

# Welcome

Thank you for choosing Freenove products!

## Getting Started

When reading this, you should have downloaded the ZIP file for this product.

Unzip it and you will get a folder containing tutorials and related files. Please start with this PDF tutorial.

- ! Unzip the ZIP file instead of opening the file in the ZIP file directly.
- ! Do not move, delete or rename files in the folder just unzipped.

## Get Support

Encounter problems? Don't worry! Refer to "TroubleShooting.pdf" or contact us.

When there are packaging damage, quality problems, questions encountering in use, etc., just send us an email. We will reply to you within one working day and provide a solution.

[support@freenove.com](mailto: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.

Any concerns?  [support@freenove.com](mailto:support@freenove.com)

## 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>

## Copyright

All the files, materials and instructional guides provided are released under [Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License](#). A copy of this license can be found in the folder containing the Tutorial and software files associated with this product.



This means you can use these resources in your own derived works, in part or completely, but **NOT for the intent or purpose of commercial use.**

Freenove brand and logo are copyright of Freenove Creative Technology Co., Ltd. and cannot be used without written permission.



Other registered trademarks and their owners appearing in this document:

Arduino® is a trademark of Arduino LLC (<https://www.arduino.cc/>).

Raspberry Pi® is a trademark of Raspberry Pi Foundation (<https://www.raspberrypi.org/>).

Raspberry Pi Pico® is a trademark of Raspberry Pi Foundation (<https://www.raspberrypi.org/>).

micro:bit® is a trademark of Micro:bit Educational Foundation (<https://www.microbit.org/>).

ESPRESSIF® and ESP32® are trademarks of ESPRESSIF Systems (Shanghai) Co, Ltd (<https://www.espressif.com/>).

**Any concerns? ✉ [support@freenove.com](mailto:support@freenove.com)**

# Contents

Welcome.....	1
Contents .....	1
Preface.....	3
Raspberry Pi Pico.....	4
Raspberry Pi Pico W .....	7
Raspberry Pi Pico 2.....	10
Chapter 0 Getting Ready (Important).....	13
Programming Software.....	13
Installation of Development Board Support Package .....	16
Uploading Arduino-compatible Firmware for Pico.....	18
Paste the Sticker on the Breadboard.....	22
Chapter 1 LED (Important).....	23
Project 1.1 Blink .....	23
Project 1.2 Blink .....	29
Chapter 2 Button & LED .....	34
Project 2.1 Button & LED.....	35
Project 2.2 MINI table lamp.....	39
Chapter 3 LED Bar .....	43
Project 3.1 Flowing Light .....	43
Chapter 4 Analog & PWM .....	48
Project 4.1 Breathing LED.....	48
Project 4.2 Meteor Flowing Light .....	54
Chapter 5 RGBLED .....	59
Project 5.1 Random Color Light.....	59
Project 5.2 Gradient Color Light.....	64
Chapter 6 Buzzer .....	66
Project 6.1 Doorbell.....	66
Project 6.2 Alertor .....	72
Chapter 7 Serial Communication.....	77

---

Project 7.1 Serial Print.....	77
Project 7.2 Serial Read and Write .....	81
<b>Chapter 8 AD Converter.....</b>	<b>84</b>
Project 8.1 Read the Voltage of Potentiometer.....	84
<b>Chapter 9 Potentiometer &amp; LED .....</b>	<b>90</b>
Project 9.1 Soft Light.....	90
<b>Chapter 10 Photoresistor &amp; LED .....</b>	<b>94</b>
Project 10.1 Control LED through Photoresistor.....	94
<b>Chapter 11 Thermistor .....</b>	<b>99</b>
Project 11.1 Thermometer .....	99
<b>Chapter 12 WiFi Working Modes (Only for Pico W).....</b>	<b>104</b>
Project 12.1 Station mode.....	104
Project 12.2 AP mode.....	109
Project 12.3 AP+Station mode .....	114
<b>Chapter 13 TCP/IP (Only for Pico W).....</b>	<b>118</b>
Project 13.1 as Client .....	118
Project 13.2 as Server .....	132
<b>Chapter 14 Control LED with Web (Only for Pico W).....</b>	<b>138</b>
Project 14.1 Control the LED with Web .....	138
<b>Chapter 15 Bluetooth (Only for Pico W).....</b>	<b>146</b>
Project 15.1 Bluetooth Passthrough .....	146
Project 15.2 Bluetooth Low Energy Data Passthrough.....	153
Project 15.3 Bluetooth Control LED .....	167
Project 15.4 Bluetooth Low Energy Control LED .....	173
<b>Chapter 16 Bluetooth Audio (Only for Pico W).....</b>	<b>181</b>
Project 15.1 Bluetooth Passthrough .....	181
<b>What's Next? .....</b>	<b>189</b>

# Preface

Raspberry Pi Pico is a tiny, fast, and versatile board built using RP2040, a brand new microcontroller chip designed by Raspberry Pi in the UK. Supporting Python and C/C++ development, it is perfect for DIY projects. In this tutorial, we use Arduino to learn Pico. If you want to learn the Python version, please refer to another tutorial: [python\\_tutorial.pdf](#).

Using Arduino IDE as the development environment for Raspberry Pi Pico allows users to learn Pico better and more quickly, which is just like developing Arduino programs. In addition, resources such as Arduino's libraries can be directly used to greatly improve the efficiency of development.

If you have not downloaded the related material for Raspberry Pi Pico tutorial, you can download it from this link:

[https://github.com/Freenove/Freenove\\_Basic\\_Starter\\_Kit\\_for\\_Raspberry\\_Pi\\_Pico](https://github.com/Freenove/Freenove_Basic_Starter_Kit_for_Raspberry_Pi_Pico)

In this tutorial, we divide each project into 4 sections:

- 1, Component list: helps users to learn and find what components are needed in each project.
- 2, Component Knowledge: allows you to learn the features and usage of the components.
- 3, Circuit: assists to build circuit for each project.
- 4, Sketches and comments: makes it easier for users to learn to use Raspberry Pi Pico and make secondary development.

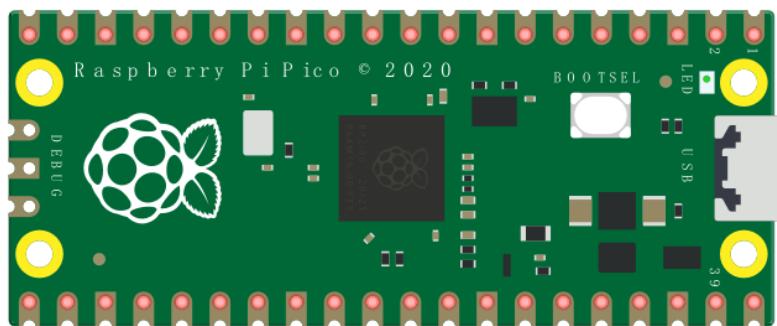
After completing the projects in this tutorial, you can also combine the components in different projects to make your own smart homes, smart car, robot, etc., bringing your imagination and creativity to life with Raspberry Pi Pico.

If you have any problems or difficulties using this product, please contact us for quick and free technical support: [support@freenove.com](mailto:support@freenove.com)

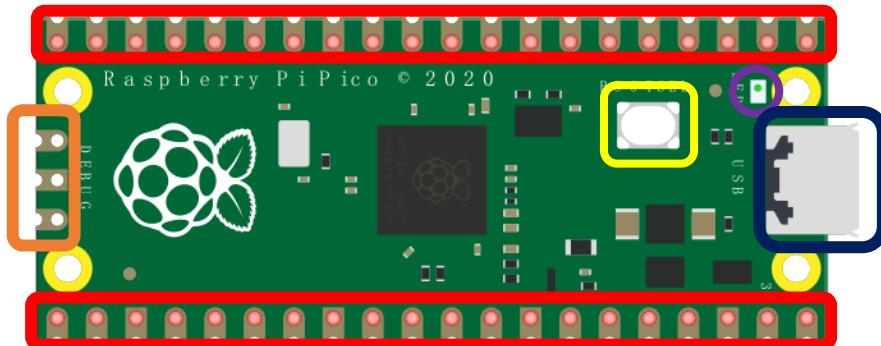
## Raspberry Pi Pico

Raspberry Pi Pico applies to all chapters except Wireless in this tutorial.

Before learning Pico, we need to know about it. Below is an imitated diagram of Pico, which looks very similar to the actual Pico.

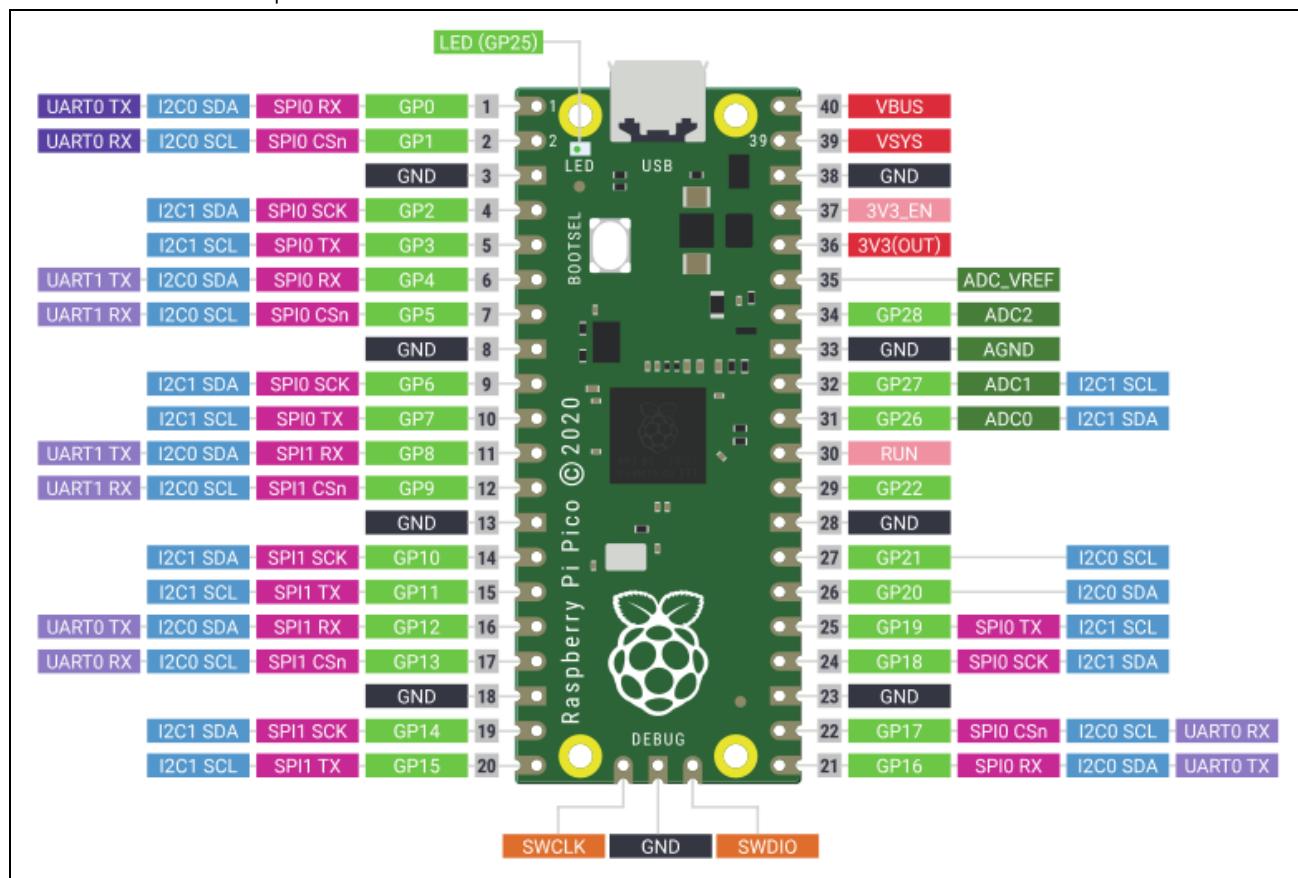


The hardware interfaces are distributed as follows:



Frame color	Description
	Pins
	BOOTSEL button
	USB port
	LED
	Debugging

Function definition of pins:



Color	Pins	Color	Pins
Black	GND	Red	Power
Green	GPIO	Dark Green	ADC
Magenta	UART(default)	Lavender	UART
Magenta	SPI	Blue	I2C
Pink	System Control	Orange	Debugging

For details: <https://datasheets.raspberrypi.org/pico/pico-datasheet.pdf>



## UART, I2C, SPI Default Pin

In Arduino IDE, the default pins of serial port are Pin0 and Pin1.

Note: Serial port is virtualized by RP2040. Therefore, when using the serial port, please enable the verification function of DTR. It can work under any baud rate.

### UART

Function	Default
UART_BAUDRATE	X
UART_BITS	8
UART_STOP	1
UART_TX	Pin 0
UART_RX	Pin 1

### I2C

Function	Default
I2C Frequency	400000
I2C_SDA	Pin 4
I2C_SCL	Pin 5

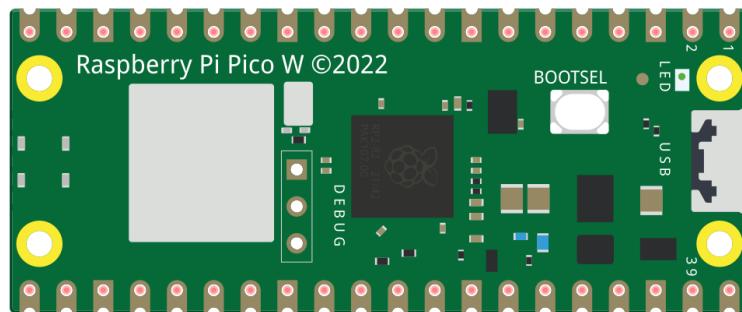
### SPI

Function	Default
SPI_BAUDRATE	1000000
SPI_POLARITY	0
SPI_PHASE	0
SPI_BITS	8
SPI_FIRSTBIT	MSB
SPI_SCK	Pin 18
SPI_MOSI	Pin 19
SPI_MISO	Pin 16
SPI_SS	Pin 17

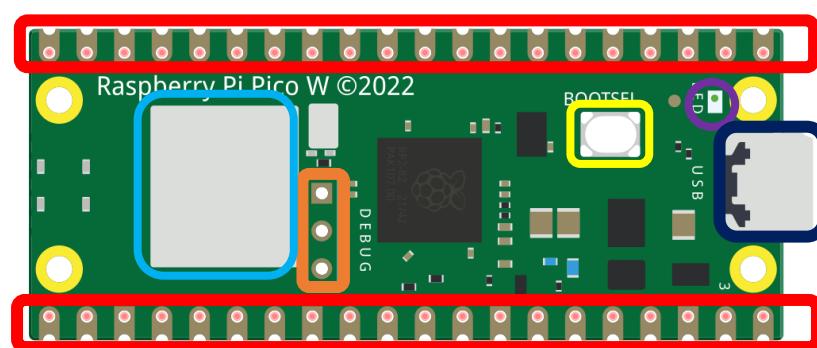
## Raspberry Pi Pico W

Raspberry Pi Pico W applies to all chapters in this tutorial.

Raspberry Pi Pico W adds CYW43439 as the WiFi function based on Raspberry Pi Pico. It is connected to RP2040 chip through SPI interface.

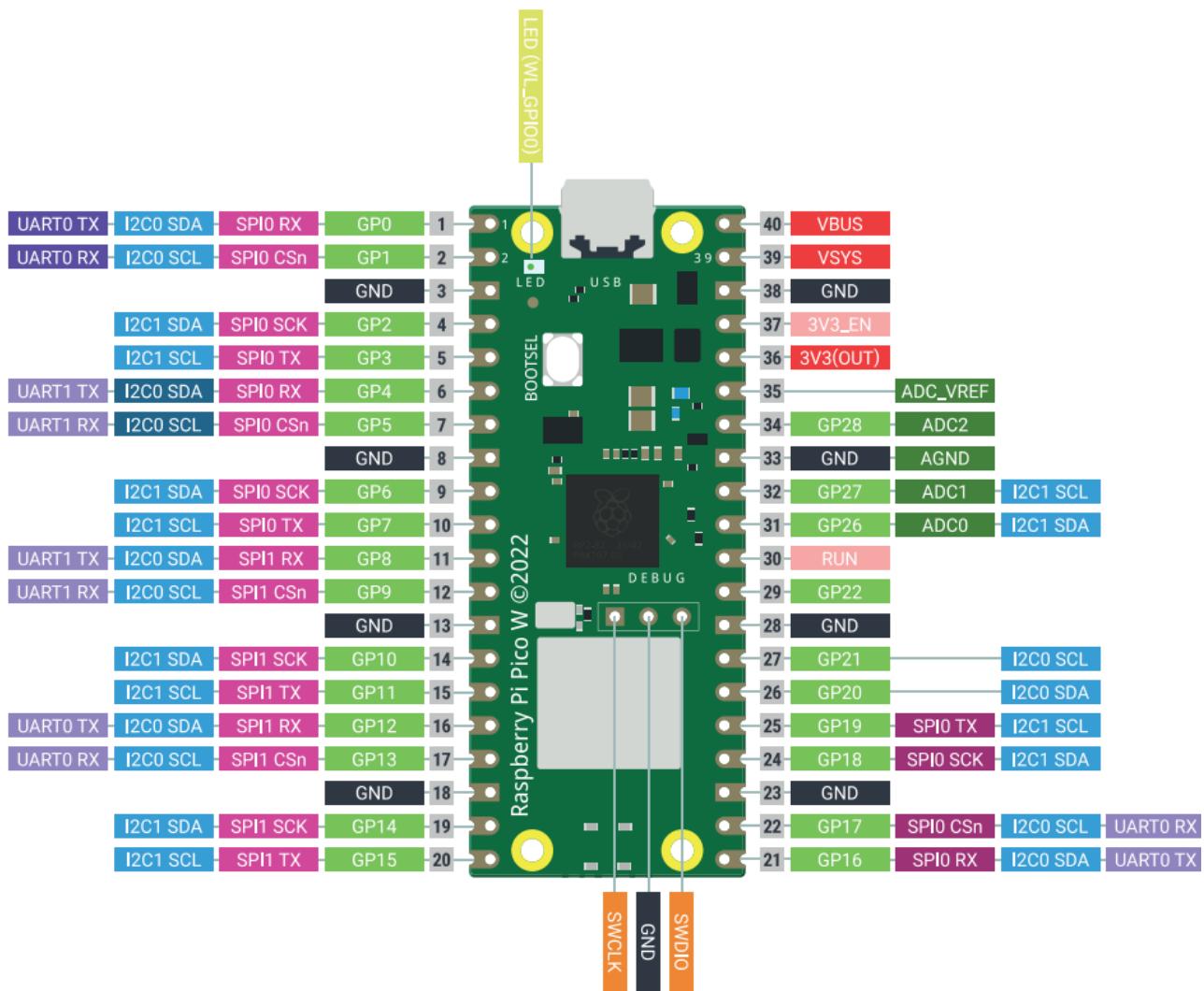


The hardware interfaces are distributed as follows:



Frame color	Description
	Pins
	BOOTSEL button
	USB port
	LED
	Debugging
	Wireless

Function definition of pins:



Color	Pins	Color	Pins
Black	GND	Red	Power
Green	GPIO	Dark Green	ADC
Purple	UART(default)	Lavender	UART
Magenta	SPI	Cyan	I2C
Pink	System Control	Orange	Debugging

For details: <https://datasheets.raspberrypi.com/picow/pico-w-datasheet.pdf>

## UART, I2C, SPI, Wireless Default Pin

In Arduino IDE, the default pins of serial port are Pin0 and Pin1.

Note: Serial port is virtualized by RP2040. Therefore, when using the serial port, please enable the verification function of DTR. It can work under any baud rate.

### UART

Function	Default
UART_BAUDRATE	X
UART_BITS	8
UART_STOP	1
UART_TX	Pin 0
UART_RX	Pin 1

### I2C

Function	Default
I2C Frequency	400000
I2C_SDA	Pin 4
I2C_SCL	Pin 5

### SPI

Function	Default
SPI_BAUDRATE	1000000
SPI_POLARITY	0
SPI_PHASE	0
SPI_BITS	8
SPI_FIRSTBIT	MSB
SPI_SCK	Pin 18
SPI_MOSI	Pin 19
SPI_MISO	Pin 16
SPI_SS	Pin 17

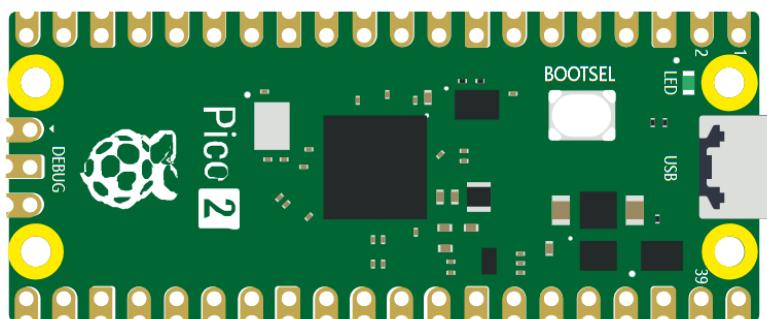
### Wireless

Function	Default
WL_ON	GPIO23
WL_D	GPIO24
WL_CLK	GPIO29_ADC
WL_CS	GPIO25

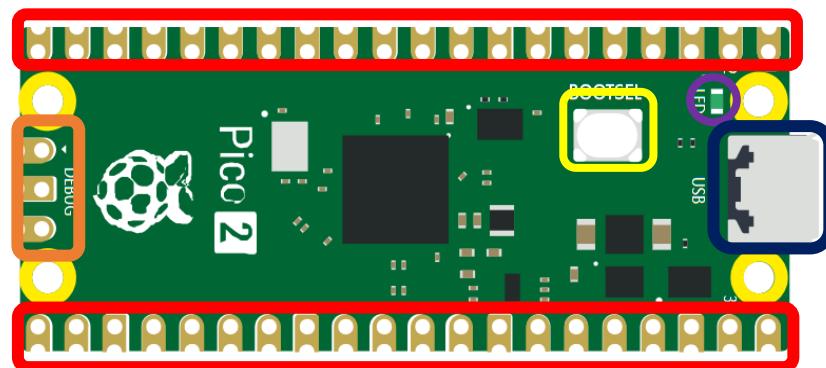
## Raspberry Pi Pico 2

Raspberry Pi Pico 2 is applicable to all chapters in this tutorial except RFID and those involving WiFi.

Raspberry Pi Pico 2 uses RP2350 chip as the main controller, which equipped with dual Cortex-M33 or Hazard3 processors, capable of running up to 150 MHz, providing a significant boost in processing power, compared with the original Pico. It also doubles the memory with 520KB of SRAM and 4MB of onboard flash memory, with the ADC sampling frequency increasing to up to 500ksps. In addition, it adds 8 more PWM channels, and features additional interfaces like 2× Timer with 4 alarms, 1× AON Timer and 4 x PIO.

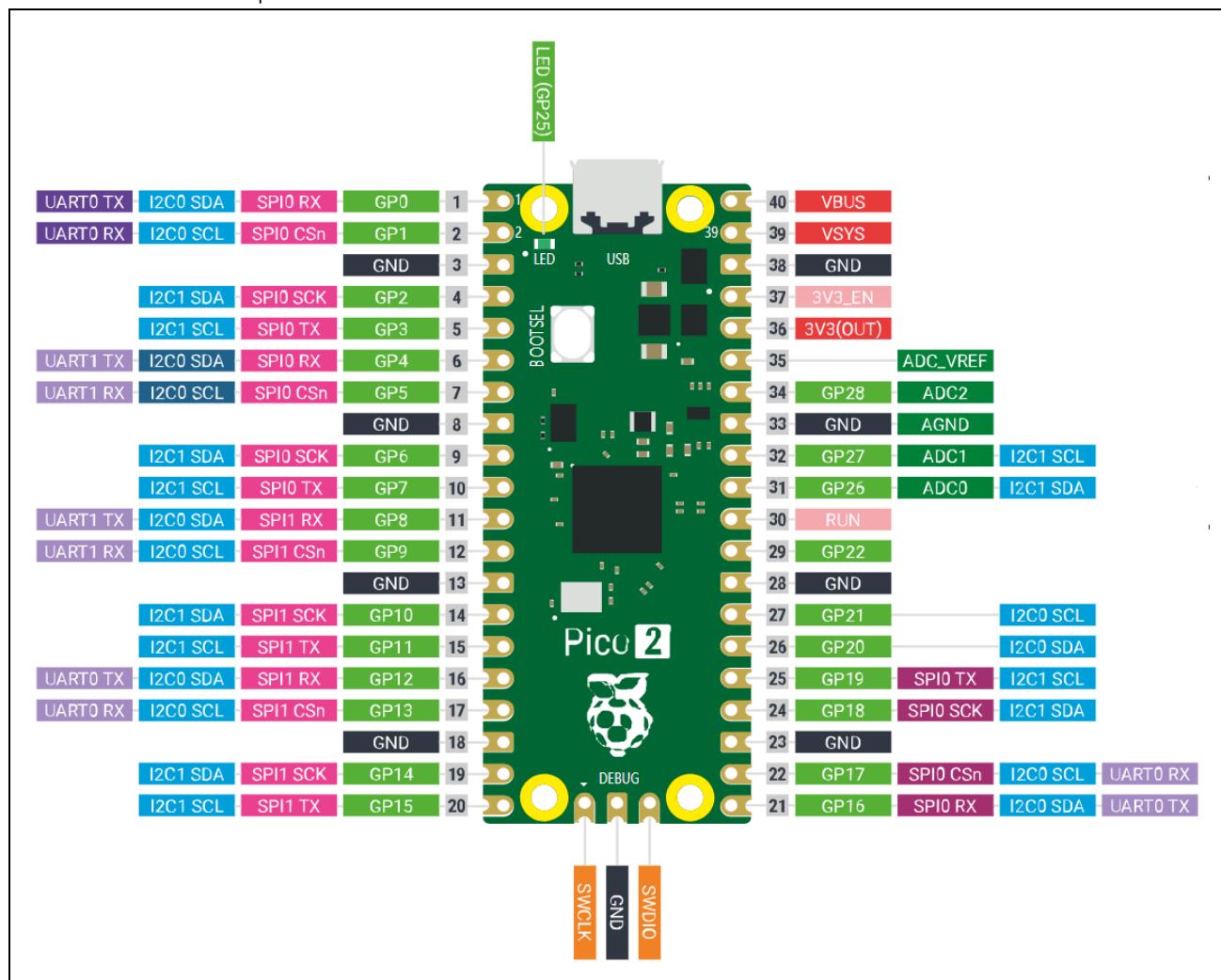


The hardware interfaces are distributed as follows:



Frame color	Description
	Pins
	BOOTSEL button
	USB port
	LED
	Debugging

Function definition of pins:



Color	Pins	Color	Pins
Black	GND	Red	Power
Green	GPIO	Dark Green	ADC
Purple	UART(default)	Lavender	UART
Magenta	SPI	Blue	I2C
Pink	System Control	Orange	Debugging

For details: <https://datasheets.raspberrypi.com/pico/pico-2-datasheet.pdf>

## UART, I2C, SPI Default Pin

In Arduino IDE, the default pins of serial port are Pin0 and Pin1.

