharpness	Image resolution	Sharpness
FRAMESIZE_96x96	96x96	FRAMESIZE_HVGA	480x320
FRAMESIZE_QQVGA	160x120	FRAMESIZE_VGA	640x480
FRAMESIZE_QCIF	176x144	FRAMESIZE_SVGA	800x600
FRAMESIZE_HQVGA	240x176	FRAMESIZE_XGA	1024x768
FRAMESIZE_240x240	240x240	FRAMESIZE_HD	1280x720
FRAMESIZE_QVGA	320x240	FRAMESIZE_SXGA	1280x1024
FRAMESIZE_CIF	400x296	FRAMESIZE_UXGA	1600x1200

We recommend that the resolution not exceed VGA(640x480).

Project 6.2 Video Web Server

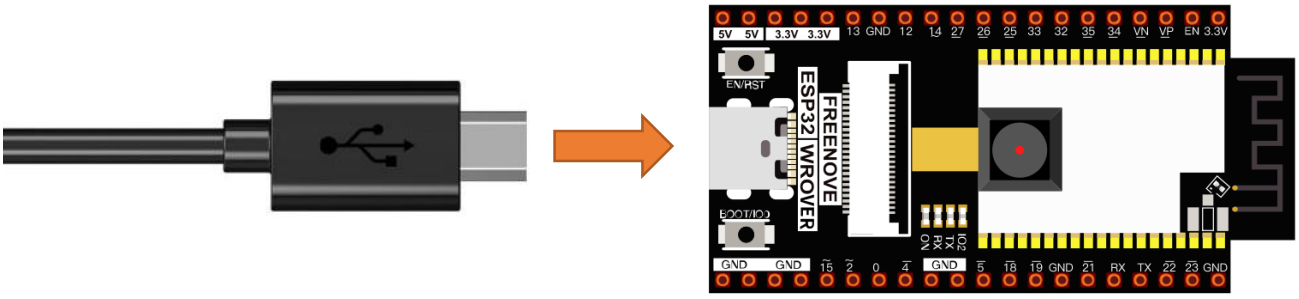
Connect to ESP32 using USB and view its IP address through a serial monitor. Access IP addresses through web pages to obtain real-time video data.

Component List

Micro USB Wire x1 	ESP32-WROVER x1 

Circuit

Connect Freenove ESP32 to the computer using USB cable.

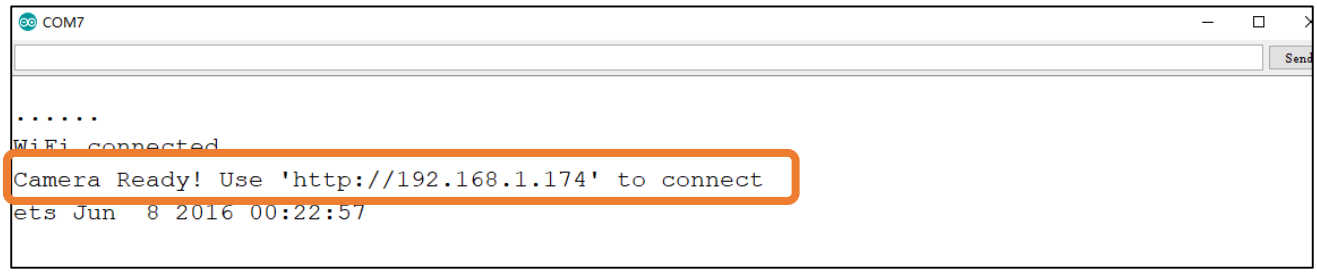


Sketch

Sketch_06.2_As_VideoWebServer

Before running the program, please modify your router's name and password in the box shown in the illustration above to make sure that your Sketch can compile and work successfully.