Note: Serial port is virtualized by RP2350. Therefore, when using the serial port, please enable the verification function of DTR. It can work under any baud rate.

### UART

Function	Default
UART_BAUDRATE	X
UART_BITS	8
UART_STOP	1
UART_TX	Pin 0
UART_RX	Pin 1

### I2C

Function	Default
I2C Frequency	400000
I2C_SDA	Pin 4
I2C_SCL	Pin 5

### SPI

Function	Default
SPI_BAUDRATE	1000000
SPI_POLARITY	0
SPI_PHASE	0
SPI_BITS	8
SPI_FIRSTBIT	MSB
SPI_SCK	Pin 18
SPI_MOSI	Pin 19
SPI_MISO	Pin 16
SPI_SS	Pin 17

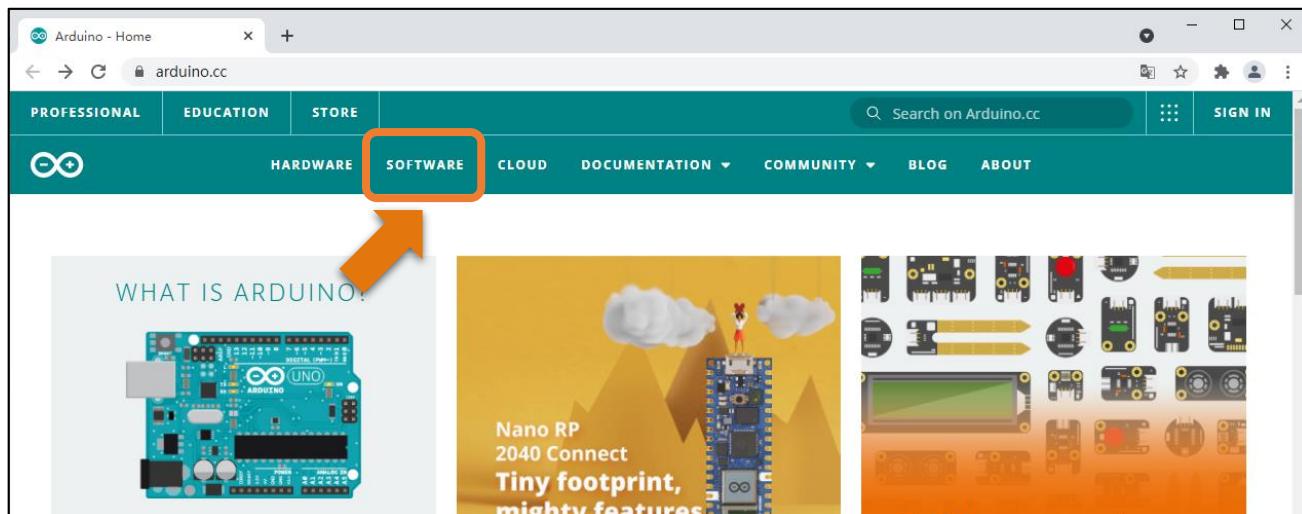
# Chapter 0 Getting Ready (Important)

Before starting building the projects, you need to make some preparation first, which is so crucial that you must not skip.

## 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.

## Downloads

The screenshot shows the download page for Arduino IDE 2.3.2. On the left, there's a thumbnail of the Arduino IDE icon, the title "Arduino IDE 2.3.2", and a brief description: "The new major release of the Arduino IDE is faster and even more powerful! In addition to a more modern editor and a more responsive interface it features autocomplete, code navigation, and even a live debugger." Below this, there's a link to "Arduino IDE 2.0 documentation" and a note about nightly builds. On the right, there's a large section titled "DOWNLOAD OPTIONS" with links for Windows (Win 10 and newer, 64 bits, MSI installer, ZIP file), Linux (AppImage 64 bits (X86-64), ZIP file 64 bits (X86-64)), and macOS (Intel, 10.15: "Catalina" or newer, 64 bits, Apple Silicon, 11: "Big Sur" or newer, 64 bits). A link to "Release Notes" is also present.

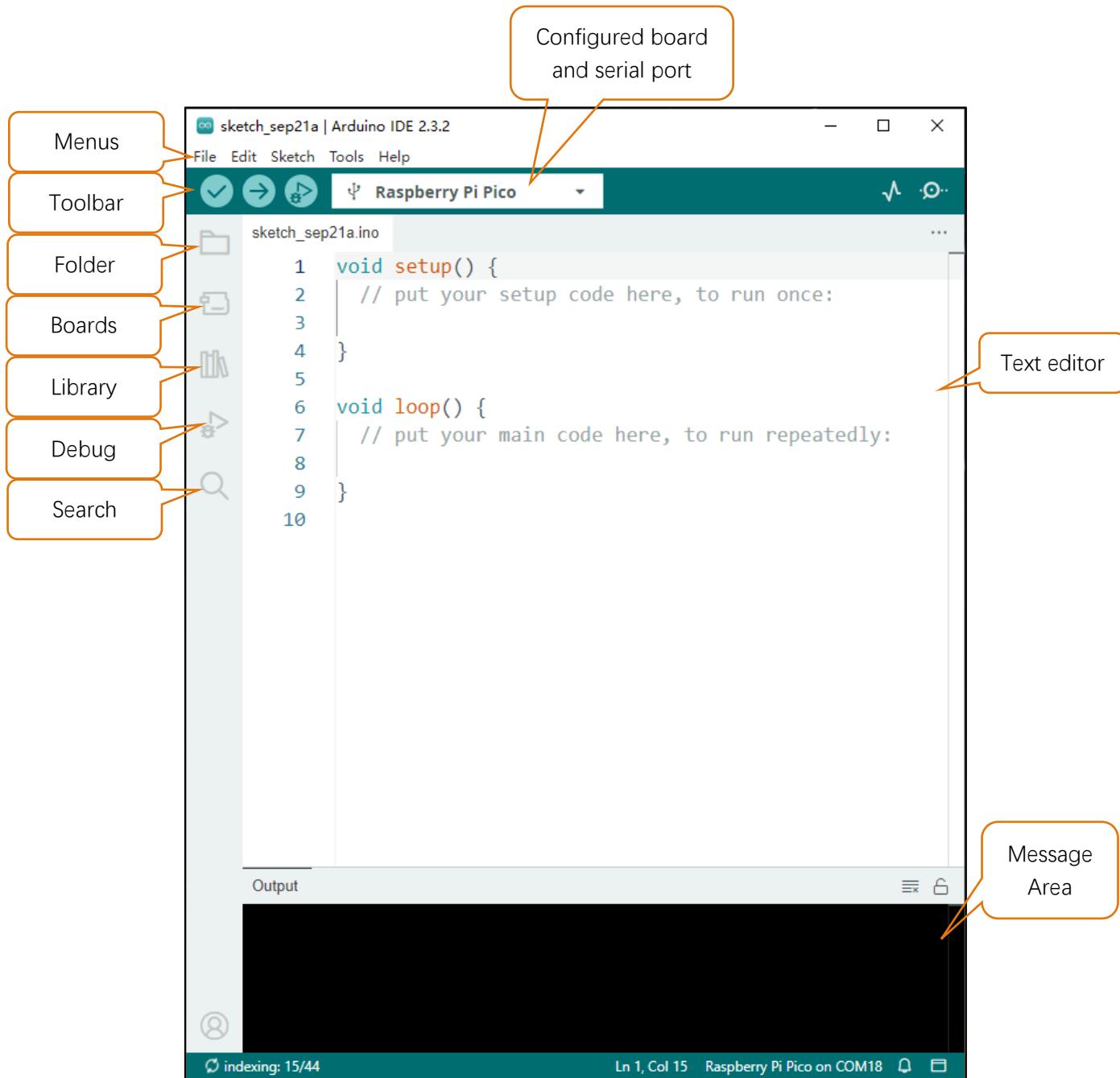


After the download completes, run the installer. For Windows users, there may pop up an installation dialog box of driver during the installation process. When it comes up, please allow the installation.

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, upload, and debug programs, select board & port, and open serial plotter and serial monitor.



### Verify

Check your code for compile errors.



### Upload

Compile your code and upload them to the configured board.



### Debug

Test and debug programs in real time.



### Select Board & Port

Detected Arduino boards automatically show up here, along with the port number.



### 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.

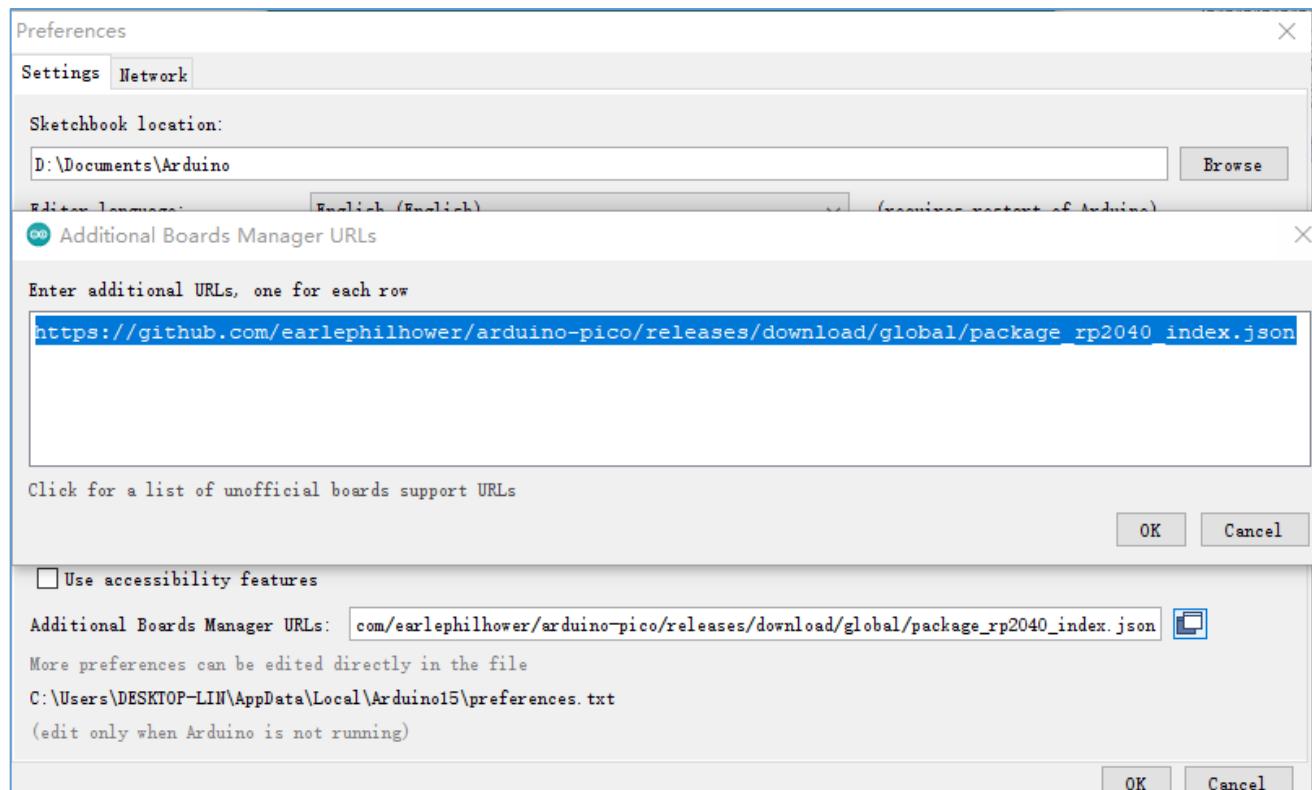


## Installation of Development Board Support Package

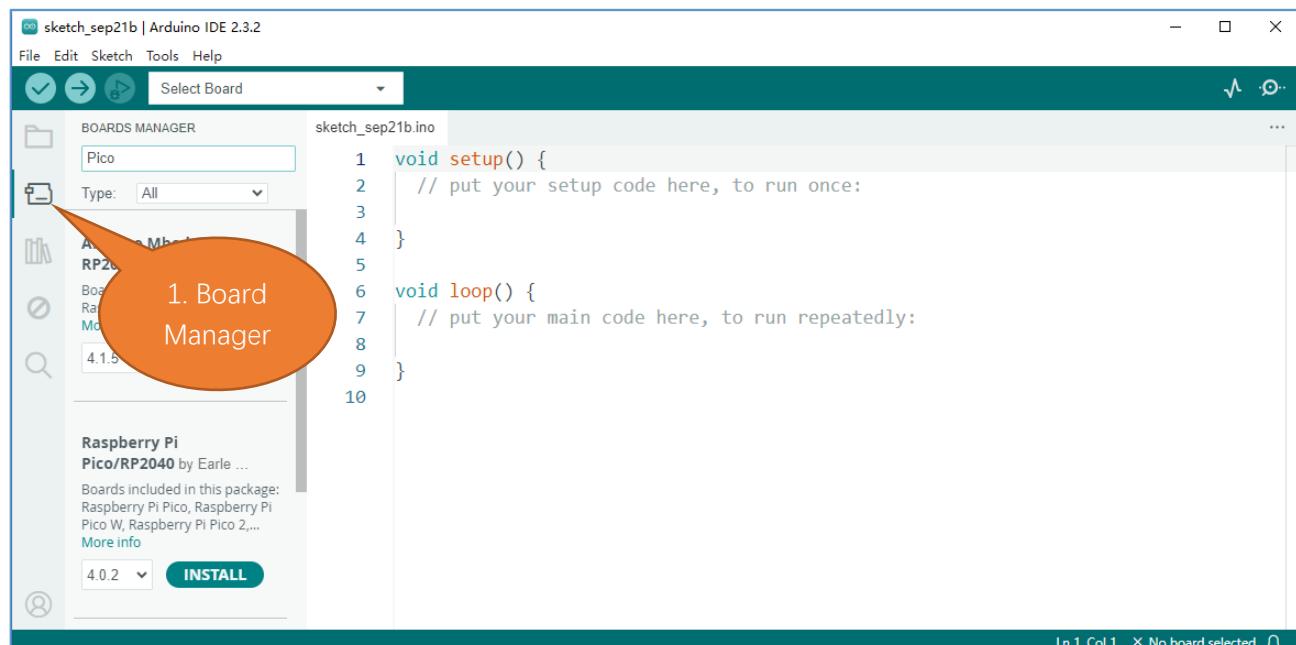
1. Make sure your network is of good connection.
2. Open Arduino IDE, and click File>Preference. In new pop-up window, find "Additional Boards Manager URLs", and replace with a new line:

[https://github.com/earlephilhower/arduino-pico/releases/download/global/package\\_rp2040\\_index.json](https://github.com/earlephilhower/arduino-pico/releases/download/global/package_rp2040_index.json)

As shown below:

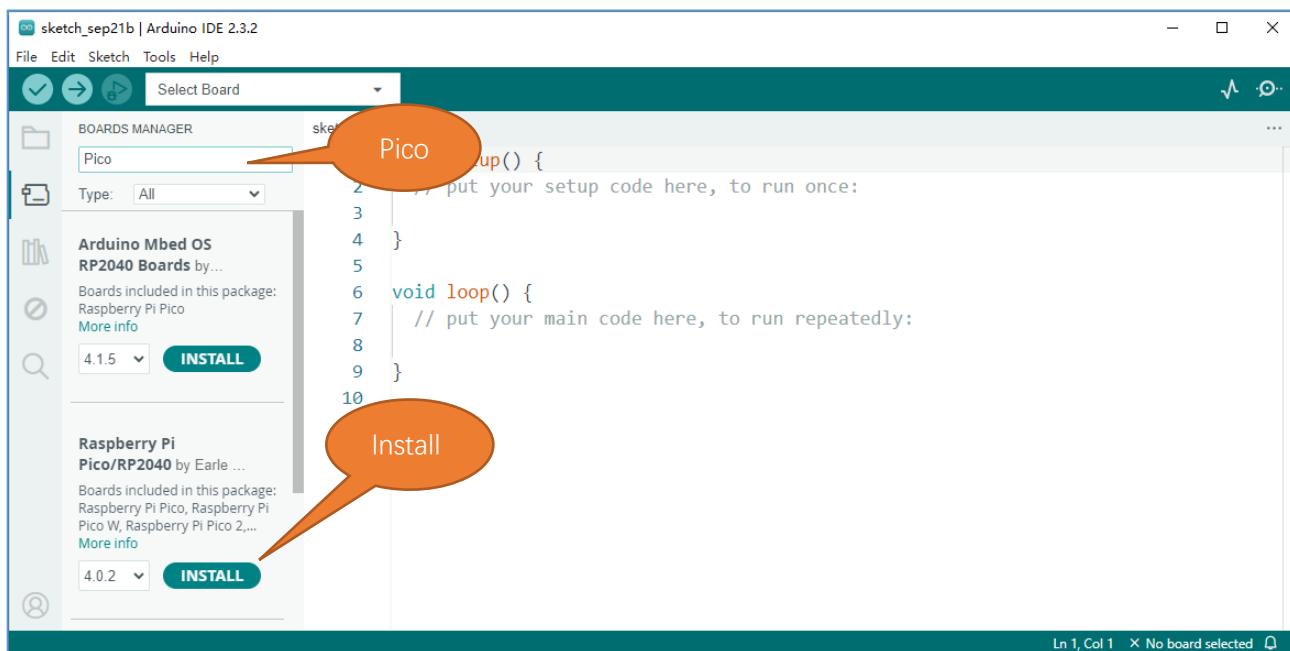


3. Open Arduino IDE; click Boards Manager on the left.



Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

4. Enter Pico in the searching box, and select “Raspberry Pi Pico/RP2040” and click on Install.



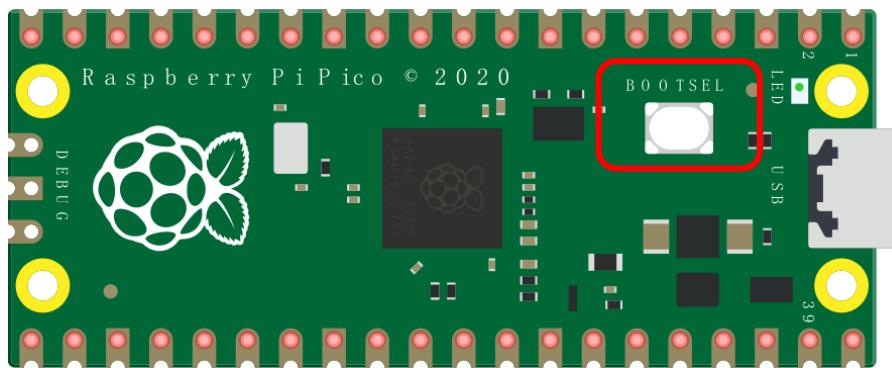
5. Click Yes in the pop-up “**dpinst-amd64.exe**” installation window. (Without it, you will fail to communicate with Arduino.) Thus far, we have finished installing the development support package.



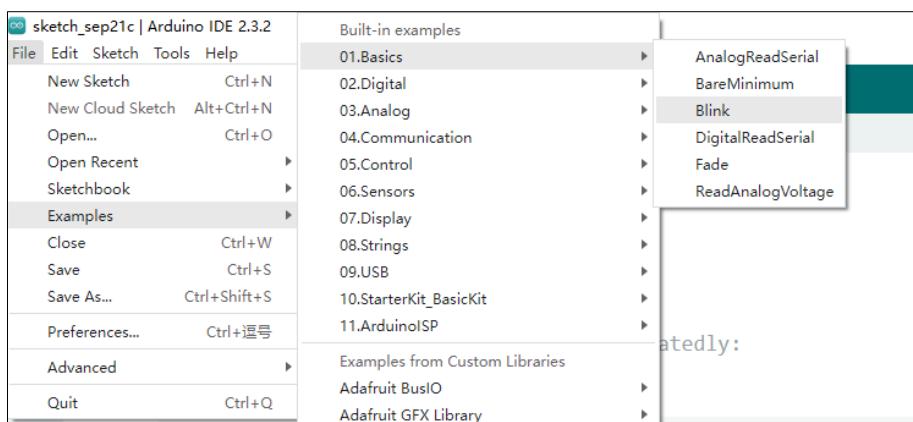
## Uploading Arduino-compatible Firmware for Pico

If your Pico is new and you want to use Arduino to learn and develop, you need to upload an Arduino-compatible Firmware for it. Please refer to the following steps to configure.

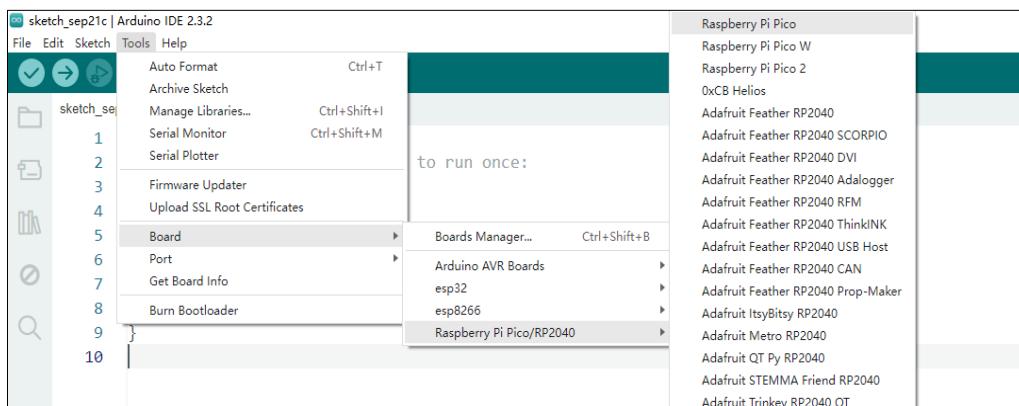
1. Disconnect Pico from computer. Keep pressing the white button (BOOTSEL) on Pico, and connect Pico to computer before releasing the button. (Note: Be sure to keep pressing the button before powering the Pico, otherwise the firmware will not download successfully)



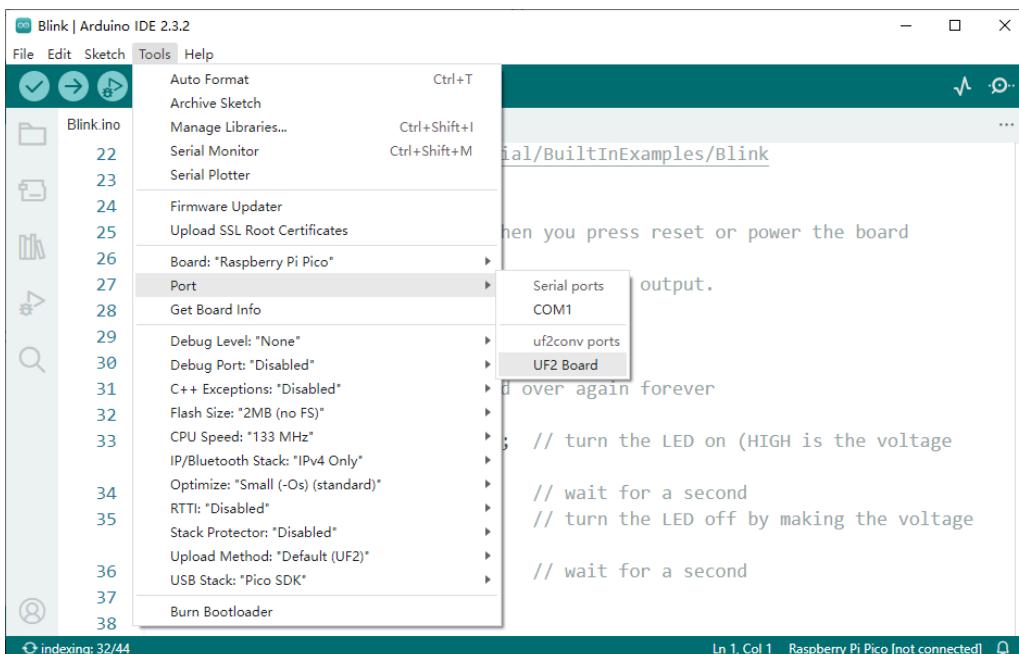
2. Open Arduino IDE. Click File>Examples>01.Basics>Blink.



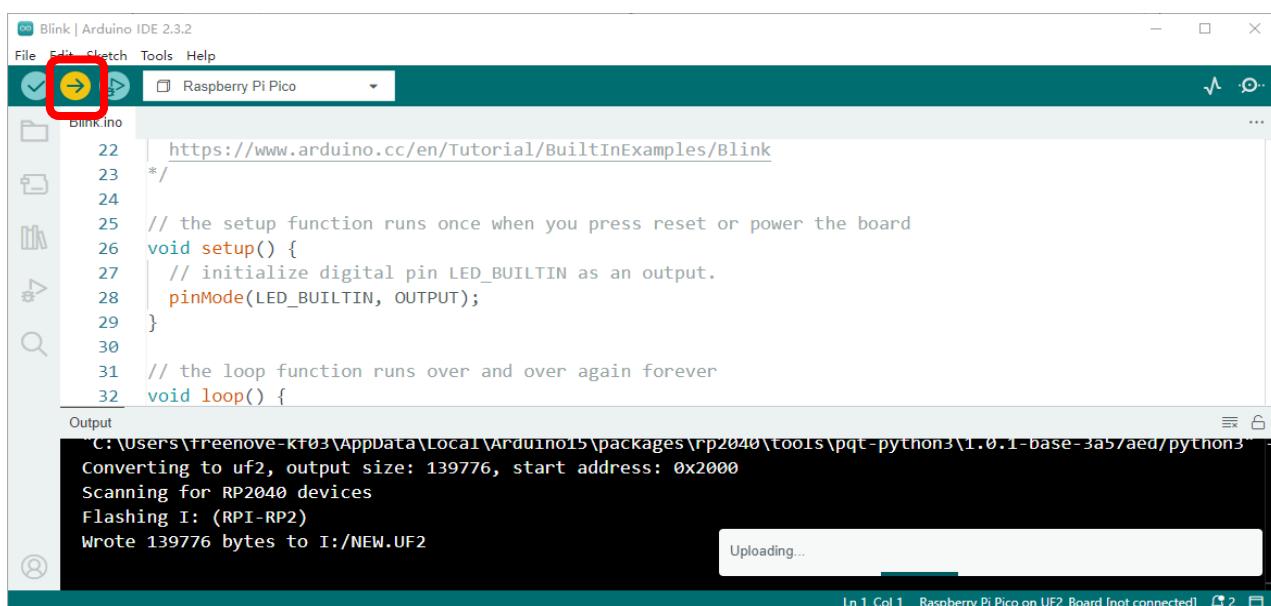
### 3. Click Tools>Board>Raspberry Pi RP2040 Boards>Raspberry Pi Pico.



### 4. Click Tools>Port>UF2 Board.



### 5. Upload sketch to Pico.



Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)



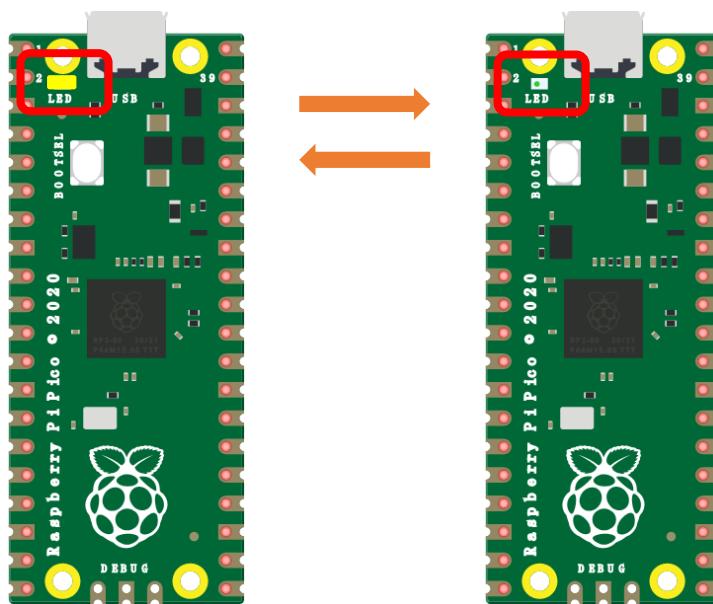
When the sketch finishes uploading, you can see the following prompt.

The screenshot shows the Arduino IDE's Serial Monitor window. The title bar says "Output". The main area displays the following text:

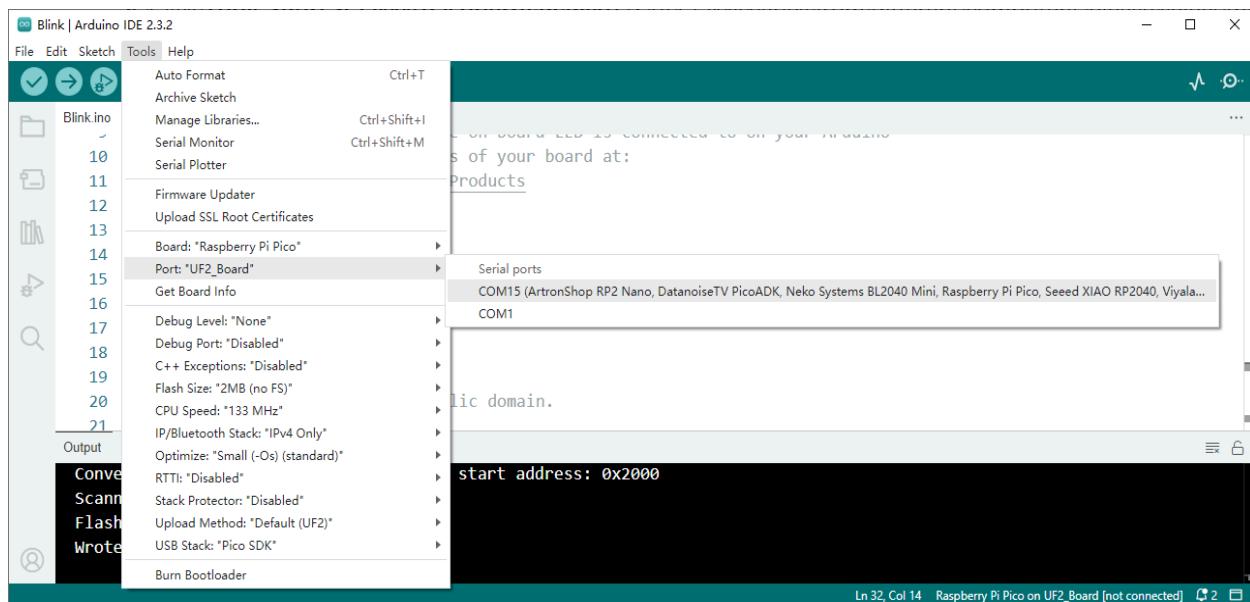
```
Converting to uf2, output size: 139776, start address: 0x2000
Scanning for RP2040 devices
Flashing I: (RPI-RP2)
Wrote 139776 bytes to I:/NEW.UF2
```

At the bottom right of the window, it says "Ln 32, Col 14" and "Raspberry Pi Pico on UF2\_Board [not connected]". There are also some small icons for copy, paste, and close.

And you can see the indicator on Pico starts to flash.



5. Click **Tools>Port>COMx(Raspberry Pi Pico)**. X of COMx varies from different computers. Please select the correct one on your computer. In our case, it is COM15.

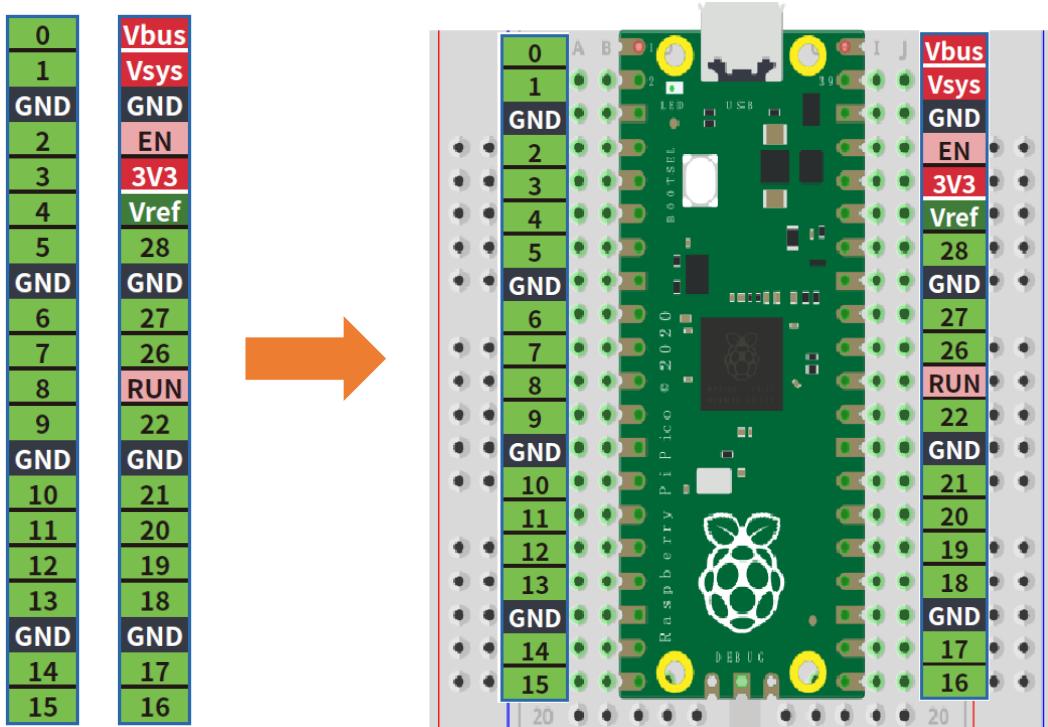


#### Note:

1. At the first time you use Arduino to upload sketch for Pico, you do not need to select port. After that, each time before uploading sketch, please check whether the port has been selected; otherwise, the downloading may fail.
2. Sometimes when using, Pico may lose firmware due to the code and fail to work. At this point, you can upload firmware for Pico as mentioned above.

## Paste the Sticker on the Breadboard

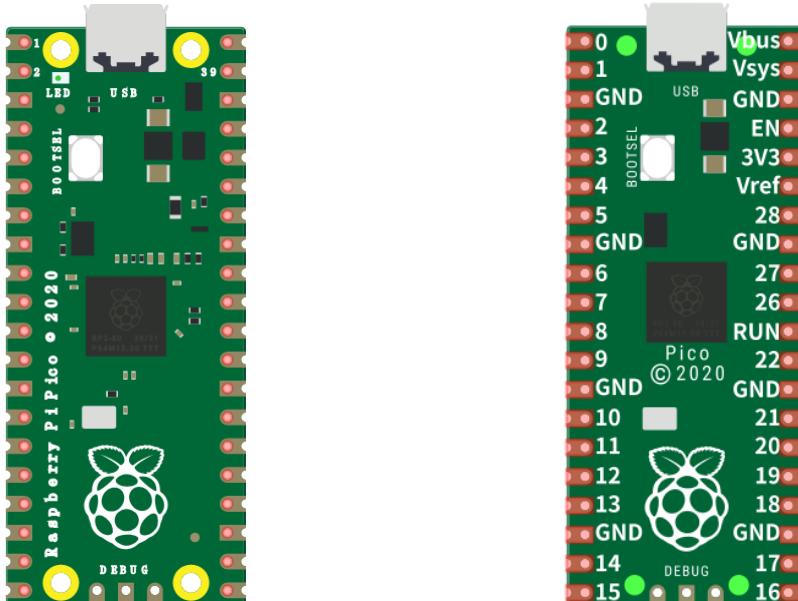
It is not difficult to use the Pico. However, officially, the pin functions are printed on the back of the board, which makes it inconvenient to use. To help users finish each project in the tutorial faster and easier, we provide stickers of the pin functions as follows:



You can paste the sticker on the blank area of the breadboard as above.

To make the tutorial more intuitive, we have made some changes to the simulation diagram as below. The left one is the actual Pico and the right one is its simulation diagram. Please note that to avoid misunderstanding.

In addition, the external pin interface functions of Pico, Pico W and Pico 2 are identical.



Any concerns? ✉ support@freenove.com

# Chapter 1 LED (Important)

Note:

Raspberry Pi Pico, Raspberry Pi Pico W and Raspberry Pi Pico 2 only differ by one wireless function, and are almost identical in other aspects. In this tutorial, except for the wireless function, other parts use Raspberry Pi Pico's map for tutorial demonstration.

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

## Project 1.1 Blink

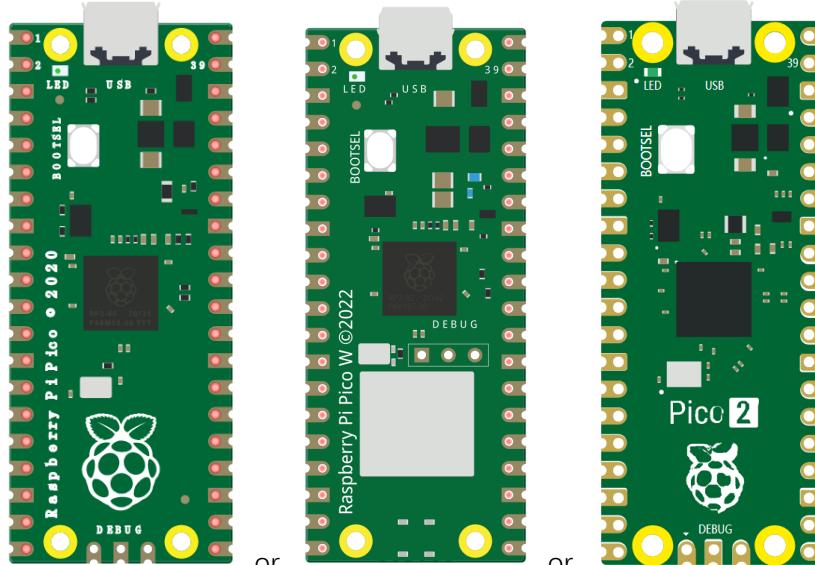
In this project, we will use Raspberry Pi Pico to control blinking a common LED.

If you have not installed Arduino IDE, you can click [Here](#).

If you have not uploaded firmware for Pico, you can click [Here](#) to upload.

## Component List

Raspberry Pi Pico (or Pico W or Pico 2) x1



USB cable x1

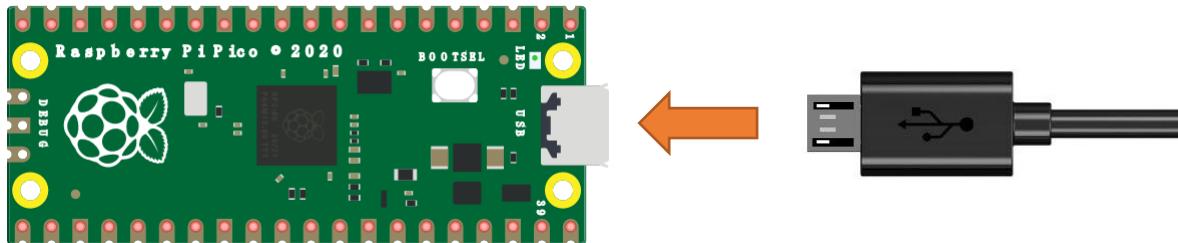




## Power

Raspberry Pi Pico requires 5V power supply. You can either connect external 5V power supply to Vsys pin of Pico or connect a USB cable to the onboard USB base to power Pico.

In this tutorial, we use USB cable to power Pico and upload sketches.



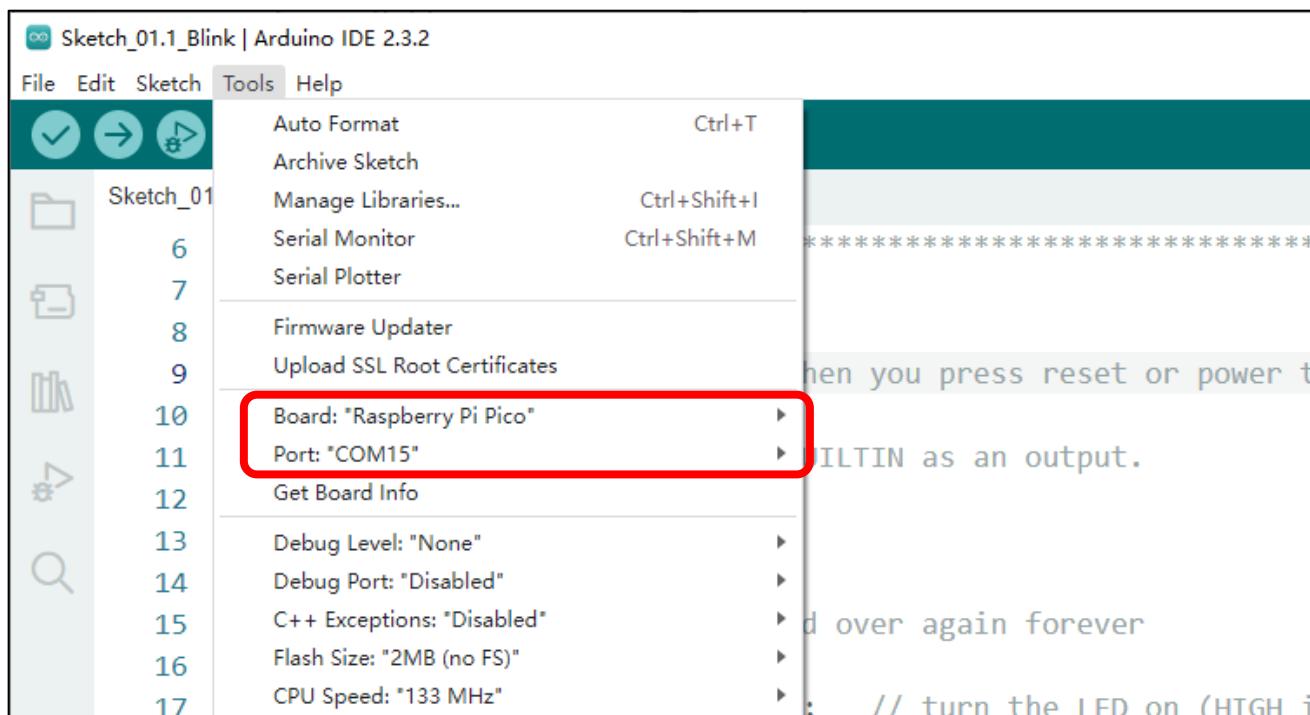
## Sketch

The onboard LED of Raspberry Pi Pico is controlled by GP25. When GP25 outputs high level, LED lights up; when it outputs low, LED lights off. You can open the provided code:

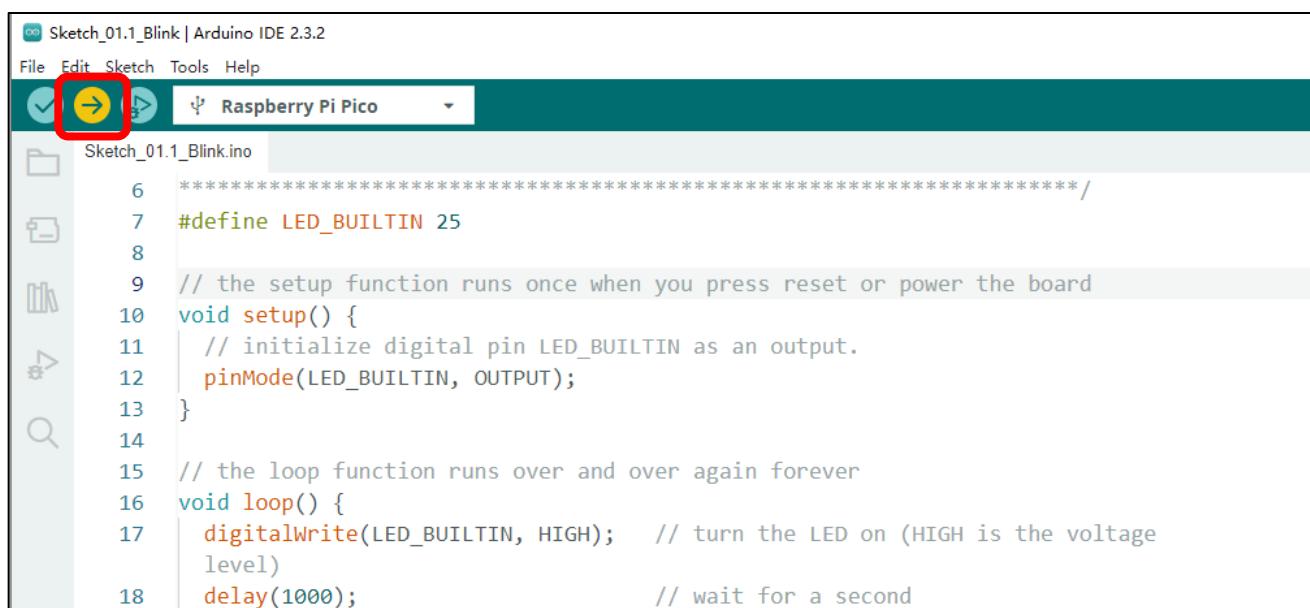
**Freenove\_Basic\_Starter\_Kit\_for\_Raspberry\_Pi\_Pico\Sketches\Sketch\_01.1\_Blink.**

Before uploading code to Pico, please check the configuration of Arduino IDE.

Click Tools, make sure Board and Port are as follows:



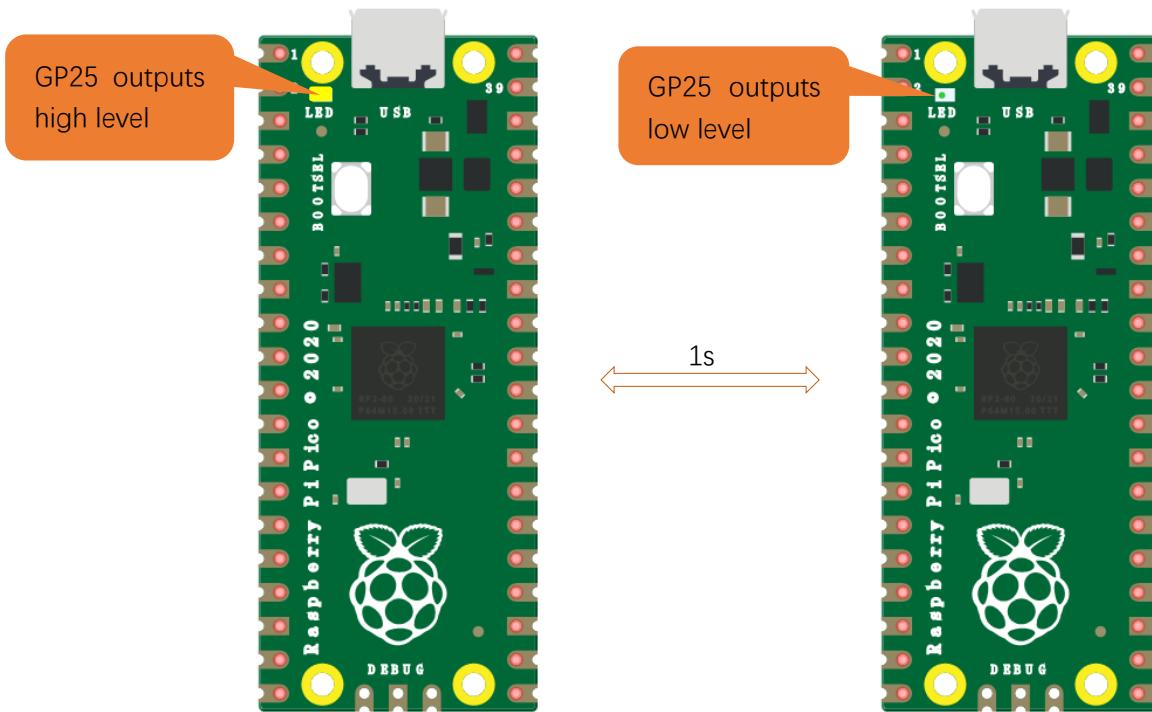
Click "Upload" to upload the sketch to Pico.