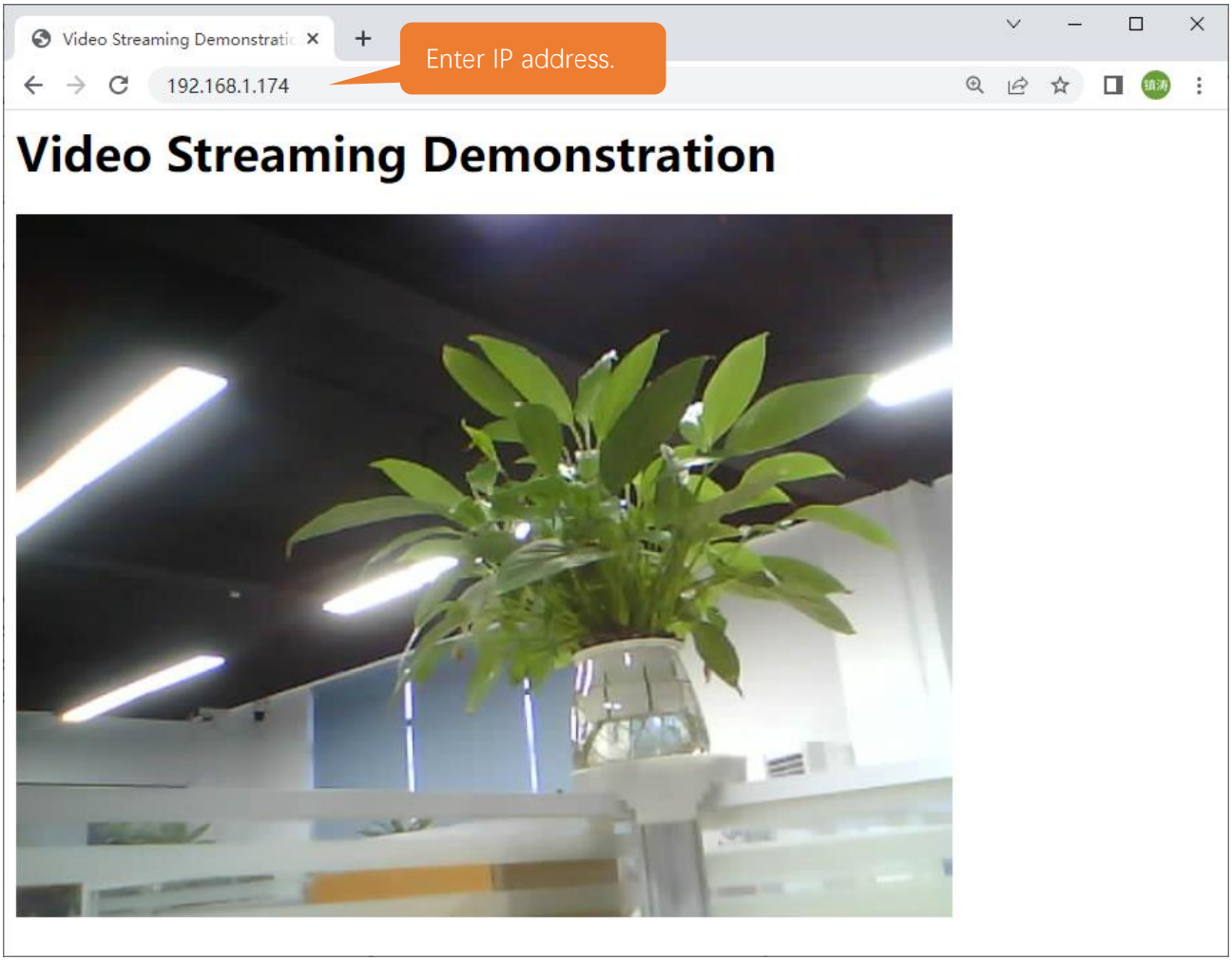
Compile and upload codes to ESP32, open the serial monitor and set the baud rate to 115200, and the serial monitor will print out a network link address.



If your ESP32 has been in the process of connecting to router, but the information above has not been printed out, please re-check whether the router name and password have been entered correctly and press the reset key on ESP32-WROVER to wait for a successful connection prompt.

Open a web browser, enter the IP address printed by the serial monitor in the address bar, and access it. Taking the Google browser as an example, here's what the browser prints out after successful access to ESP32's IP.

The effect is shown in the image below.



The following is the main program code. You need include other code files in the same folder when write your own code.

```
#include "esp_camera.h"
#include <WiFi.h>
//
// WARNING!!! PSRAM IC required for UXGA resolution and high JPEG quality
//            Ensure ESP32 Wrover Module or other board with PSRAM is selected
//            Partial images will be transmitted if image exceeds buffer size
//

// Select camera model
#define CAMERA_MODEL_WROVER_KIT // Has PSRAM