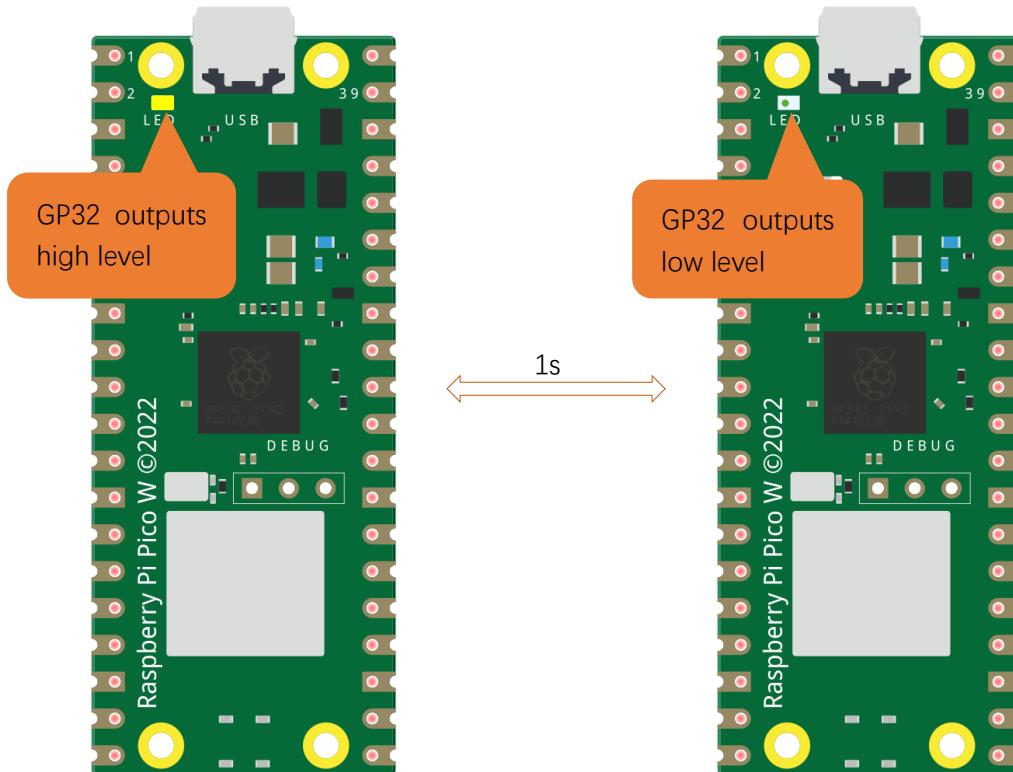
If you have any concerns, please contact us at [support@freenove.com](mailto:support@freenove.com)

Pico's on-board LED lights on and off every 1s, flashing cyclically.



Note: Pico's on-board LED is driven by GPIO25. Pico W's on-board LED uses WL\_GPIO0, which is defined as GPIO32 on Arduino.

If you use Pico W, please change "# define LED\_BUILTIN 25" to "# define LED\_BUILTIN 32" in the code.



Any concerns? ✉ [support@freenove.com](mailto:support@freenove.com)

The following is the program code:

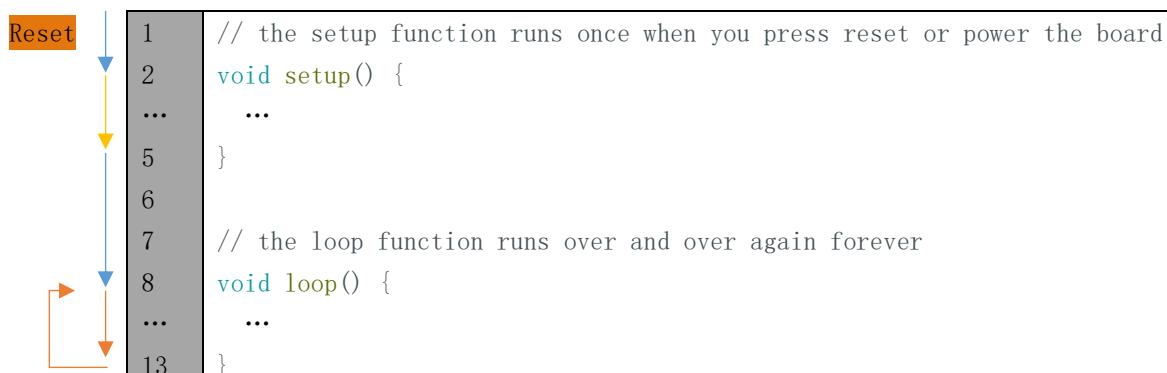
```

1 #define LED_BUILTIN 25
2
3 // the setup function runs once when you press reset or power the board
4 void setup() {
5     // initialize digital pin LED_BUILTIN as an output.
6     pinMode(LED_BUILTIN, OUTPUT);
7 }
8
9 // the loop function runs over and over again forever
10 void loop() {
11     digitalWrite(LED_BUILTIN, HIGH);      // turn the LED on (HIGH is the voltage level)
12     delay(1000);                      // wait for a second
13     digitalWrite(LED_BUILTIN, LOW);     // turn the LED off by making the voltage LOW
14     delay(1000);                      // wait for a second
15 }
```

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 back to the beginning of loop() to run again.



In the circuit, GP25 of Pico is connected to the LED, so the LED pin is defined as 25.

```
1 #define LED_BUILTIN 25
```

This means that after this line of code, all LED\_BUILTIN will be regarded as 25.

In the setup() function, first, we set the LED\_BUILTIN as output mode, which can make the port output high or low level.

```

4 // initialize digital pin LED_BUILTIN as an output.
5 pinMode(LED_BUILTIN, OUTPUT);
```

Then, in the loop() function, set the LED\_BUILTIN to output high level to make LED light up.

```
10 digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (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 LED\_BUILTIN to output low level, and LED lights off. One second later, the execution of loop() function will be completed.

```
12     digitalWrite(LED_BUILTIN, LOW); // turn the LED off 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 as either an input or an output.

### Parameters

pin: the pin number to set the mode of LED.

mode: INPUT, OUTPUT, INPUT\_PULLDOWN, or INPUT\_PULLUP.

**void digitalWrite (int pin, int value);**

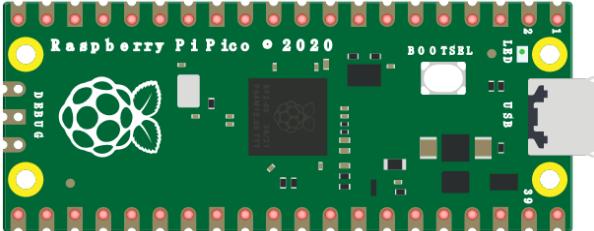
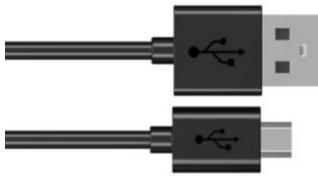
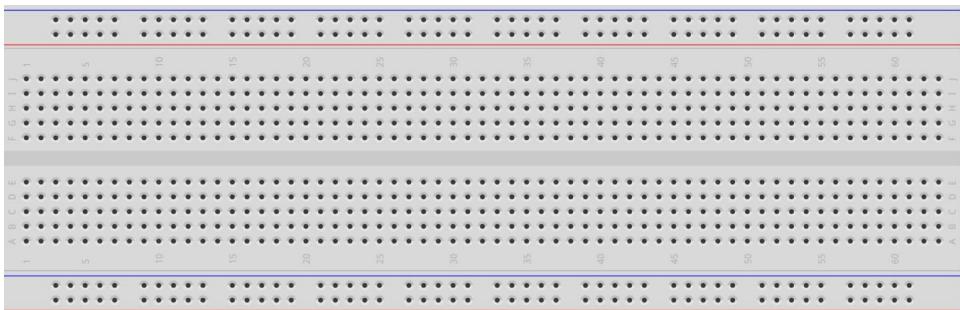
Writes the value HIGH or LOW (1 or 0) to the given pin that must have been previously set as an output.

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

## Project 1.2 Blink

In this project, we will use Raspberry Pi Pico to control blinking a common LED.

### Component List

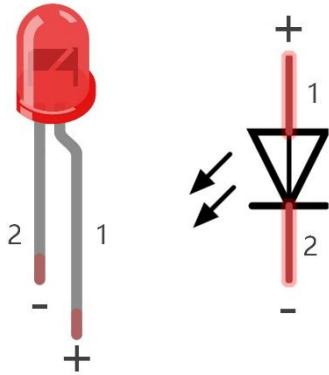
Raspberry Pi Pico x1	USB Cable x1
	
Breadboard x1	
	
LED x1	Resistor 220Ω x1
	
	Jumper
	

## Component Knowledge

### LED

An LED is a type of diode. All diodes only work if current is flowing in the correct direction and have two Poles. An LED will only work (light up) if the longer pin (+) of LED is connected to the positive output from a power source and the shorter pin is connected to the negative (-). Negative output is also referred to as Ground (GND). This type of component is known as "Polar" (think One-Way Street).

All common two-lead diodes are the same in this respect. Diodes work only if the voltage of its positive electrode is higher than its negative electrode and there is a narrow range of operating voltage for most all common diodes of 1.9 and 3.4V. If you use much more than 3.3V the LED will be damaged and burn out.



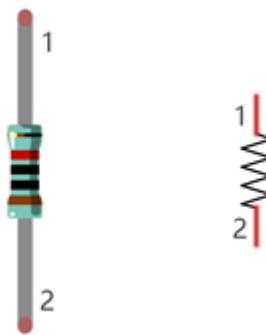
LED	Voltage	Maximum current	Recommended current
Red	1.9 - 2.2V	20mA	10mA
Green	2.9 - 3.4V	10mA	5mA
Blue	2.9 - 3.4V	10mA	5mA
Volt ampere characteristics conform to diode			

Note: LEDs cannot be directly connected to a power supply, which usually ends in a damaged component. A resistor with a specified resistance value must be connected in series to the LED you plan to use.

### Resistor

Resistors use Ohms ( $\Omega$ ) as the unit of measurement of their resistance (R).  $1M\Omega=1000k\Omega$ ,  $1k\Omega=1000\Omega$ .

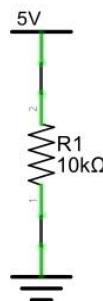
A resistor is a passive electrical component that limits or regulates the flow of current in an electronic circuit. On the left, we see a physical representation of a resistor, and the right is the symbol used to represent the presence of a resistor in a circuit diagram or schematic.



The bands of color on a resistor is a shorthand code used to identify its resistance value. For more details of resistor color codes, please refer to the appendix of this tutorial.

With a fixed voltage, there will be less current output with greater resistance added to the circuit. The relationship between Current, Voltage and Resistance can be expressed by this formula:  $I=V/R$  known as Ohm's Law where I = Current, V = Voltage and R = Resistance. Knowing the values of any two of these allows you to solve the value of the third.

In the following diagram, the current through R1 is:  $I=U/R=5V/10k\Omega=0.0005A=0.5mA$ .



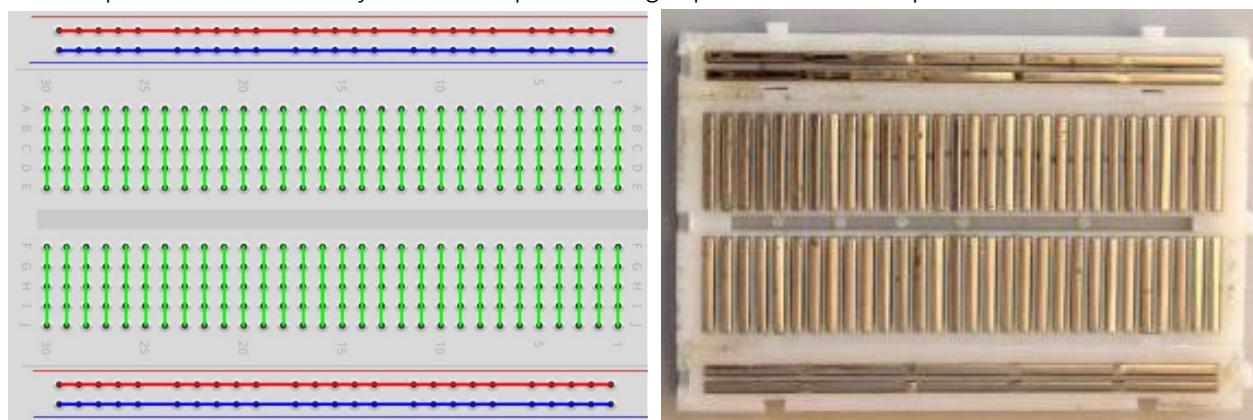
**WARNING:** Never connect the two poles of a power supply with anything of low resistance value (i.e. a metal object or bare wire) this is a Short and results in high current that may damage the power supply and electronic components.

Note: Unlike LEDs and Diodes, Resistors have no poles and are non-polar (it does not matter which direction you insert them into a circuit, it will work the same)

### Breadboard

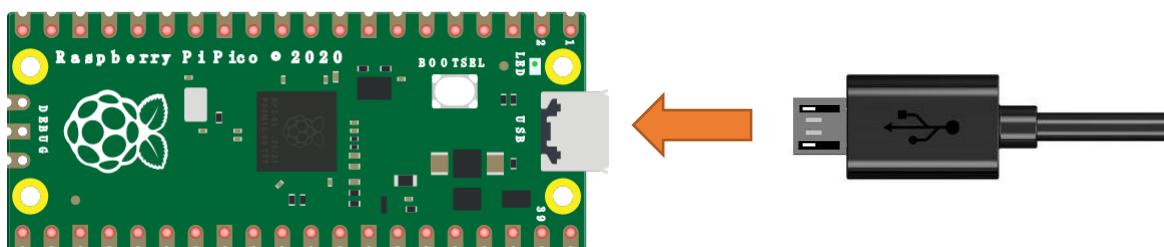
Here we have a small breadboard as an example of how the rows of holes (sockets) are electrically attached.

The left picture shows the way to connect pins. The right picture shows the practical internal structure.



### Power

In this tutorial, we connect Raspberry Pi Pico and computer with a USB cable.



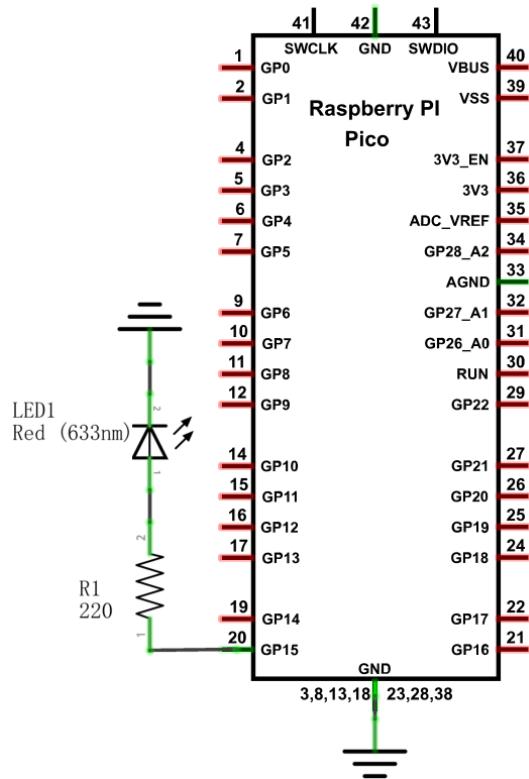
## Circuit

First, disconnect all power from the Raspberry Pi Pico. Then build the circuit according to the circuit and hardware diagrams. After the circuit is built and verified correct, connect the PC to Raspberry Pi Pico.

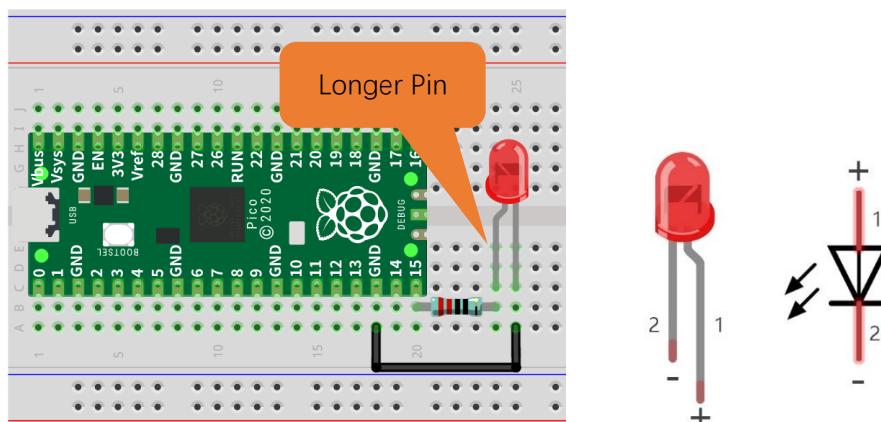
**CAUTION:** Avoid any possible short circuits (especially connecting 3.3V and GND)!

**WARNING:** A short circuit can cause high current in your circuit, create excessive component heat and cause permanent damage to your hardware!

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: [support@freenove.com](mailto:support@freenove.com)



Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

Any concerns? ✉ [support@freenove.com](mailto:support@freenove.com)

## Sketch

According to the circuit diagram, when GP15 of Pico outputs high level, LED lights up; when it outputs low, LED lights off. Therefore, we can make LED flash repeatedly by controlling GP15 to output high and low repeatedly.

You can open the provided code:

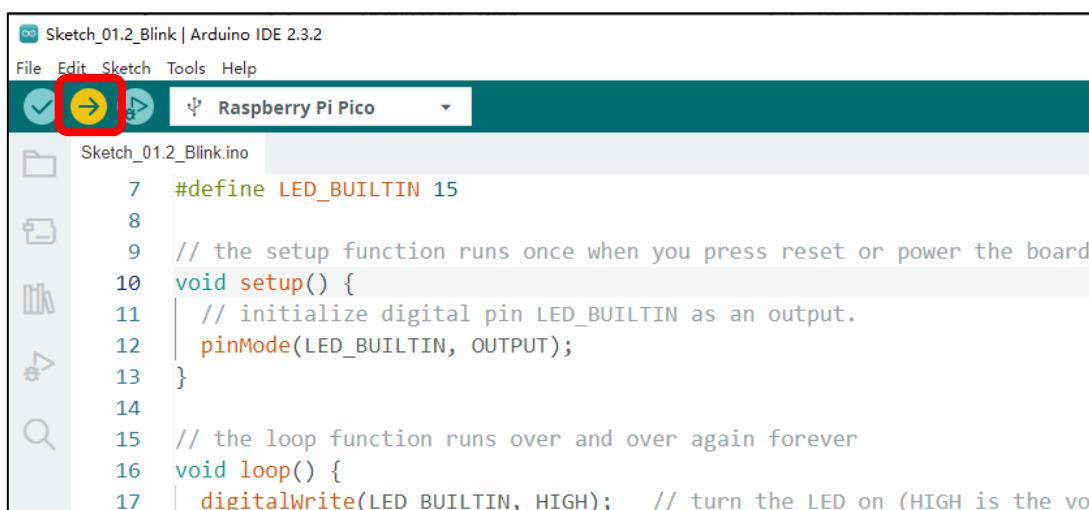
**Freenove\_Basic\_Starter\_Kit\_for\_Raspberry\_Pi\_Pico\Sketches\Sketch\_01.2\_Blink.**

Before uploading code to Pico, please check the configuration of Arduino IDE.

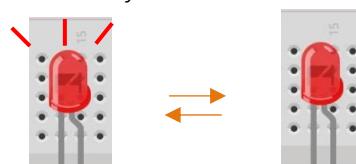
Click Tools, make sure Board and Port are as follows:



Click "Upload" to upload the sketch to Pico.



Click "Upload". Download the code to Pico and your LED in the circuit starts Blink.

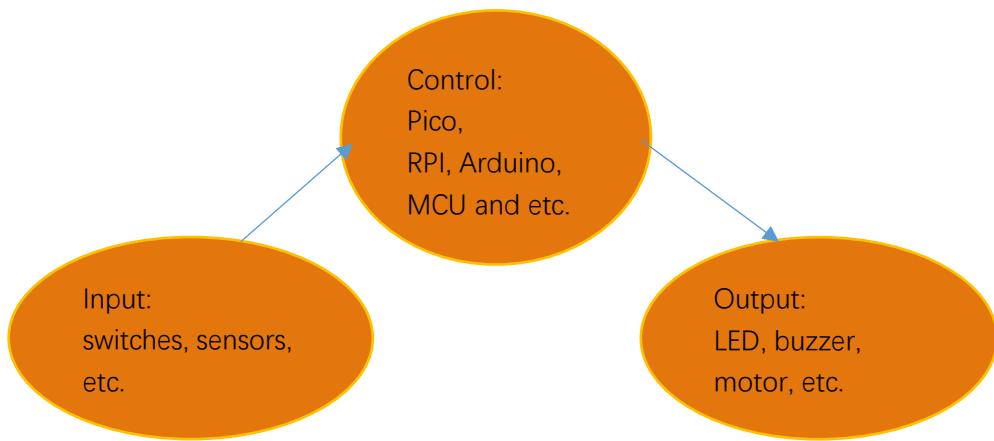


If you have any concerns, please contact us via: [support@freenove.com](mailto:support@freenove.com)

Any concerns? ✉ [support@freenove.com](mailto:support@freenove.com)

# Chapter 2 Button & LED

Usually, there are three essential parts in a complete automatic control device: INPUT, OUTPUT, and CONTROL. In last section, the LED module was the output part and Raspberry Pi Pico was the control part. In practical applications, we not only make LEDs blink, but also make a device sense the surrounding environment, receive instructions and then take the appropriate action such as lighting up LEDs, turning ON a buzzer and so on.



Next, we make a simple project: build a control system with button, LED and Raspberry Pi Pico.

Input: Button

Control: Raspberry Pi Pico

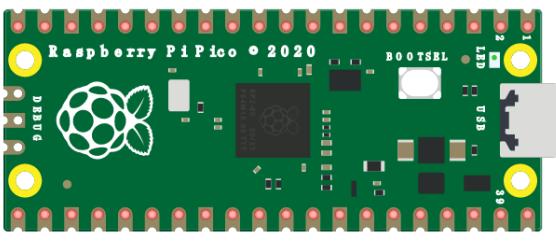
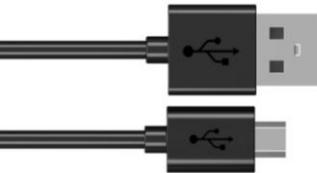
Output: LED

## Project 2.1 Button & LED

Note: Raspberry Pi Pico, Raspberry Pi Pico W and Raspberry Pi Pico 2 only differ by wireless function, and are almost identical in other aspects. In this tutorial, except for the wireless function, other parts use Raspberry Pi Pico's map for tutorial demonstration.

In the project, we will control the LED state through a Push Button Switch. When the button is pressed, our LED will turn ON, and when it is released, the LED will turn OFF.

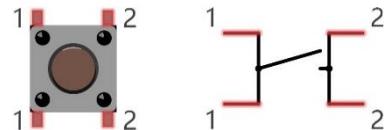
### Component List

Raspberry Pi Pico x1		USB cable x1							
Breadboard x1									
Jumper		LED x1		Resistor 220Ω x1		Resistor 10kΩ x2		Push button x1	

## Component Knowledge

### Push button

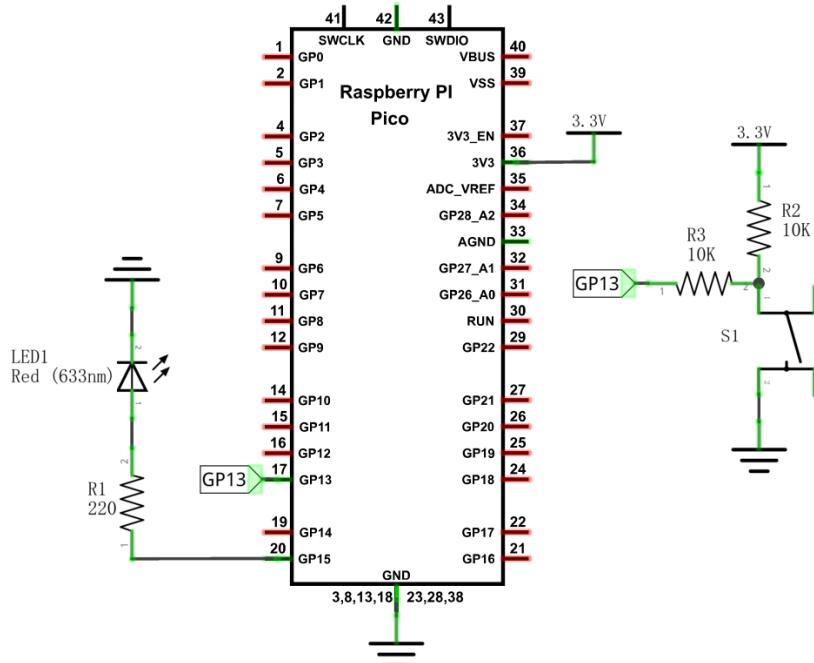
This type of Push Button Switch has four pins (2 Pole Switch). Two pins on the left are connected, and both left and right sides are the same per the illustration:



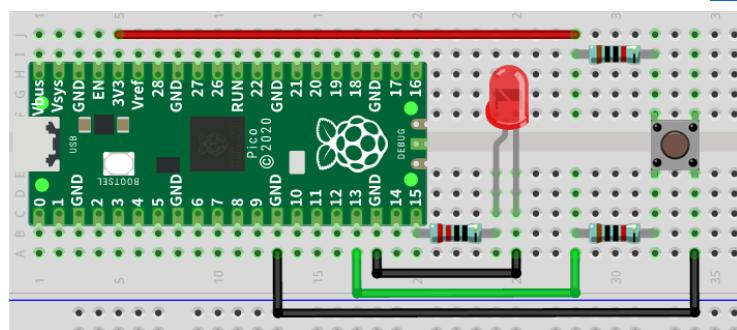
When the button on the switch is pressed, the circuit is completed (your project is Powered ON).

## Circuit

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: [support@freenove.com](mailto:support@freenove.com)



Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

**Any concerns? ✉ [support@freenove.com](mailto:support@freenove.com)**

## Sketch

This project is designed for learning how to use push button switch to control an LED. We first need to read the state of switch, and then determine whether to turn the LED ON in accordance to the state of the switch. Upload following sketch:

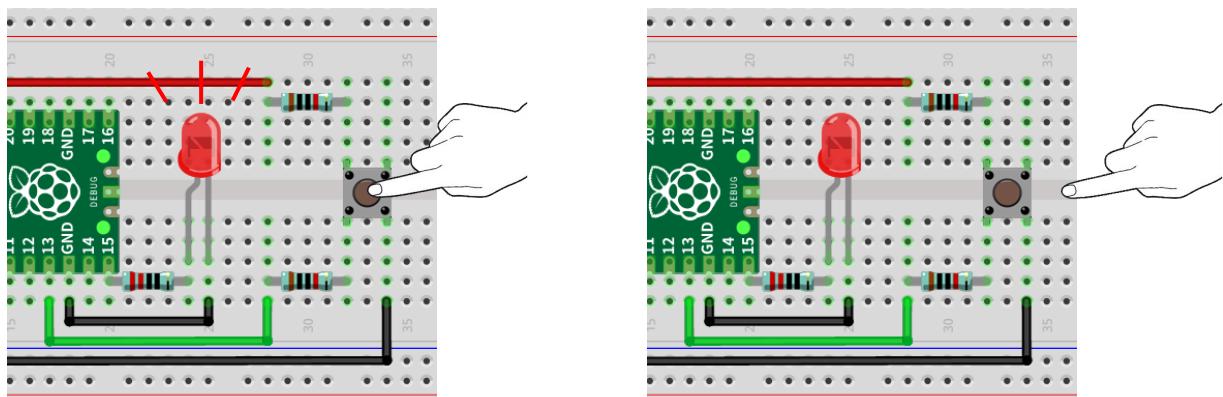
**Freenove\_Basic\_Starter\_Kit\_for\_Raspberry\_Pi\_Pico\Sketches\Sketch\_02.1\_ButtonAndLed.**

**Sketch\_02.1\_ButtonAndLed**



```
Sketch_02.1_ButtonAndLed | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Sketch_02.1_ButtonAndLed.ino
1 #define PIN_LED    15
2 #define PIN_BUTTON 13
3
4 void setup() {
5     // initialize digital pin PIN_LED as an output.
6     pinMode(PIN_LED, OUTPUT);
7     pinMode(PIN_BUTTON, INPUT);
8 }
9
10 // the loop function runs over and over again forever
11 void loop() {
12     if (digitalRead(PIN_BUTTON) == LOW) {
13         digitalWrite(PIN_LED,HIGH);
14     }else{
15         digitalWrite(PIN_LED,LOW);
16     }
17 }
```

Upload the sketch to Pico. When pressing the button, LED lights up; when releasing the button, LED lights OFF.





The following is the program code:

```

1 #define PIN_LED    15
2 #define PIN_BUTTON 13
3 // the setup function runs once when you press reset or power the board
4 void setup() {
5     // initialize digital pin PIN_LED as an output.
6     pinMode(PIN_LED, OUTPUT);
7     pinMode(PIN_BUTTON, INPUT);
8 }
9
10 // the loop function runs over and over again forever
11 void loop() {
12     if (digitalRead(PIN_BUTTON) == LOW) {
13         digitalWrite(PIN_LED, HIGH);
14     }else{
15         digitalWrite(PIN_LED, LOW);
16     }
17 }
```

In the circuit connection, LED and button are connected with GP15 and GP13 respectively, so define ledPin and buttonPin as 15 and 13 respectively.

```

1 #define PIN_LED    15
2 #define PIN_BUTTON 13
```

In the while cycle of main function, use digitalRead(buttonPin) to determine the state of button. When the button is pressed, the function returns low level and the result of "if" is true, so LED lights up. Otherwise, LED lights OFF.

```

11 void loop() {
12     if (digitalRead(PIN_BUTTON) == LOW) {
13         digitalWrite(PIN_LED, HIGH);
14     }else{
15         digitalWrite(PIN_LED, LOW);
16     }
17 }
```

### Reference

```
int digitalRead (int pin);
```

This function returns the value read at the given pin. It will be "HIGH" or "LOW" (1 or 0) depending on the logic level at the pin.

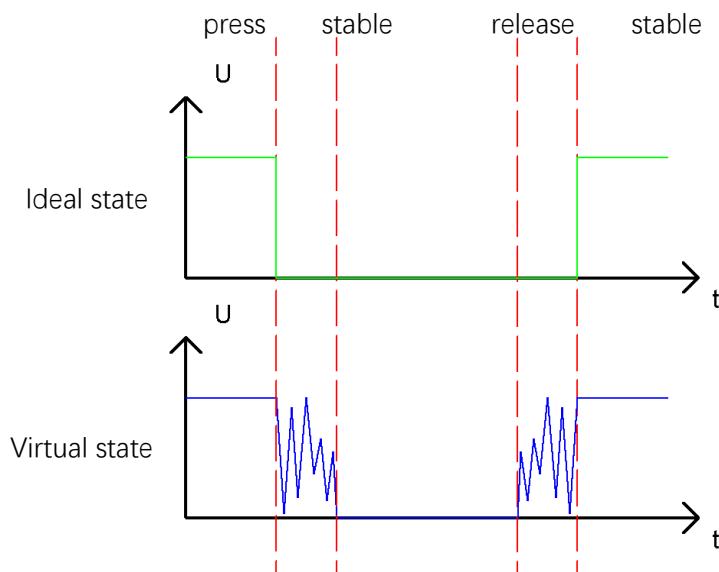
## Project 2.2 MINI table lamp

We will also use a Push Button Switch, LED and Raspberry Pi Pico to make a MINI Table Lamp but this will function differently: Press the button, the LED will turn ON, and pressing the button again, the LED turns OFF. The ON switch action is no longer momentary (like a door bell) but remains ON without needing to continually press on the Button Switch.

First, let us learn something about the push button switch.

### Debounce for Push Button

When a Momentary Push Button Switch is pressed, it will not change from one state to another state immediately. Due to tiny mechanical vibrations, there will be a short period of continuous buffeting before it completely reaches another state too fast for Humans to detect but not for computer microcontrollers. The same is true when the push button switch is released. This unwanted phenomenon is known as “bounce”.



Therefore, if we can directly detect the state of the Push Button Switch, there are multiple pressing and releasing actions in one pressing cycle. This buffeting will mislead the high-speed operation of the microcontroller to cause many false decisions. Therefore, we need to eliminate the impact of buffeting. Our solution: to judge the state of the button multiple times. Only when the button state is stable (consistent) over a period of time, can it indicate that the button is actually in the ON state (being pressed).  
This project needs the same components and circuits as we used in the previous section.

## Sketch

Upload following sketch:

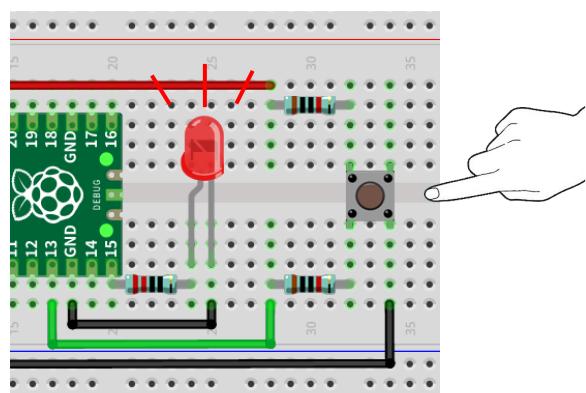
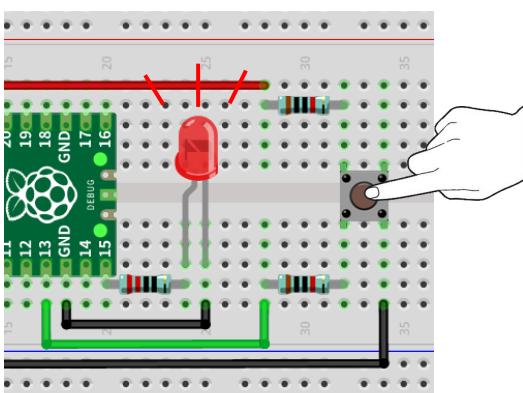
**Freenove\_Basic\_Starter\_Kit\_for\_Raspberry\_Pi\_Pico\Sketches\Sketch\_02.2\_TableLamp.**

**Sketch\_02.2\_TableLamp**

```

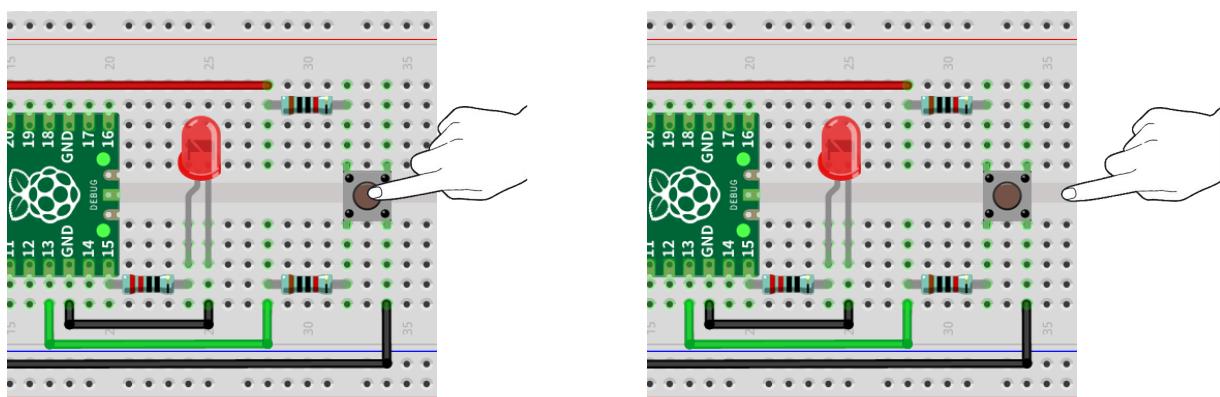
Sketch_02.2_TableLamp | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Sketch_02.2_TableLamp.ino
1 #define PIN_LED 15
2 #define PIN_BUTTON 13
3 bool ledState = false;
4
5 void setup() {
6     // initialize digital pin PIN_LED as an output.
7     pinMode(PIN_LED, OUTPUT);
8     pinMode(PIN_BUTTON, INPUT);
9 }
10
11 // the loop function runs over and over again forever
12 void loop() {
13     if (digitalRead(PIN_BUTTON) == LOW) {
14         delay(20);
15         if (digitalRead(PIN_BUTTON) == LOW) {
16             reverseGPIO(PIN_LED);
17         }
18         while (digitalRead(PIN_BUTTON) == LOW);
19     }
20 }
21
22 void reverseGPIO(int pin) {
23     ledState = !ledState;
24     digitalWrite(pin, ledState);
25 }
26 }
```

Upload the sketch to Pico. When the button is pressed, LED lights up; when the button is released, LED is still ON.



Any concerns? ✉ support@freenove.com

When the button is pressed again, LED turns OFF; when released, LED keeps OFF.



The following is the program code:

```

1 #define PIN_LED    15
2 #define PIN_BUTTON 13
3 bool ledState = false;
4
5 void setup() {
6     // initialize digital pin PIN_LED as an output.
7     pinMode(PIN_LED, OUTPUT);
8     pinMode(PIN_BUTTON, INPUT);
9 }
10
11 // the loop function runs over and over again forever
12 void loop() {
13     if (digitalRead(PIN_BUTTON) == LOW) {
14         delay(20);
15         if (digitalRead(PIN_BUTTON) == LOW) {
16             reverseGPIO(PIN_LED);
17         }
18         while (digitalRead(PIN_BUTTON) == LOW);
19     }
20 }
21
22 void reverseGPIO(int pin) {
23     ledState = !ledState;
24     digitalWrite(pin, ledState);
25 }
```

In the circuit connection, LED and button are connected with GP15 and GP13 respectively, so define ledPin and buttonPin as 15 and 13 respectively.

```

1 #define PIN_LED    15
2 #define PIN_BUTTON 13
```

Define a variable to store the status of LED.

```
3 bool ledState = false;
```



When judging the push button state, if it is detected as "pressed down", wait for a certain time to detect again to eliminate the effect of bounce. When confirmed, flip the LED on and off. Then it starts to wait for the pressed button to be released, and waits for a certain time to eliminate the effect of bounce after it is released.

```
13  if (digitalRead(PIN_BUTTON) == LOW) {  
14      delay(20);  
15      if (digitalRead(PIN_BUTTON) == LOW) {  
16          reverseGPIO(PIN_LED);  
17      }  
18      while (digitalRead(PIN_BUTTON) == LOW);  
19  }
```

When the button is pressed, reverseGPIO function is called to change the variable that controls LED's statue, and write it to Pico to reverse the pin's output state.

```
22  void reverseGPIO(int pin) {  
23      ledState = !ledState;  
24      digitalWrite(pin, ledState);  
25  }
```

# Chapter 3 LED Bar

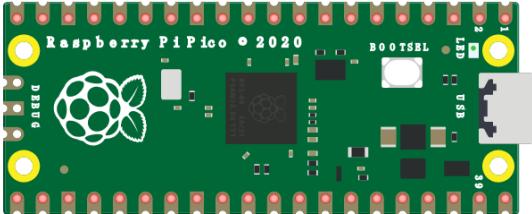
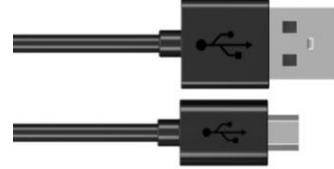
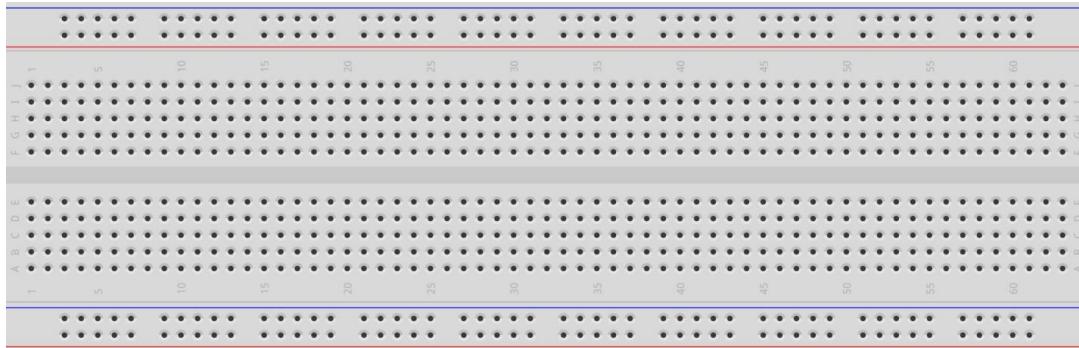
We have learned how to control an LED blinking, next we will learn how to control a number of LEDs.

## Project 3.1 Flowing Light

Note: Raspberry Pi Pico, Raspberry Pi Pico W and Raspberry Pi Pico 2 only differ by wireless function, and are almost identical in other aspects. In this tutorial, except for the wireless function, other parts use Raspberry Pi Pico's map for tutorial demonstration.

In this project, we use a number of LEDs to make a flowing light.

## Component List

Raspberry Pi Pico x1	USB cable x1	
		
Breadboard x1		
Jumper	LED bar graph x1	Resistor 220Ω x10

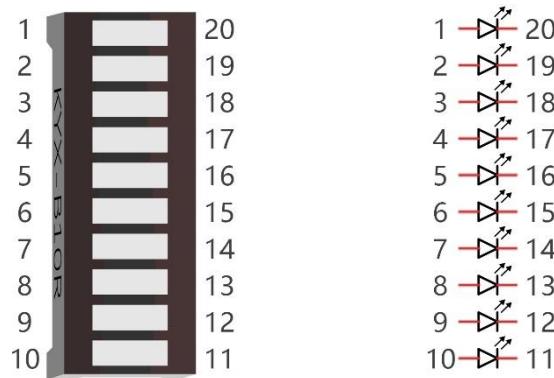


## Component Knowledge

Let us learn about the basic features of these components to use and understand them better.

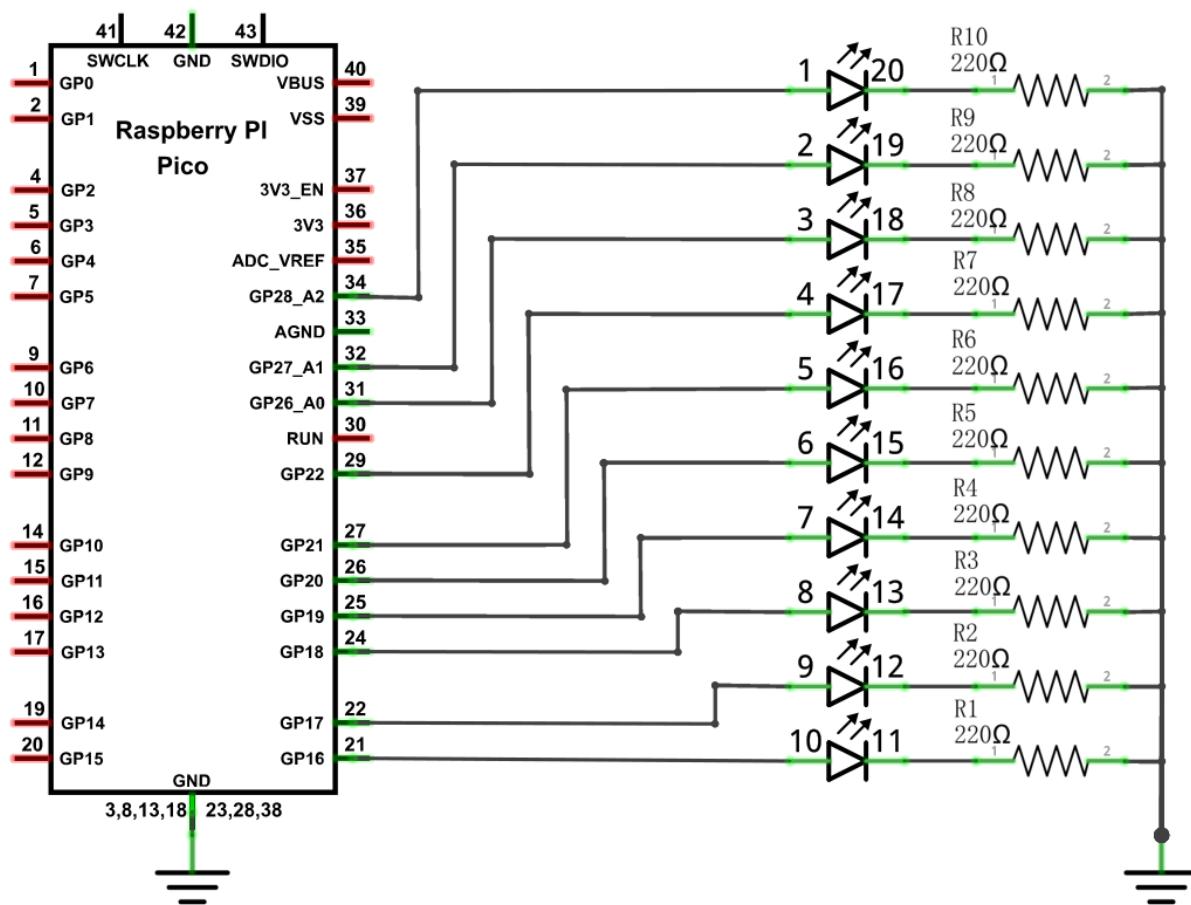
### LED bar

A Bar Graph LED has 10 LEDs integrated into one compact component. The two rows of pins at its bottom are paired to identify each LED like the single LED used earlier.

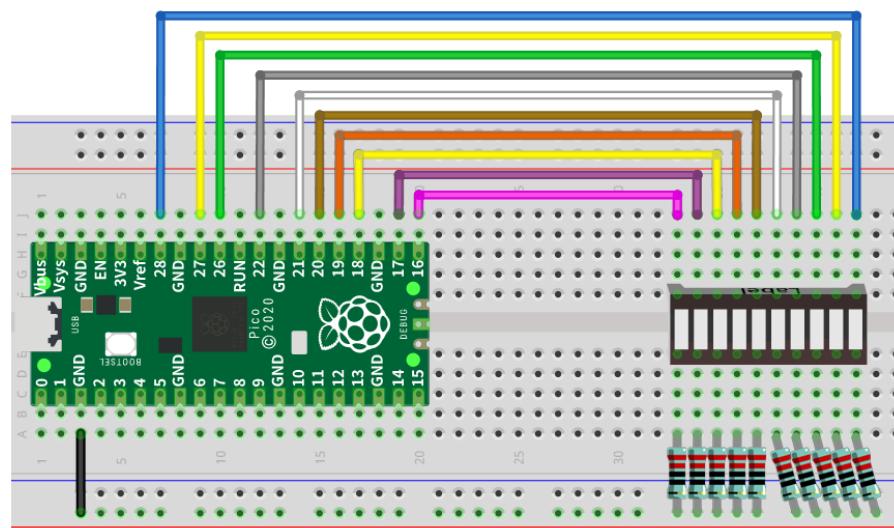


## Circuit

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: [support@freenove.com](mailto:support@freenove.com)



Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

If LEDbar does not work, try to rotate LEDbar for 180°. The label is random.

Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)



## Sketch

This project is designed to make a flowing water lamp, which are these actions: First turn LED #1 ON, and then turn it OFF. Then turn LED #2 ON, and then turn it OFF... and repeat the same to all 10 LEDs until the last LED is turns OFF. This process is repeated to achieve the “movements” of flowing water.

Upload following sketch:

**Freenove\_Basic\_Starter\_Kit\_for\_Raspberry\_Pi\_Pico\Sketches\Sketch\_03.1\_FlowingLight.**

**Sketch\_03.1\_FlowingLight**

```

Sketch_03.1_FlowingLight | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Raspberry Pi Pico
Sketch_03.1_FlowingLight.ino
7 byte ledPins[] = {16, 17, 18, 19, 20, 21, 22, 26, 27, 28};
8 int ledCounts;
9
10 void setup() {
11     ledCounts = sizeof(ledPins);
12     for (int i = 0; i < ledCounts; i++) {
13         pinMode(ledPins[i], OUTPUT);
14     }
15 }
16
17 void loop() {
18     for (int i = 0; i < ledCounts; i++) {
19         digitalWrite(ledPins[i], HIGH);
20         delay(100);
21         digitalWrite(ledPins[i], LOW);
22     }
23     for (int i = ledCounts - 1; i > -1; i--) {
24         digitalWrite(ledPins[i], HIGH);
25         delay(100);
26         digitalWrite(ledPins[i], LOW);
27     }
28 }

```

Click Upload to upload the sketch to Pico. LEDs of LED bar graph lights up one by one from left to right and then back from right to left.