```

```
12  #include "camera_pins.h"
13
14  const char* ssid     = "********";   //input your wifi name
15  const char* password = "********";  //input your wifi passwords
16
17  void startCameraServer();
18
19  void setup() {
20    Serial.begin(115200);
21    Serial.setDebugOutput(true);
22    Serial.println();
23
24    camera_config_t config;
25    config.ledc_channel = LEDC_CHANNEL_0;
26    config.ledc_timer = LEDC_TIMER_0;
27    config.pin_d0 = Y2_GPIO_NUM;
28    config.pin_d1 = Y3_GPIO_NUM;
29    config.pin_d2 = Y4_GPIO_NUM;
30    config.pin_d3 = Y5_GPIO_NUM;
31    config.pin_d4 = Y6_GPIO_NUM;
32    config.pin_d5 = Y7_GPIO_NUM;
33    config.pin_d6 = Y8_GPIO_NUM;
34    config.pin_d7 = Y9_GPIO_NUM;
35    config.pin_xclk = XCLK_GPIO_NUM;
36    config.pin_pclk = PCLK_GPIO_NUM;
37    config.pin_vsync = VSYNC_GPIO_NUM;
38    config.pin_href = HREF_GPIO_NUM;
39    config.pin_sccb_sda = SIOD_GPIO_NUM;
40    config.pin_sccb_scl = SIOC_GPIO_NUM;
41    config.pin_pwdn = PWDN_GPIO_NUM;
42    config.pin_reset = RESET_GPIO_NUM;
43    config.xclk_freq_hz = 20000000;
44    config.pixel_format = PIXFORMAT_JPEG;
45
46    // if PSRAM IC present, init with UXGA resolution and higher JPEG quality
47    //                      for larger pre-allocated frame buffer.
48    if(psramFound()){
49      config.frame_size = FRAMESIZE_VGA;
50      config.jpeg_quality = 10;
51      config.fb_count = 2;
52    } else {
53      config.frame_size = FRAMESIZE_HVGA;
54      config.jpeg_quality = 12;
55      config.fb_count = 1;
```

```
  }

  // camera init
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
  }

  sensor_t * s = esp_camera_sensor_get();
  // drop down frame size for higher initial frame rate
  s->set_framesize(s, FRAMESIZE_QVGA);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");

  startCameraServer();

  Serial.print("Camera Ready! Use 'http://");
  Serial.print(WiFi.localIP());
  Serial.println("' to connect");
}

void loop() {
  // put your main code here, to run repeatedly:
  delay(10000);
}
```

Configure parameters including interface pins of the camera. Note: It is generally not recommended to change them.

```
  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
```

```
32    config.pin_d5 = Y7_GPIO_NUM;
33    config.pin_d6 = Y8_GPIO_NUM;
34    config.pin_d7 = Y9_GPIO_NUM;
35    config.pin_xclk = XCLK_GPIO_NUM;
36    config.pin_pclk = PCLK_GPIO_NUM;
37    config.pin_vsync = VSYNC_GPIO_NUM;
38    config.pin_href = HREF_GPIO_NUM;
39    config.pin_sccb_sda = SIOD_GPIO_NUM;
40    config.pin_sccb_scl = SIOC_GPIO_NUM;
41    config.pin_pwdn = PWDN_GPIO_NUM;
42    config.pin_reset = RESET_GPIO_NUM;
43    config.xclk_freq_hz = 20000000;
44    config.pixel_format = PIXFORMAT_JPEG;
```

ESP32 connects to the router and prints a successful connection prompt. If it has not been successfully connected, press the reset key on the ESP32-WROVER.

```
69    WiFi.begin(ssid, password);
70
71    while (WiFi.status() != WL_CONNECTED) {
72      delay(500);
73      Serial.print(".");
74    }
75    Serial.println("");
76    Serial.println("WiFi connected");
```

Open the video streams server function of the camera and print its IP address via serial port.

```
78    startCameraServer();
79
80    Serial.print("Camera Ready! Use 'http://");
81    Serial.print(WiFi.localIP());
82    Serial.println("' to connect");
```

What's next?

Thanks for your reading. This tutorial is all over here. If you find any mistakes, omissions or you have other ideas and questions about contents of this tutorial or the kit and etc., please feel free to contact us:

support@freenove.com

We will check and correct it as soon as possible.

If you want learn more about ESP32, you view our ultimate tutorial:
https://github.com/Freenove/Freenove_ESP32_WROVER_Board/archive/master.zip

If you want to learn more about Arduino, Raspberry Pi, smart cars, robots and other interesting products in science and technology, please continue to focus on our website. We will continue to launch cost-effective, innovative and exciting products.

http://www.freenove.com/

End of the Tutorial

Thank you again for choosing Freenove products.