If you have any concerns, please contact us via: [support@freenove.com](mailto:support@freenove.com)

Any concerns? ✉ [support@freenove.com](mailto:support@freenove.com)

The following is the program code:

```
1 byte ledPins[] = {16, 17, 18, 19, 20, 21, 22, 26, 27, 28};  
2 int ledCounts;  
3  
4 void setup() {  
5     ledCounts = sizeof(ledPins);  
6     for (int i = 0; i < ledCounts; i++) {  
7         pinMode(ledPins[i], OUTPUT);  
8     }  
9 }  
10  
11 void loop() {  
12     for (int i = 0; i < ledCounts; i++) {  
13         digitalWrite(ledPins[i], HIGH);  
14         delay(100);  
15         digitalWrite(ledPins[i], LOW);  
16     }  
17     for (int i = ledCounts - 1; i > -1; i--) {  
18         digitalWrite(ledPins[i], HIGH);  
19         delay(100);  
20         digitalWrite(ledPins[i], LOW);  
21     }  
22 }
```

Use an array to define 10 GPIO ports connected to LED bar graph for easier operation.

```
1 byte ledPins[] = {16, 17, 18, 19, 20, 21, 22, 26, 27, 28};
```

In setup(), use sizeof() to get the number of array, which is the number of LEDs, then configure the GPIO port to output mode.

```
5 ledCounts = sizeof(ledPins);  
6 for (int i = 0; i < ledCounts; i++) {  
7     pinMode(ledPins[i], OUTPUT);  
8 }
```

Then, in loop(), use two “for” loop to realize flowing water light from left to right and from right to left.

```
12 for (int i = 0; i < ledCounts; i++) {  
13     digitalWrite(ledPins[i], HIGH);  
14     delay(100);  
15     digitalWrite(ledPins[i], LOW);  
16 }  
17 for (int i = ledCounts - 1; i > -1; i--) {  
18     digitalWrite(ledPins[i], HIGH);  
19     delay(100);  
20     digitalWrite(ledPins[i], LOW);  
21 }
```



# Chapter 4 Analog & PWM

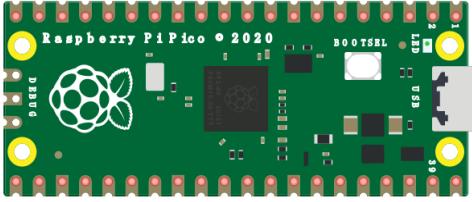
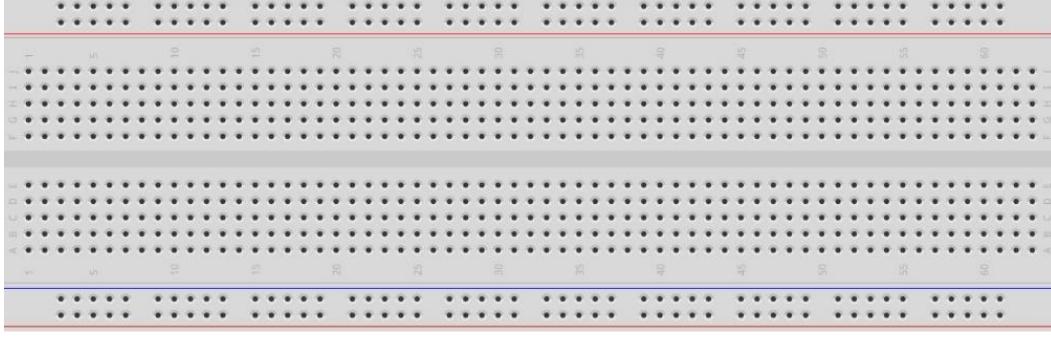
In previous study, we have known that one button has two states: pressed and released, and LED has light-on/off state, then how to enter a middle state? How to output an intermediate state to let LED "semi bright"? That is what we are going to learn.

First, let us learn how to control the brightness of an LED.

## Project 4.1 Breathing LED

Breathing light, that is, LED is turned from off to on gradually and gradually from on to off, just like "breathing". So, how to control the brightness of an LED? We will use PWM to achieve this target.

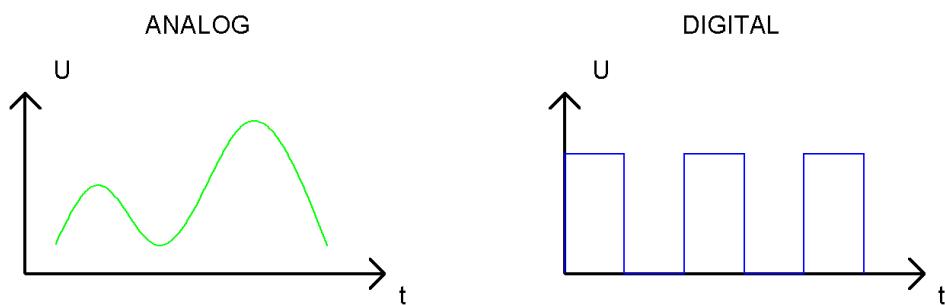
### Component List

Raspberry Pi Pico x1		USB cable x1	
Breadboard x1			
LED x1		Resistor 220Ω x1	Jumper

## Related Knowledge

### Analog & Digital

An Analog Signal is a continuous signal in both time and value. On the contrary, a Digital Signal or discrete-time signal is a time series consisting of a sequence of quantities. Most signals in life are analog signals. A familiar example of an Analog Signal would be how the temperature throughout the day is continuously changing and could not suddenly change instantaneously from 0°C to 10°C. However, Digital Signals can instantaneously change in value. This change is expressed in numbers as 1 and 0 (the basis of binary code). Their differences can more easily be seen when compared when graphed as below.



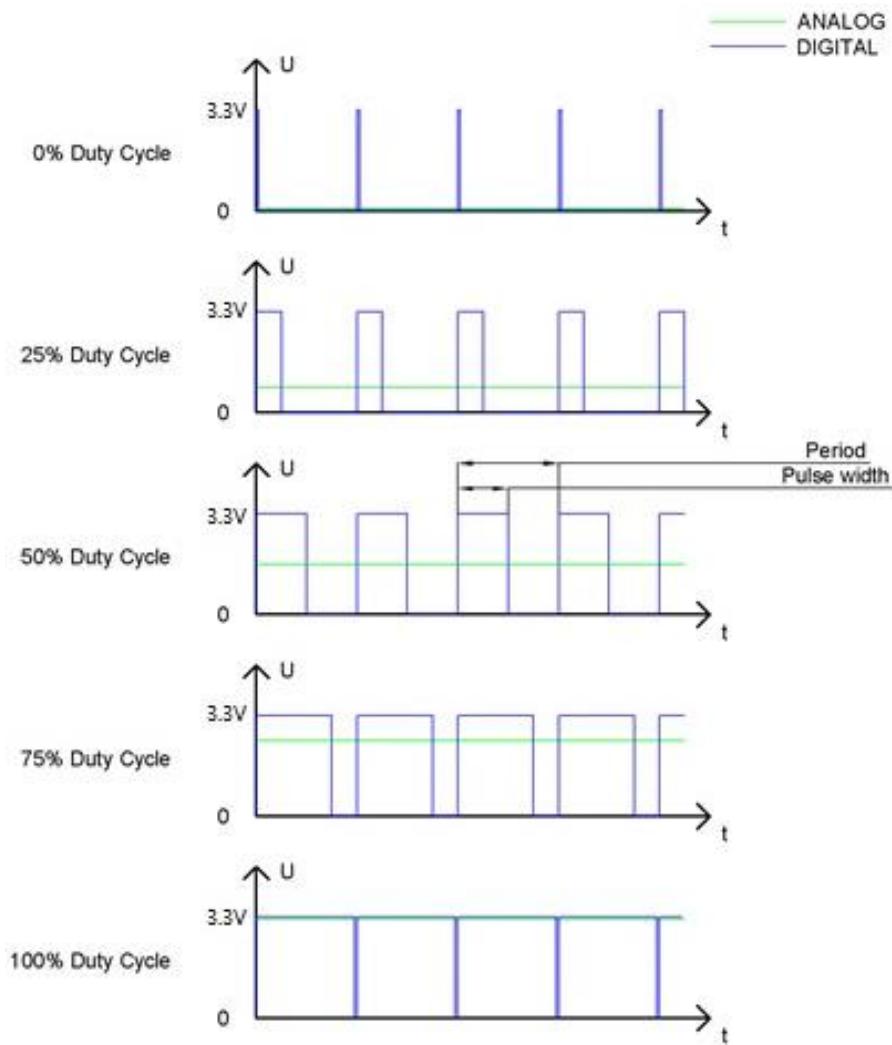
In practical application, we often use binary as the digital signal, that is a series of 0's and 1's. Since a binary signal only has two values (0 or 1), it has great stability and reliability. Lastly, both analog and digital signals can be converted into the other.

### PWM

PWM, Pulse-Width Modulation, is a very effective method for using digital signals to control analog circuits. Common processors cannot directly output analog signals. PWM technology makes it very convenient to achieve this conversion (translation of digital to analog signals)

PWM technology uses digital pins to send certain frequencies of square waves, that is, the output of high levels and low levels, which alternately last for a while. The total time for each set of high levels and low levels is generally fixed, which is called the period (Note: the reciprocal of the period is frequency). The time of high level outputs are generally called "pulse width", and the duty cycle is the percentage of the ratio of pulse duration, or pulse width (PW) to the total period (T) of the waveform.

The longer the output of high levels last, the longer the duty cycle and the higher the corresponding voltage in the analog signal will be. The following figures show how the analog signal voltages vary between 0V-5V (high level is 5V) corresponding to the pulse width 0%-100%:



The longer the PWM duty cycle is, the higher the output power will be. Now that we understand this relationship, we can use PWM to control the brightness of an LED or the speed of DC motor and so on. It is evident from the above that PWM is not real analog, and the effective value of the voltage is equivalent to the corresponding analog. Therefore, we can control the output power of the LED and other output modules to achieve different effects.

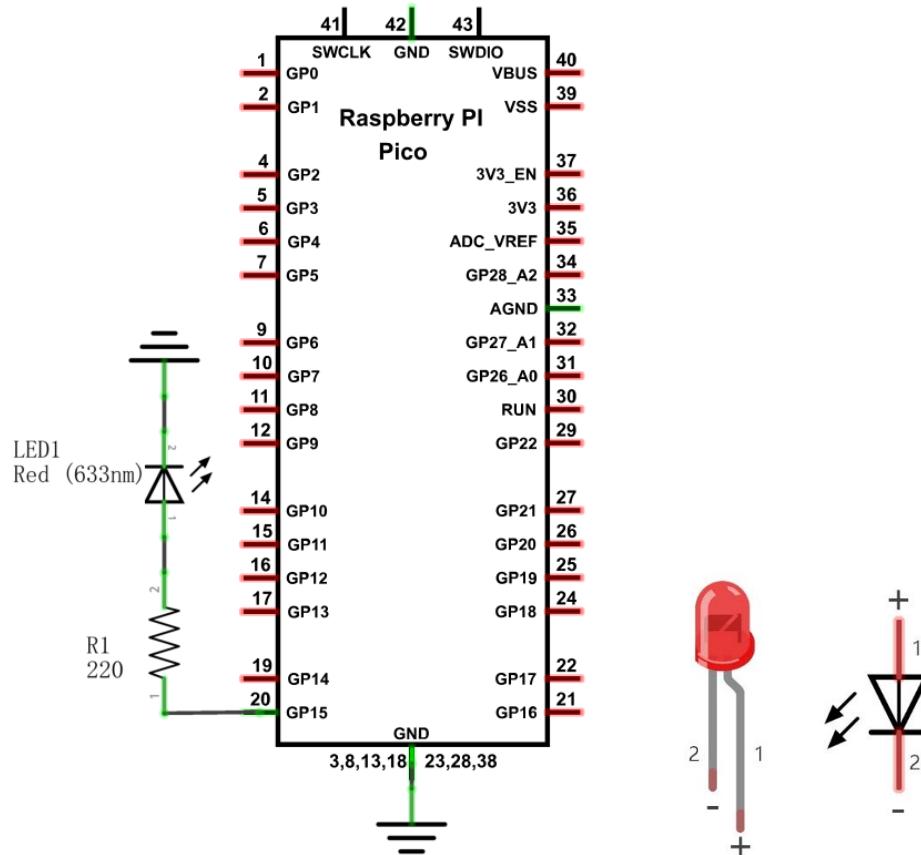
### Raspberry Pi Pico and PWM

Raspberry Pi Pico has 16 PWM channels, each of which can control frequency and duty cycle independently. Every pin on Raspberry Pi Pico can be configured as PWM output. In Arduino, PWM frequency is set to 500Hz. You can change the PWM output by changing duty cycle.

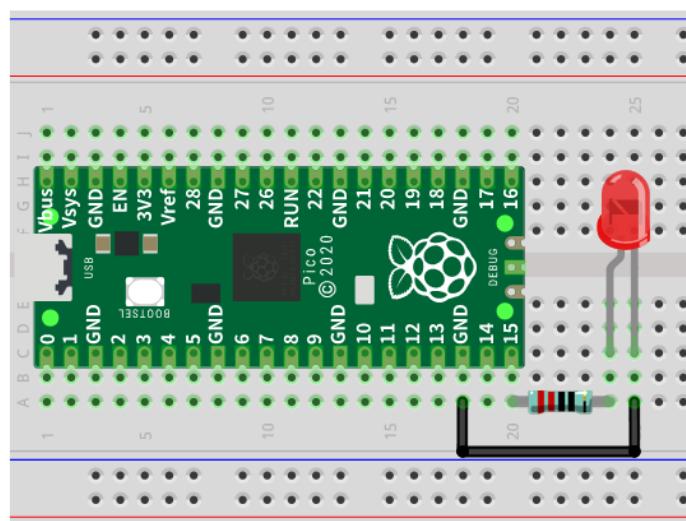
## Circuit

This circuit is the same as the one in project Blink.

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: [support@freenove.com](mailto:support@freenove.com)



Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.



## Sketch

This project is designed to make PWM output GP15 with pulse width increasing from 0% to 100%, and then reducing from 100% to 0% gradually.

### Sketch\_04.1\_BreathingLight

The screenshot shows the Arduino IDE interface with the following details:

- Title Bar:** Sketch\_04.1\_BreathingLight | Arduino IDE 2.3.2
- Toolbar:** File, Edit, Sketch, Tools, Help
- Sketch Selection:** Raspberry Pi Pico
- Code Area:** Displays the following C++ code for `Sketch_04.1_BreathingLight.ino`:

```
7 #define PIN_LED 15 //define the led pin
8
9 void setup() {
10     pinMode(PIN_LED, OUTPUT);
11 }
12
13 void loop() {
14     for (int i = 0; i < 255; i++) { //make light fade in
15         analogWrite(PIN_LED, i);
16         delay(5);
17     }
18     for (int i = 255; i > -1; i--) { //make light fade out
19         analogWrite(PIN_LED, i);
20         delay(5);
21     }
22 }
```

- Output Area:** A large black rectangular area labeled "Output".
- Status Bar:** Ln 3, Col 69 Raspberry Pi Pico on COM15

Download the code to Pico, and you will see that LED is turned from on to off and then from off to on gradually like breathing.



The following is the program code:

```

1 #define PIN_LED 15 //define the led pin
2
3 void setup() {
4     pinMode(PIN_LED, OUTPUT);
5 }
6
7 void loop() {
8     for (int i = 0; i < 255; i++) { //make light fade in
9         analogWrite(PIN_LED, i);
10        delay(5);
11    }
12    for (int i = 255; i > -1; i--) { //make light fade out
13        analogWrite(PIN_LED, i);
14        delay(5);
15    }
16 }
```

Set the pin controlling LED to output mode.

```
7 pinMode(PIN_LED, OUTPUT);
```

In the loop(), there are two “for” loops. The first makes the LED Pin output PWM from 0% to 100% and the second makes the LED Pin output PWM from 100% to 0%. This allows the LED to gradually light and extinguish.

```

11 for (int i = 0; i < 255; i++) { //make light fade in
12     analogWrite(PIN_LED, i);
13     delay(5);
14 }
15 for (int i = 255; i > -1; i--) { //make light fade out
16     analogWrite(PIN_LED, i);
17     delay(5);
18 }
```

You can also adjust the rate of the state change of LED by changing the parameters of the delay() function in the “for” loop.

**analogWrite(pin, value)**

Arduino IDE provides the function, analogWrite(pin, value), which can make ports directly output PWM waves. Every pin on Pico board can be configured to output PWM. In the function called analogWrite(pin, value), the parameter "pin" specifies the port used to output PWM wave. The range of value is 0-255, which represents the duty cycle of 0%-100%.

In order to use this function, we need to set the port to output mode.



## Project 4.2 Meteor Flowing Light

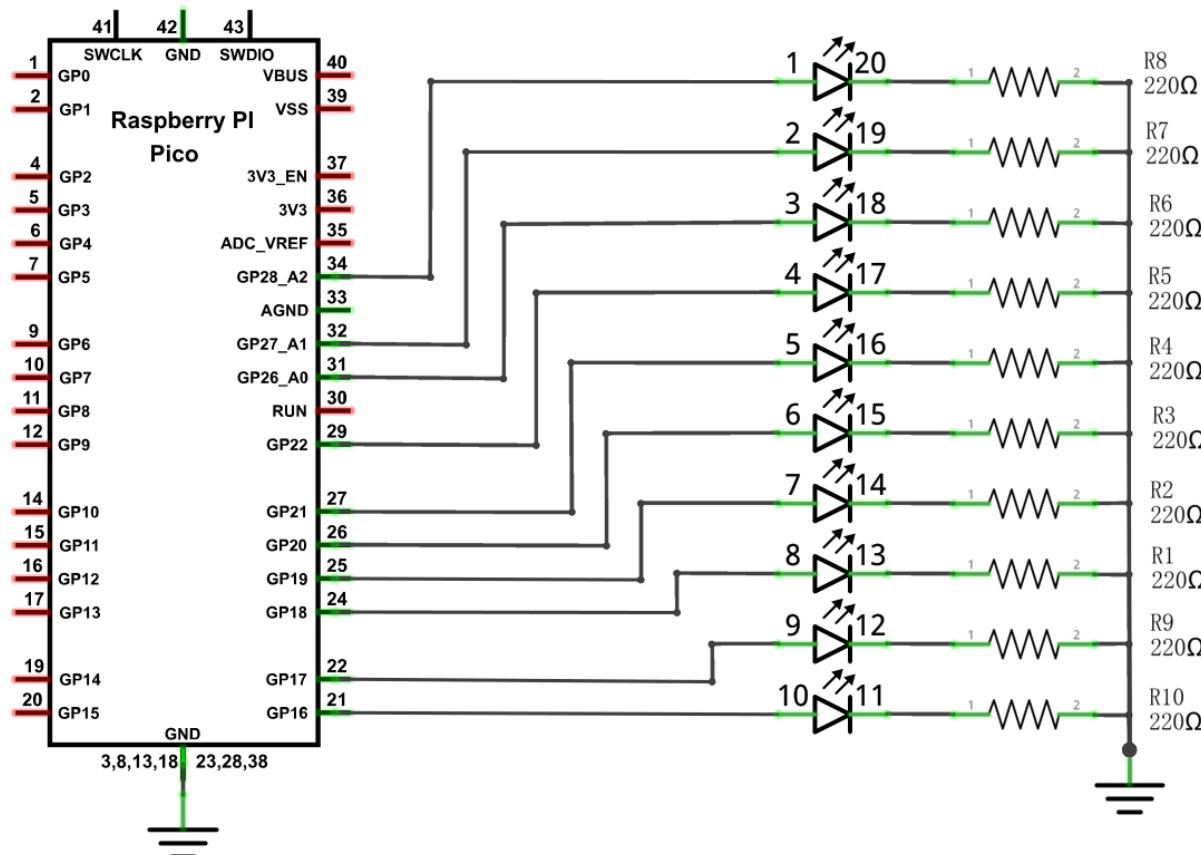
After learning about PWM, we can use it to control LED bar graph and realize a cooler flowing light. The component list, circuit, and hardware are exactly consistent with the project [Flowing Light](#).

### Component List

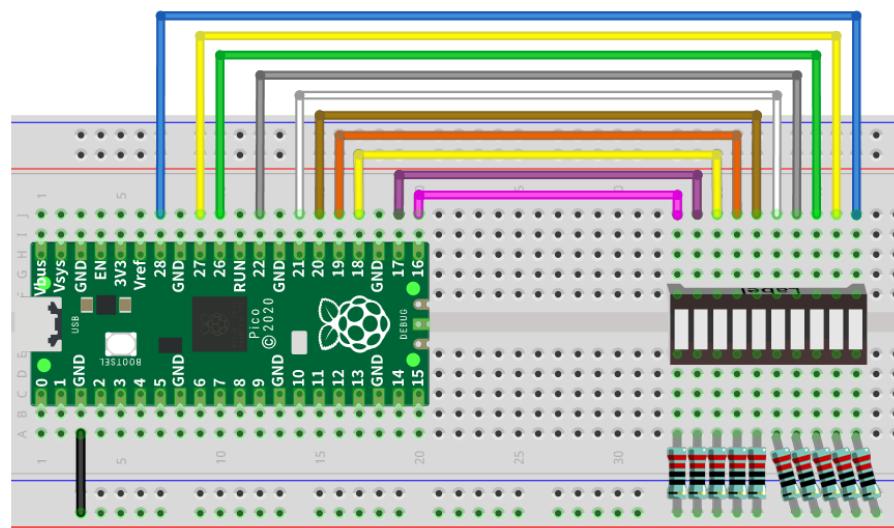
Raspberry Pi Pico x1	USB cable x1	
A green printed circuit board (PCB) with a central Broadcom SoC chip. It has four yellow circular pads labeled 'Raspberry Pi Pico • 2020'. Various components like resistors, capacitors, and connectors are soldered onto the board.	Two black USB cables, one with a standard A-type connector and the other with a smaller micro-B or similar connector.	
Breadboard x1		
Jumper	LED bar graph x1	Resistor 220Ω x10

## Circuit

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: [support@freenove.com](mailto:support@freenove.com)



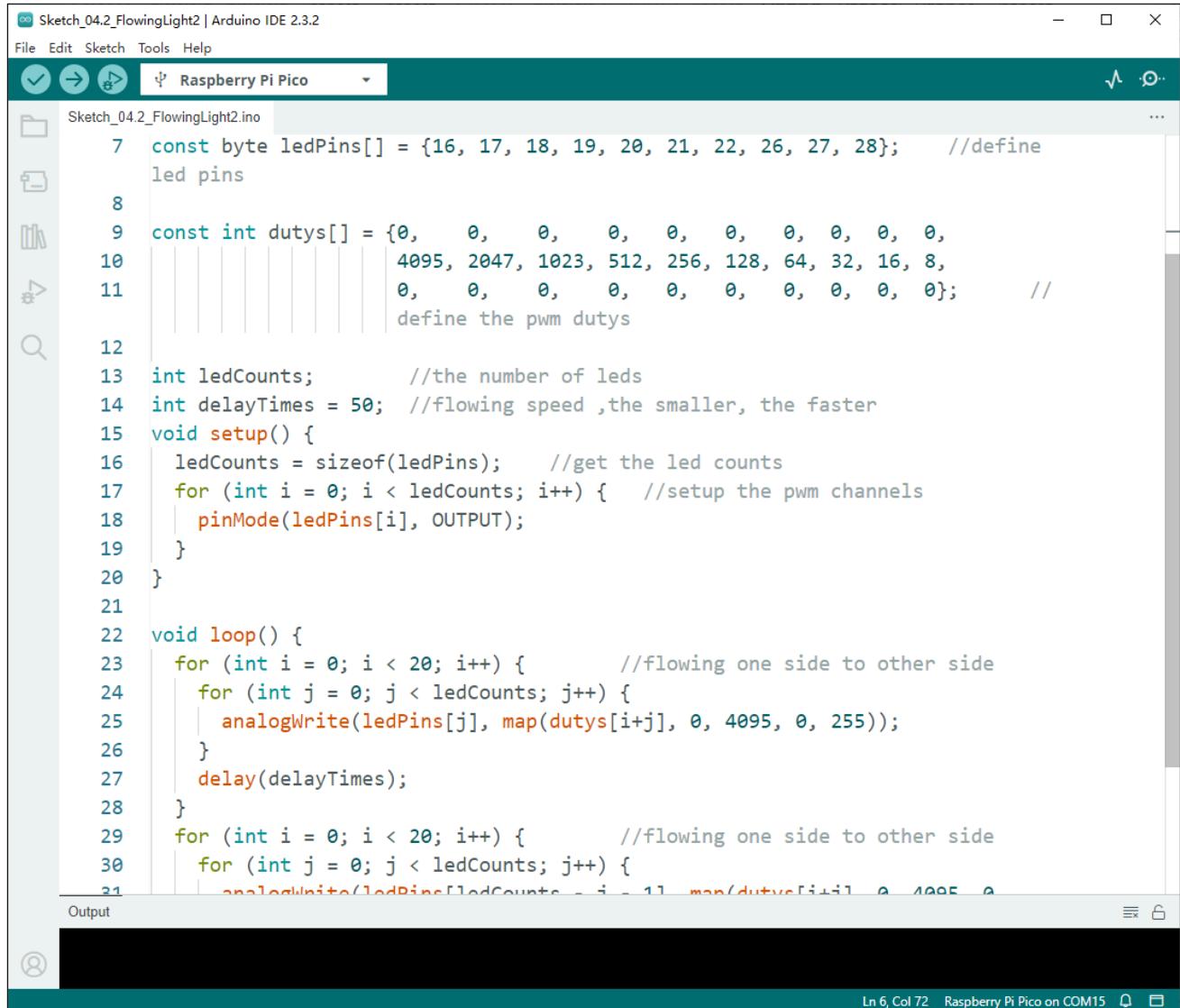
Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

If LEDbar does not work, try to rotate LEDbar for 180°. The label is random.

## Sketch

Meteor flowing light will be implemented with PWM.

### Sketch\_04.2\_FlowingLight2



```

Sketch_04.2_FlowingLight2 | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Raspberry Pi Pico
Sketch_04.2_FlowingLight2.ino
7 const byte ledPins[] = {16, 17, 18, 19, 20, 21, 22, 26, 27, 28}; //define
8 led pins
9 const int dutys[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
10 4095, 2047, 1023, 512, 256, 128, 64, 32, 16, 8,
11 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; //define the pwm dutys
12
13 int ledCounts; //the number of leds
14 int delayTimes = 50; //flowing speed ,the smaller, the faster
15 void setup() {
16     ledCounts = sizeof(ledPins); //get the led counts
17     for (int i = 0; i < ledCounts; i++) { //setup the pwm channels
18         pinMode(ledPins[i], OUTPUT);
19     }
20 }
21
22 void loop() {
23     for (int i = 0; i < 20; i++) { //flowing one side to other side
24         for (int j = 0; j < ledCounts; j++) {
25             analogWrite(ledPins[j], map(dutys[i+j], 0, 4095, 0, 255));
26         }
27         delay(delayTimes);
28     }
29     for (int i = 0; i < 20; i++) { //flowing one side to other side
30         for (int j = 0; j < ledCounts; j++) {
31             analogWrite(ledPins[ledCounts - i - 1], map(dutys[i+j], 0, 4095, 0,
32
Output
Ln 6, Col 72 Raspberry Pi Pico on COM15

```

Download the code to Pico, and LED bar graph will gradually light up and out from left to right, then back from right to left.

The following is the program code:

1	const byte ledPins[] = {16, 17, 18, 19, 20, 21, 22, 26, 27, 28}; //define led pins
2	
3	const int dutys[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
4	4095, 2047, 1023, 512, 256, 128, 64, 32, 16, 8,
5	0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; //define the pwm dutys
6	
7	int ledCounts; //the number of leds

Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

```

8 int delayTimes = 50; //flowing speed , the smaller, the faster
9 void setup() {
10 ledCounts = sizeof(ledPins); //get the led counts
11 for (int i = 0; i < ledCounts; i++) { //setup the pwm channels
12   pinMode(ledPins[i], OUTPUT);
13 }
14 }
15
16 void loop() {
17   for (int i = 0; i < 20; i++) { //flowing one side to other side
18     for (int j = 0; j < ledCounts; j++) {
19       analogWrite(ledPins[j], map(dutys[i+j], 0, 4095, 0, 255));
20     }
21     delay(delayTimes);
22   }
23   for (int i = 0; i < 20; i++) { //flowing one side to other side
24     for (int j = 0; j < ledCounts; j++) {
25       analogWrite(ledPins[ledCounts - j - 1], map(dutys[i+j], 0, 4095, 0, 255));
26     }
27     delay(delayTimes);
28   }
29 }
```

First, we defined 10 GPIO, 10 PWM channels, and 30 pulse width values.

```

1 const byte ledPins[] = {16, 17, 18, 19, 20, 21, 22, 26, 27, 28}; //define led pins
2
3 const int dutys[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
4                     4095, 2047, 1023, 512, 256, 128, 64, 32, 16, 8,
5                     0, 0, 0, 0, 0, 0, 0, 0, 0};//define the pwm dutys
```

Define a variable to store the number of LEDs and another to control the flashing speed of the LED bar.

```

7 int ledCounts; //the number of leds
8 int delayTimes = 50; //flowing speed , the smaller, the faster
```

`Sizeof()` function is used to obtain the number of members of the array `ledPins` and assign it to `ledCount`. Use the 'for' loop to set all pins to output mode.

```

10 ledCounts = sizeof(ledPins); //get the led counts
11 for (int i = 0; i < ledCounts; i++) { //setup the pwm channels
12   pinMode(ledPins[i], OUTPUT);
13 }
```



In loop(), a nested for loop is used to control the pulse width of the PWM, and LED bar graph moves one grid after each 1 is added in the first for loop, gradually changing according to the values in the array duties. As shown in the table below, the value of the second row is the value in the array duties, and the 10 green squares in each row below represent the 10 LEDs on the LED bar graph. Every 1 is added to  $I$ , the value of the LED bar graph will move to the right by one grid, and when it reaches the end, it will move from the end to the starting point, achieving the desired effect.

0	1	2	3	4	5	7	8	9	1	11	1	1	1	1	1	1	1	2	2	2	2	2	2	3
d	0	0	0	0	0	0	0	0	0	10	5	2	1	6	3	1	8	4	2	0	0	0	0	0
i										23	1	5	2	4	2	6								
0																								
1																								
2																								
3																								
...																								
1																								
8																								
1																								
9																								
2																								
0																								

In the code, two nested for loops are used to achieve this effect.

```

17   for (int i = 0; i < 20; i++) {          //flowing one side to other side
18     for (int j = 0; j < ledCounts; j++) {
19       analogWrite(ledPins[j], map(dutys[i+j], 0, 4095, 0, 255));
20     }
21     delay(delayTimes);
22   }
23   for (int i = 0; i < 20; i++) {          //flowing one side to other side
24     for (int j = 0; j < ledCounts; j++) {
25       analogWrite(ledPins[ledCounts - j - 1], map(dutys[i+j], 0, 4095, 0, 255));
26     }
27     delay(delayTimes);
28   }

```

map(value, fromLow, fromHigh, toLow, toHigh)

This function is used to remap a value, which will return a new value whose percentage in the range of toLow-toHigh is equal to the percentage of "value" in the range of fromLow-fromHigh. For example, 1 is the maximum in the range of 0-1 and the maximum value in the scope of 0-2 is 2, that is, the result value of map (1, 0, 1, 0, 2) is 2.

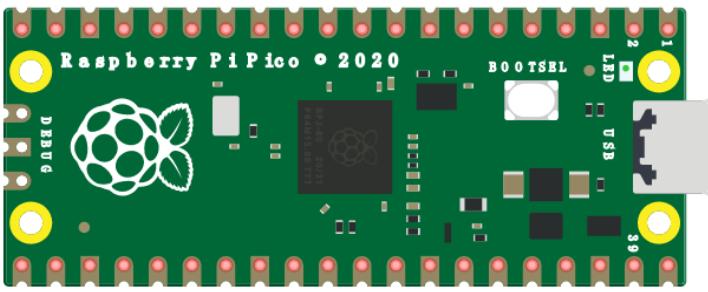
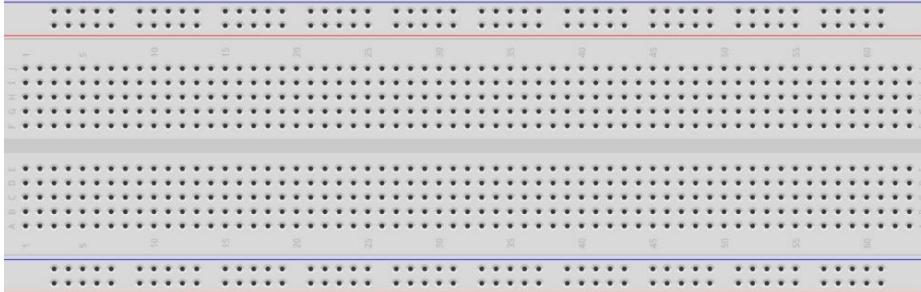
# Chapter 5 RGBLED

In this chapter, we will learn how to control an RGBLED. It can emit different colors of light. Next, we will use RGBLED to make a multicolored light.

## Project 5.1 Random Color Light

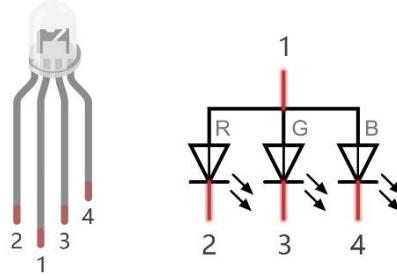
In this project, we will make a multicolored LED. And we can control RGBLED to switch different colors automatically.

### Component List

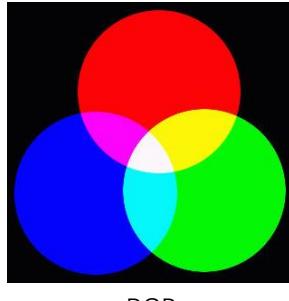
Raspberry Pi Pico x1	USB cable x1	
		
Breadboard x1		
		
RGBLED x1	Resistor 220Ω x3	Jumper
		

## Related Knowledge

RGB LED has integrated three LEDs that can respectively emit red, green and blue light. It has four pins. The long pin (1) is the common port, that is, 3 LED's positive or negative port. The RGB LED with common positive port and its symbol is shown below. We can make RGB LED emit various colors of light by controlling these three LEDs to emit light with different brightness.



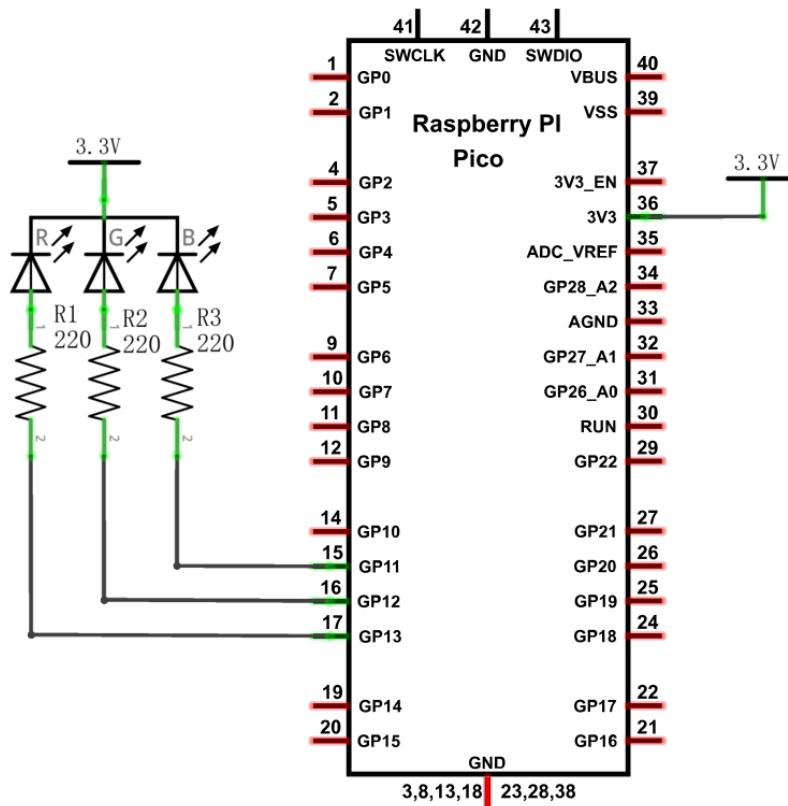
Red, green, and blue light are known as three primary colors. When you combine these three primary-color lights with different brightness, it can produce almost all kinds of visible lights. Computer screens, single pixel of cell phone screen, neon, etc. are working under this principle.



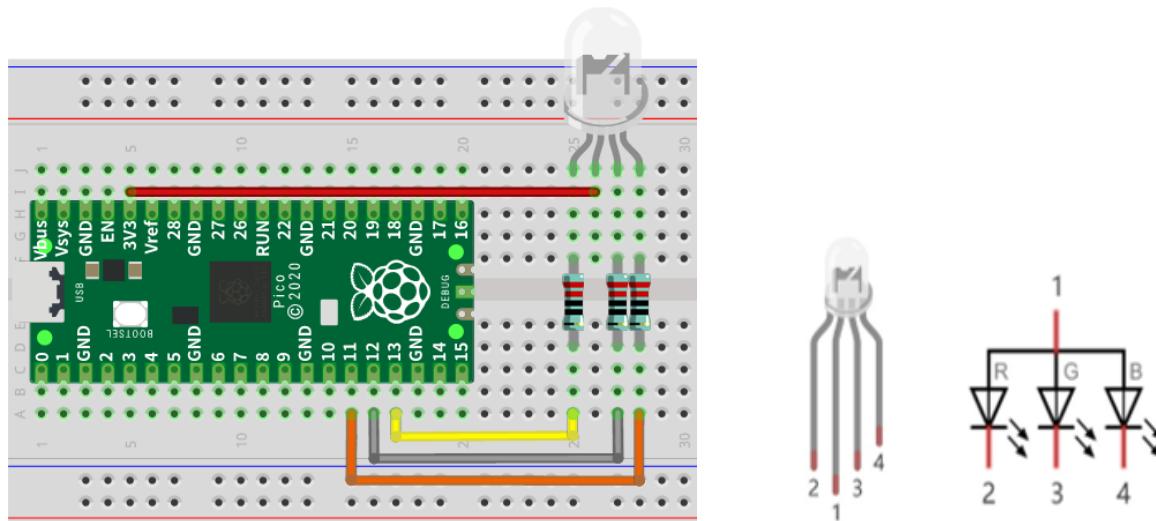
If we use three 8-bit PWMs to control the RGB LED, in theory, we can create  $2^8 \times 2^8 \times 2^8 = 16777216$  (16 million) colors through different combinations.

## Circuit

Schematic diagram



Hardware connection. If you need any support, please free to contact us via: [support@freenove.com](mailto:support@freenove.com)



Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

## Sketch

We need to create three PWM channels and use random duty cycle to make random RGB LED color.

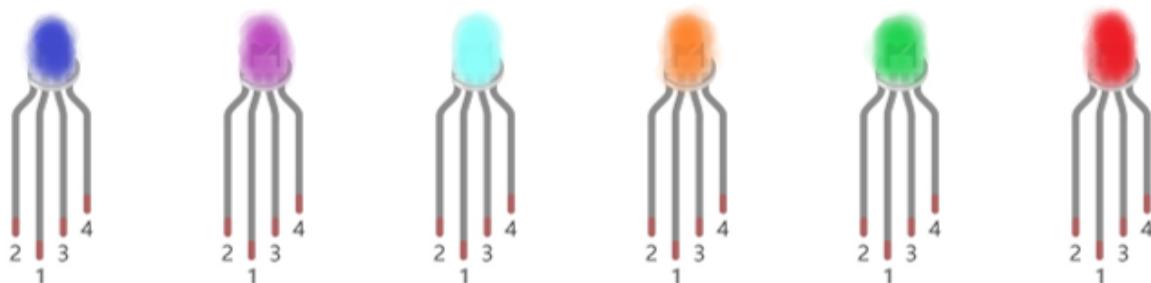
### Sketch\_05.1\_ColorfulLight



```

Sketch_05.1_RandomColorLight | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Sketch_05.1_RandomColorLight.ino
1 int ledPins[] = {13, 12, 11};      //define red, green, blue led pins
2 int red, green, blue;
3 void setup() {
4     for (int i = 0; i < 3; i++) {    //setup the pwm channels,1KHz,8bit
5         pinMode(ledPins[i], OUTPUT);
6     }
7 }
8
9 void loop() {
10    red = random(0, 255);
11    green = random(0, 255);
12    blue = random(0, 255);
13    setColor(red, green, blue);
14    delay(200);
15 }
16
17 void setColor(byte r, byte g, byte b) {
18     analogWrite(ledPins[0], 255-r); //Common anode LED, low level to turn on the
19     //led.
20     analogWrite(ledPins[1], 255-g);
21     analogWrite(ledPins[2], 255-b);
22 }
23
24
25
26
27 }
```

With the code downloaded to Pico, RGB LED begins to display random colors.



The following is the program code:

1	int ledPins[] = {13, 12, 11};      //define red, green, blue led pins
2	int red, green, blue;
3	void setup() {
4	for (int i = 0; i < 3; i++) {    //setup the pwm channels,1KHz,8bit
5	pinMode(ledPins[i], OUTPUT);
6	}
7	}
8	

```

9 void loop() {
10    red = random(0, 255);
11    green = random(0, 255);
12    blue = random(0, 255);
13    setColor(red, green, blue);
14    delay(200);
15 }
16
17 void setColor(byte r, byte g, byte b) {
18    analogWrite(ledPins[0], 255-r); //Common anode LED, low level to turn on the led.
19    analogWrite(ledPins[1], 255-g);
20    analogWrite(ledPins[2], 255-b);
21 }
```

Define pins to control RGB LED, and configure them as output mode.

```

1 int ledPins[] = {13, 12, 11}; //define red, green, blue led pins
2 int red, green, blue;
3 void setup() {
4     for (int i = 0; i < 3; i++) { //setup the pwm channels, 1KHz, 8bit
5         pinMode(ledPins[i], OUTPUT);
6     }
7 }
```

In setColor(), this function controls the output color of RGB LED by the given color value. Because the circuit uses a common anode, the LED lights up when the GPIO outputs low power. Therefore, in PWM, low level is the active level, so 255 minus the given value is necessary.

```

19 void setColor(byte r, byte g, byte b) {
20     ledcWrite(chns[0], 255 - r); //Common anode LED, low level to turn on the led.
21     ledcWrite(chns[1], 255 - g);
22     ledcWrite(chns[2], 255 - b);
23 }
```

In loop(), get three random Numbers and set them as color values.

```

12 red = random(0, 255);
13 green = random(0, 255);
14 blue = random(0, 255);
15 setColor(red, green, blue);
16 delay(200);
```

The related function of software PWM can be described as follows:

**long random(min, max);**

This function will return a random number(min --- max-1).



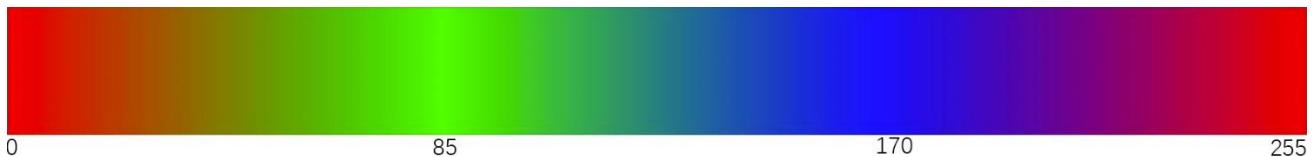
## Project 5.2 Gradient Color Light

In the previous project, we have mastered the usage of RGBLED, but the random color display is rather stiff.

This project will realize a fashionable Light with soft color changes.

Component list, the circuit is the same as the project random color light.

Using a color model, the color changes from 0 to 255 as shown below.



## Sketch

In this code, the color model will be implemented and RGBLED will change colors along the model.

### [Sketch\\_05.2\\_SoftColorfulLight](#)

The following is the program code:

```

1 const byte ledPins[] = {13, 12, 11};      //define led pins
2 void setup() {
3     for (int i = 0; i < 3; i++) {    //setup the pwm channels
4         pinMode(ledPins[i], OUTPUT);
5     }
6 }
7
8 void loop() {
9     for (int i = 0; i < 256; i++) {
10        setColor(wheel(i));
11        delay(100);
12    }
13 }
14
15 void setColor(long rgb) {
16     analogWrite(ledPins[0], 255 - (rgb >> 16) & 0xFF);
17     analogWrite(ledPins[1], 255 - (rgb >> 8) & 0xFF);
18     analogWrite(ledPins[2], 255 - (rgb >> 0) & 0xFF);
19 }
20
21 long wheel(int pos) {
22     long WheelPos = pos % 0xff;
23     if (WheelPos < 85) {
24         return ((255 - WheelPos * 3) << 16) | ((WheelPos * 3) << 8);
25     } else if (WheelPos < 170) {

```

Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

```
26     WheelPos -= 85;
27     return (((255 - WheelPos * 3) << 8) | (WheelPos * 3));
28 } else {
29     WheelPos -= 170;
30     return ((WheelPos * 3) << 16 | (255 - WheelPos * 3));
31 }
32 }
```

In `setColor()`, a variable represents the value of RGB, and a hexadecimal representation of color is a common representation, such as `0xAABBCC`, where AA represents the red value, BB represents the green value, and CC represents the blue value. The use of a variable can make the transmission of parameters more convenient, in the split, only a simple operation can take out the value of each color channel

```
15 void setColor(long rgb) {
16     ledcWrite(chns[0], 255 - (rgb >> 16) & 0xFF);
17     ledcWrite(chns[1], 255 - (rgb >> 8) & 0xFF);
18     ledcWrite(chns[2], 255 - (rgb >> 0) & 0xFF);
19 }
```

The `wheel()` function is the color selection method for the color model introduced earlier. The **pos** parameter ranges from 0 to 255 and outputs a color value in hexadecimal.

```
21 long wheel(int pos) {
22     long WheelPos = pos % 0xff;
23     if (WheelPos < 85) {
24         return (((255 - WheelPos * 3) << 16) | ((WheelPos * 3) << 8));
25     } else if (WheelPos < 170) {
26         WheelPos -= 85;
27         return (((255 - WheelPos * 3) << 8) | (WheelPos * 3));
28     } else {
29         WheelPos -= 170;
30         return ((WheelPos * 3) << 16 | (255 - WheelPos * 3));
31     }
32 }
```

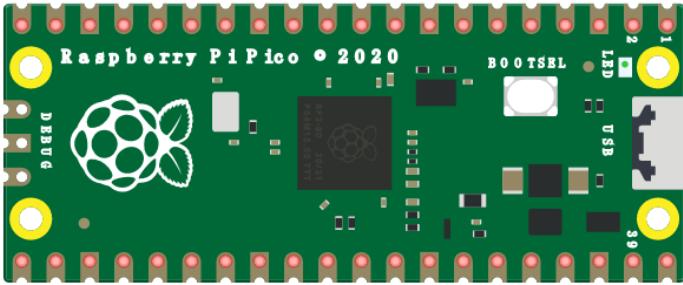
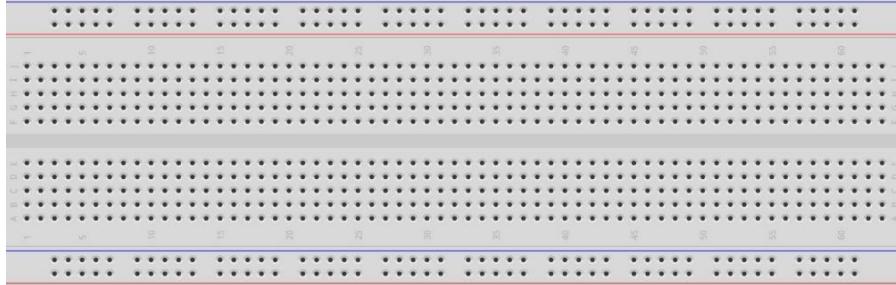
# Chapter 6 Buzzer

In this chapter, we will learn about buzzers and the sounds they make.

## Project 6.1 Doorbell

We will make this kind of doorbell: when the button is pressed, the buzzer sounds; and when the button is released, the buzzer stops sounding.

### Component List

Raspberry Pi Pico x1		USB cable x1	
Breadboard x1			
Jumper			
NPN transistor x1 (S8050)		Active buzzer x1	
Push button x1		Resistor 1kΩ x1	
		Resistor 10kΩ x2	

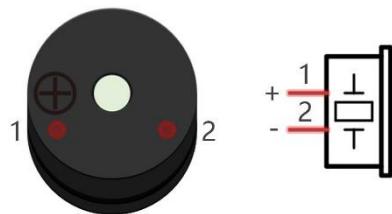
Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

## Component Knowledge

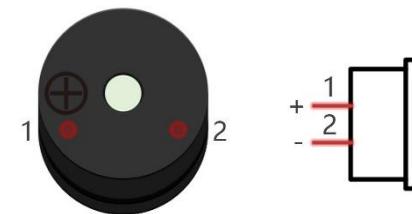
### Buzzer

Buzzer is a sounding component, which is widely used in electronic devices such as calculator, electronic warning clock and alarm. Buzzer has two types: active and passive. Active buzzer has oscillator inside, which will sound as long as it is supplied with power. Passive buzzer requires external oscillator signal (generally use PWM with different frequency) to make a sound.

Active buzzer



Passive buzzer



Active buzzer is easy to use. Generally, it can only make a specific frequency of sound. Passive buzzer requires an external circuit to make a sound, but it can be controlled to make a sound with different frequency. The resonant frequency of the passive buzzer is 2 kHz, which means the passive buzzer is loudest when its resonant frequency is 2 kHz.

Next, we will use an active buzzer to make a doorbell and a passive buzzer to make an alarm.

### How to identify active and passive buzzer?

1. Usually, there is a label on the surface of active buzzer covering the vocal hole, but this is not an absolute judgment method.
2. Active buzzers are more complex than passive buzzers in their manufacture. There are many circuits and crystal oscillator elements inside active buzzers; all of this is usually protected with a waterproof coating (and a housing) exposing only its pins from the underside. On the other hand, passive buzzers do not have protective coatings on their underside. From the pin holes viewing of a passive buzzer, you can see the circuit board, coils, and a permanent magnet (all or any combination of these components depending on the model).

Active buzzer



Passive buzzer



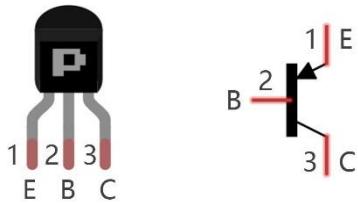
### Transistor

Because the buzzer requires such large current that GP of Raspberry Pi Pico output capability cannot meet the requirement, a transistor of NPN type is needed here to amplify the current.

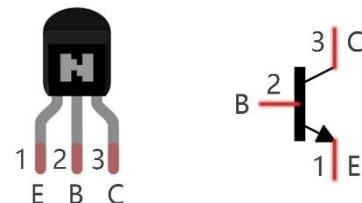
Transistor, the full name: semiconductor transistor, is a semiconductor device that controls current. Transistor

can be used to amplify weak signal, or works as a switch. It has three electrodes(PINs): base (b), collector (c) and emitter (e). When there is current passing between "be", "ce" will allow several-fold current (transistor magnification) pass, at this point, transistor works in the amplifying area. When current between "be" exceeds a certain value, "ce" will not allow current to increase any longer, at this point, transistor works in the saturation area. Transistor has two types as shown below: PNP and NPN.

PNP transistor



NPN transistor

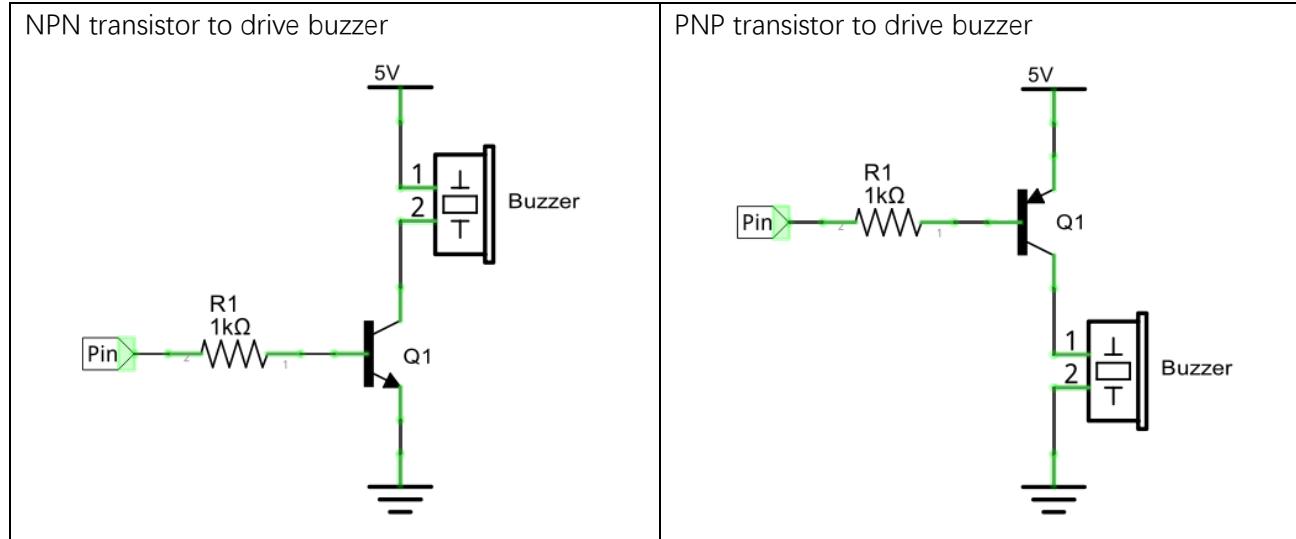


In our kit, the PNP transistor is marked with 8550, and the NPN transistor is marked with 8050.

Based on the transistor's characteristics, it is often used as a switch in digital circuits. As micro-controller's capacity to output current is very weak, we will use transistor to amplify current and drive large-current components.

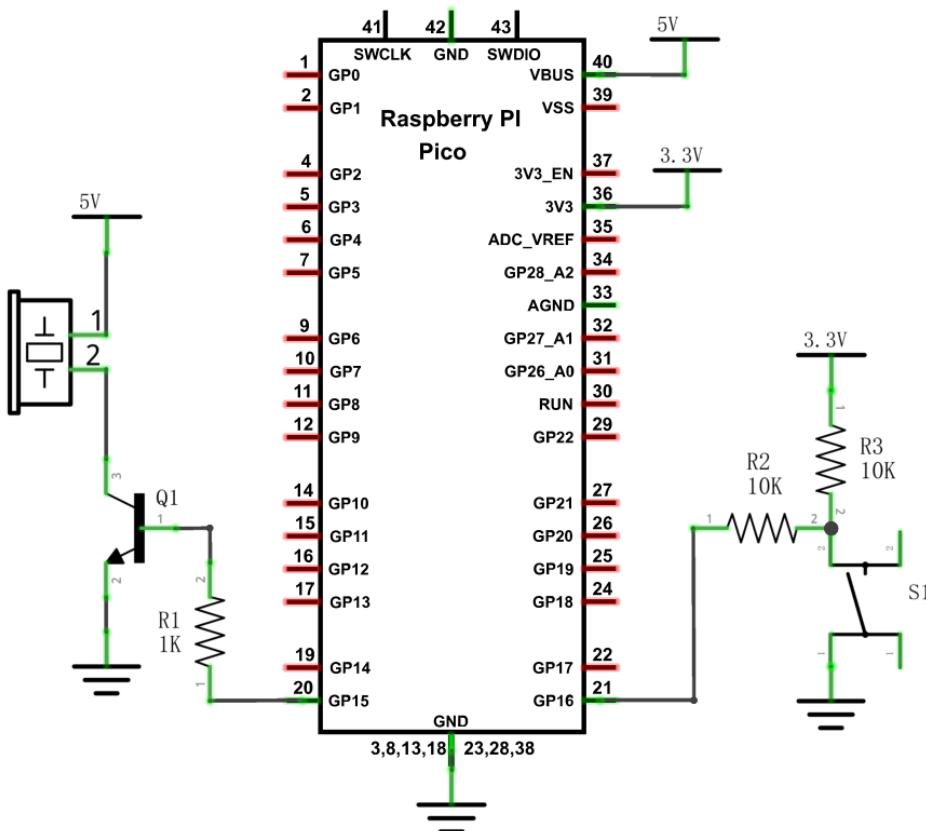
When using NPN transistor to drive buzzer, we often adopt the following method. If GP outputs high level, current will flow through R1, the transistor will be conducted, and the buzzer will sound. If GP outputs low level, no current flows through R1, the transistor will not be conducted, and buzzer will not sound.

When using PNP transistor to drive buzzer, we often adopt the following method. If GP outputs low level, current will flow through R1, the transistor will be conducted, and the buzzer will sound. If GP outputs high level, no current flows through R1, the transistor will not be conducted, and buzzer will not sound.

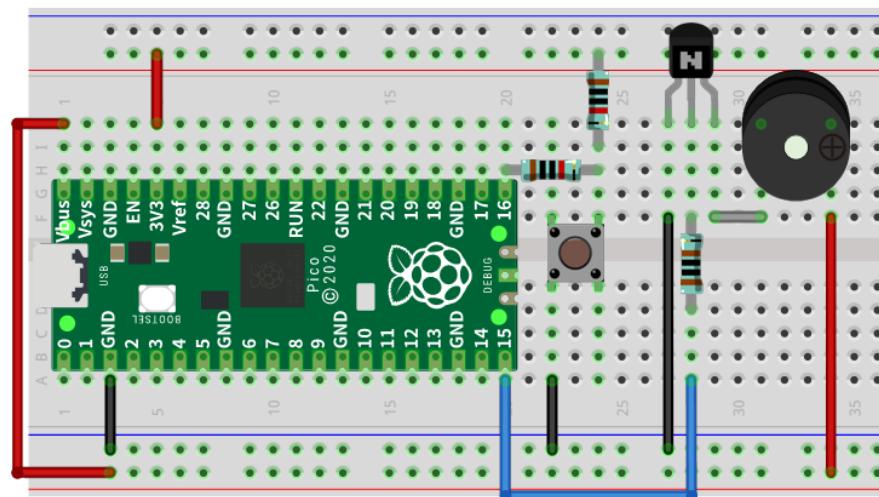


## Circuit

Schematic diagram



Hardware connection. If you need any support, please free to contact us via: [support@freenove.com](mailto:support@freenove.com)



Note:

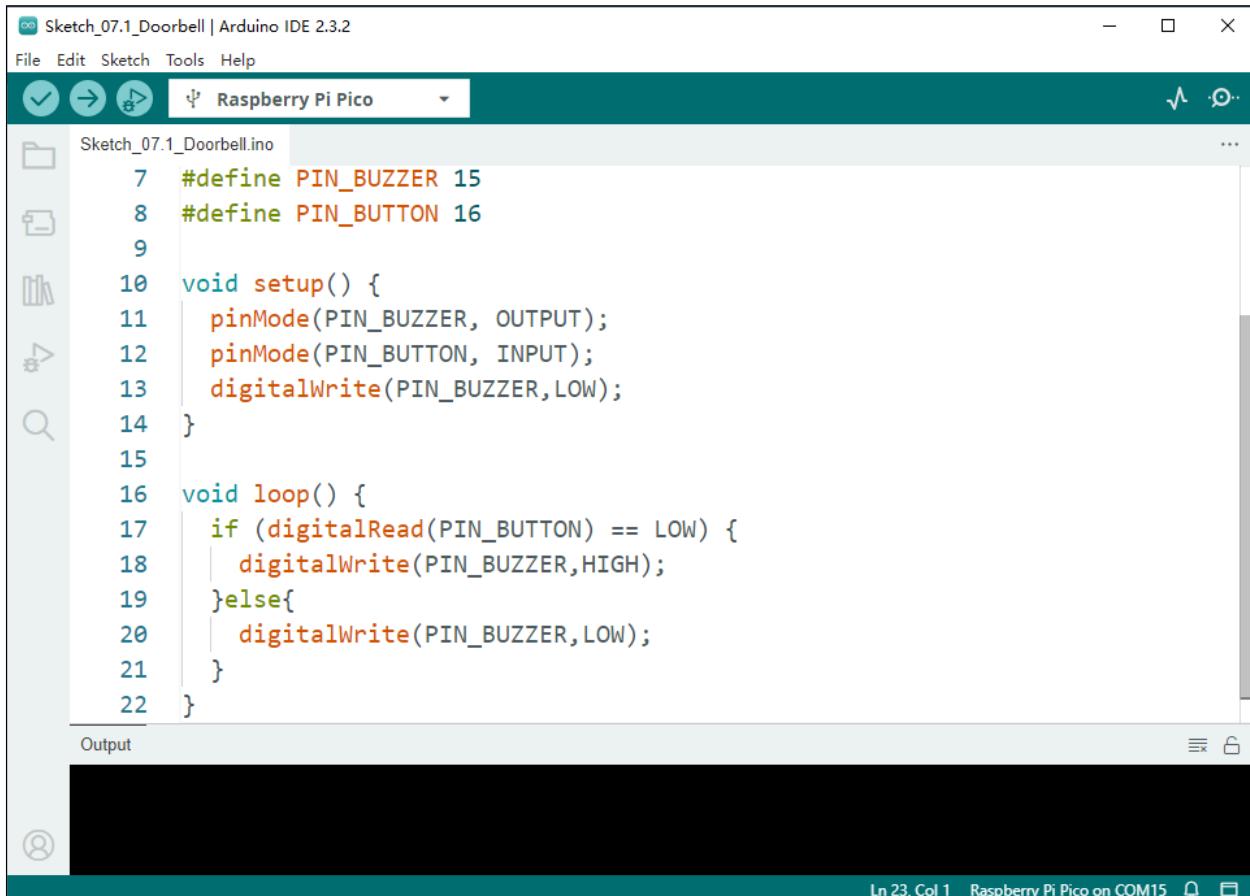
1. In this circuit, the power supply for buzzer is 5V, and pull-up resistor of the button connected to the power 3.3V. The buzzer can work when connected to power 3.3V, but it will reduce the loudness.
2. VBUS should be connect to the positive end of USB cable. If it connects to GND, it may burn the computer or Raspberry Pi Pico. Similarly, please be careful when wiring pins 36-40 of Pico to avoid short circuit.

Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

## Sketch

In this project, a buzzer will be controlled by a push button switch. When the button switch is pressed, the buzzer sounds and when the button is released, the buzzer stops. It is analogous to our earlier project that controlled an LED ON and OFF.

### Sketch\_06.1\_Doorbell



```

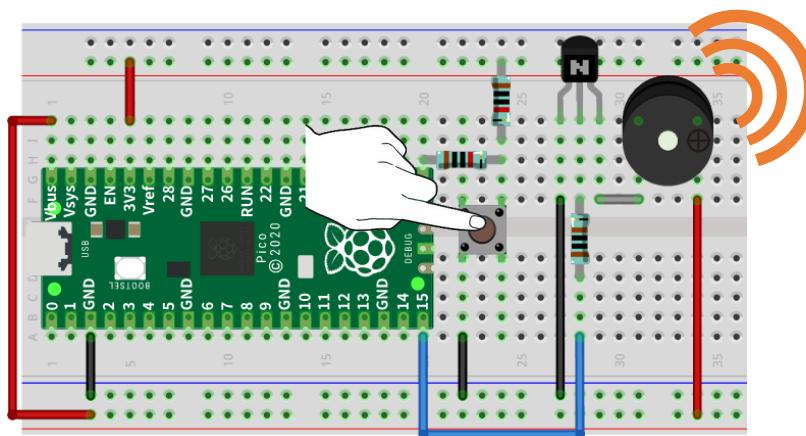
Sketch_06.1_Doorbell | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Raspberry Pi Pico
Sketch_06.1_Doorbell.ino
7 #define PIN_BUZZER 15
8 #define PIN_BUTTON 16
9
10 void setup() {
11     pinMode(PIN_BUZZER, OUTPUT);
12     pinMode(PIN_BUTTON, INPUT);
13     digitalWrite(PIN_BUZZER,LOW);
14 }
15
16 void loop() {
17     if (digitalRead(PIN_BUTTON) == LOW) {
18         digitalWrite(PIN_BUZZER,HIGH);
19     }else{
20         digitalWrite(PIN_BUZZER,LOW);
21     }
22 }

```

Output

Ln 23, Col 1 Raspberry Pi Pico on COM15

Download the code to Pico, press the push button switch and the buzzer will sound. Release the push button switch and the buzzer will stop.



The following is the program code:

```
1 #define PIN_BUZZER 15
2 #define PIN_BUTTON 16
3
4 void setup() {
5     pinMode(PIN_BUZZER, OUTPUT);
6     pinMode(PIN_BUTTON, INPUT);
7     digitalWrite(PIN_BUZZER, LOW);
8 }
9
10 void loop() {
11     if (digitalRead(PIN_BUTTON) == LOW) {
12         digitalWrite(PIN_BUZZER, HIGH);
13     } else{
14         digitalWrite(PIN_BUZZER, LOW);
15     }
16 }
```

The code is logically the same as using button to control LED.

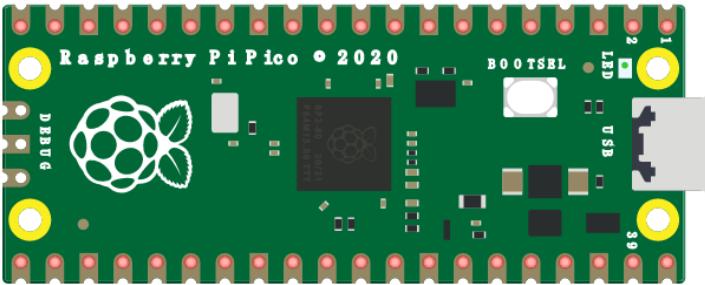
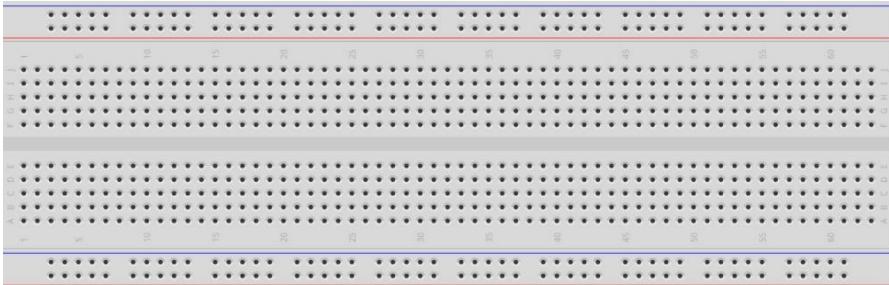
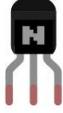


## Project 6.2 Alertor

Next, we will use a passive buzzer to make an alarm.

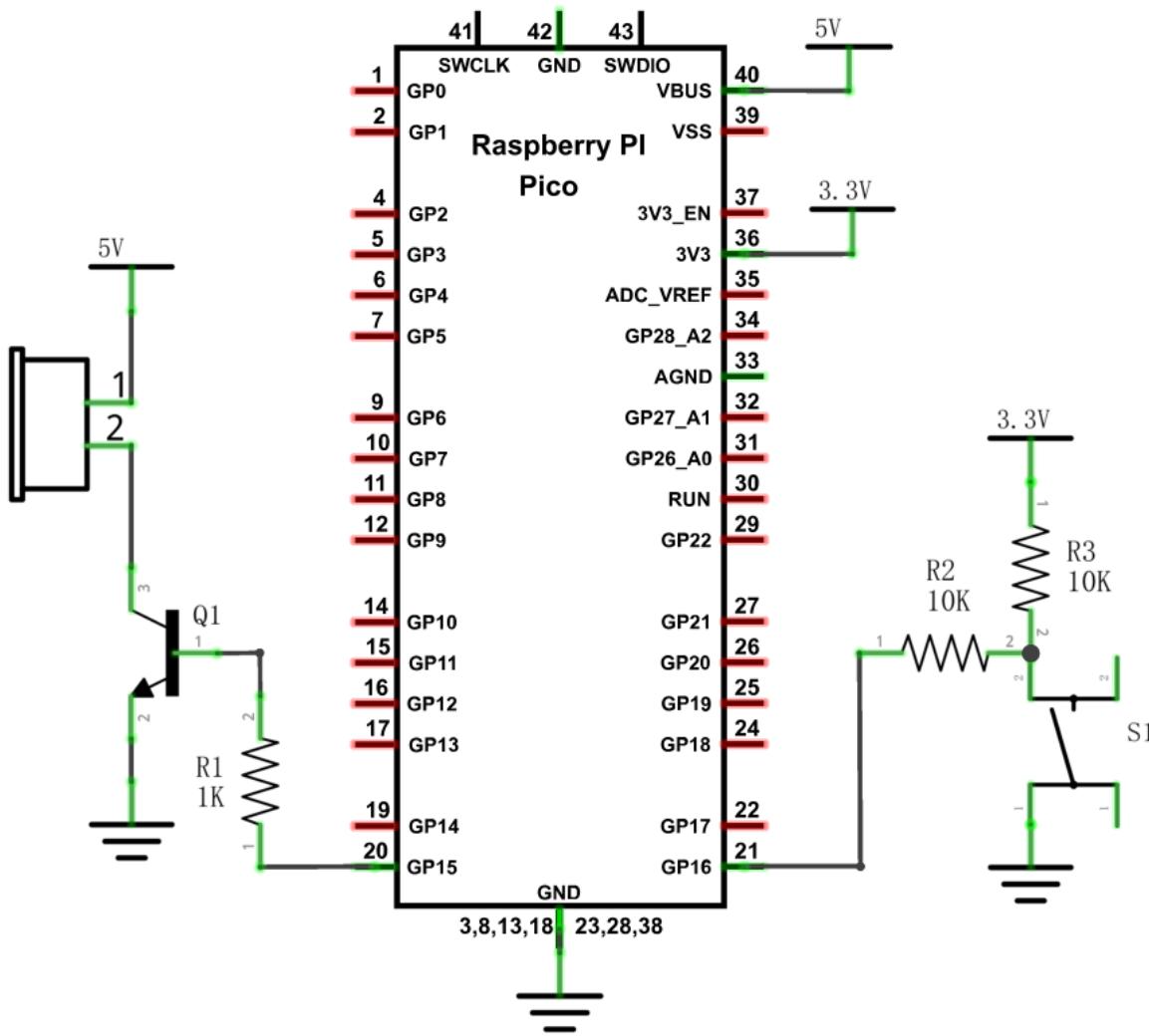
Component list and the circuit part is similar to last section, only the **active buzzer** needs to be **replaced** with a **passive buzzer** for this project.

### Component List

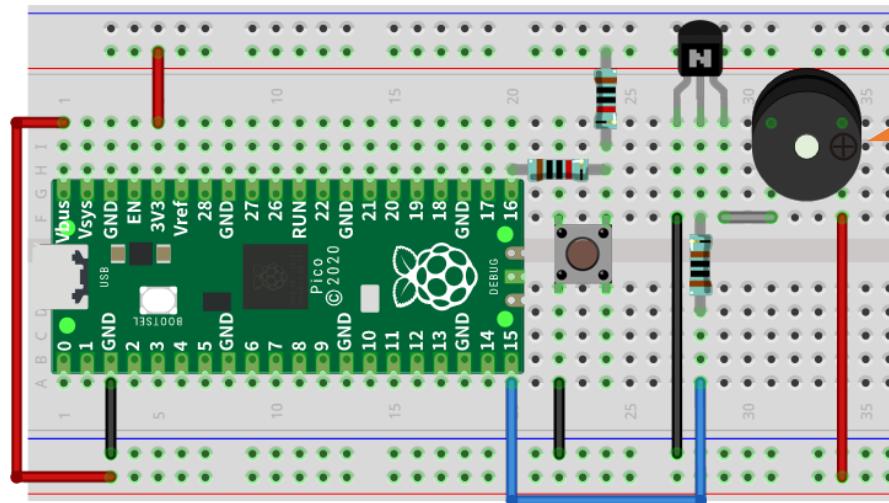
Raspberry Pi Pico x1		USB cable x1	
Breadboard x1			
Jumper			
NPN transistorx1 (S8050)		Passive buzzer x1	
Push button x1		Resistor 1kΩ x1	
Resistor 10kΩ x2			

## Circuit

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: [support@freenove.com](mailto:support@freenove.com)

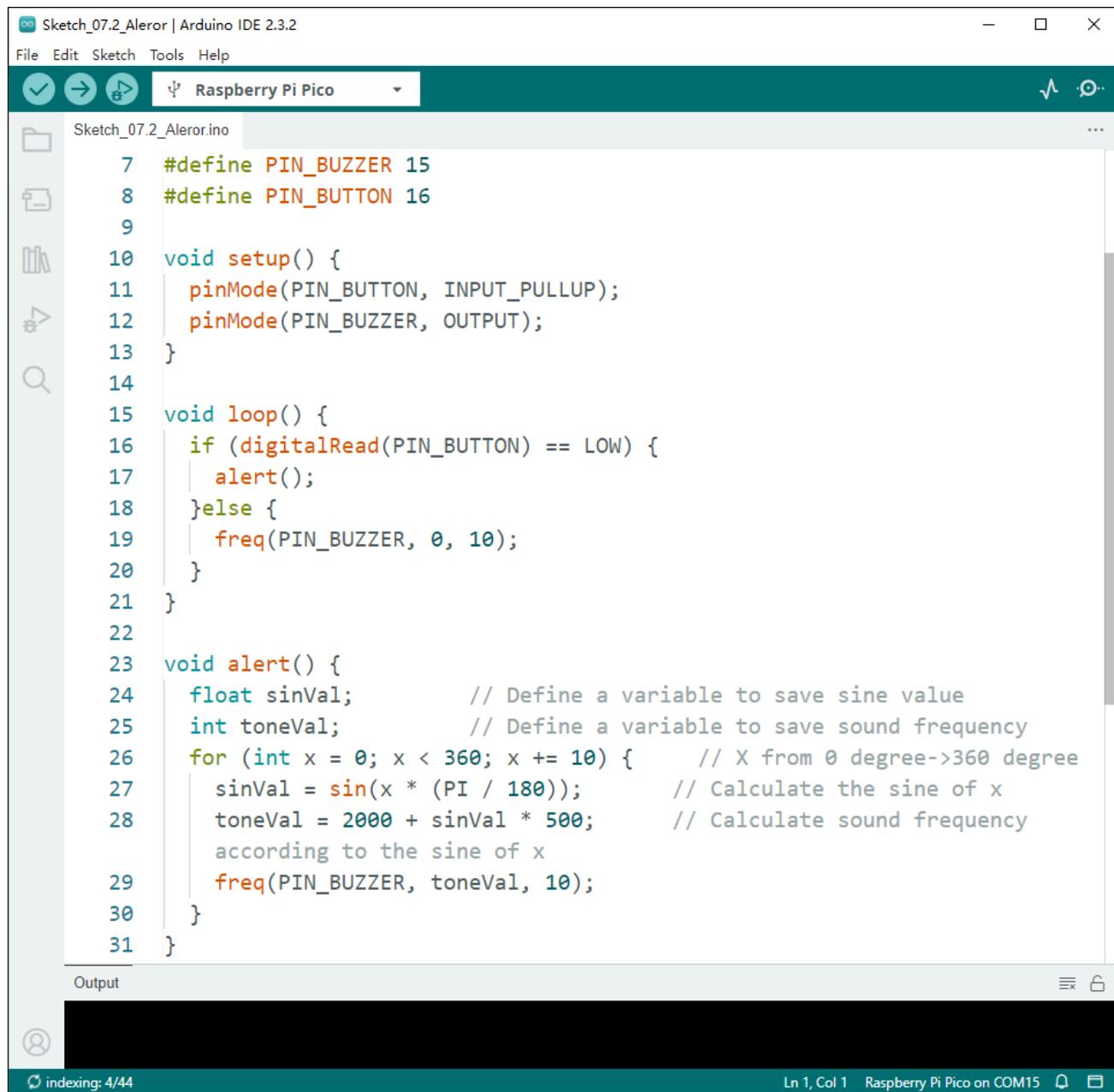


Any concerns? ✉ [support@freenove.com](mailto:support@freenove.com)

## Sketch

In this project, the buzzer alarm is controlled by the button. Press the button, then buzzer sounds. If you release the button, the buzzer will stop sounding. It is logically the same as using button to control LED, but in the control method, passive buzzer requires PWM of certain frequency to sound.

### Sketch\_06.2\_Alertor

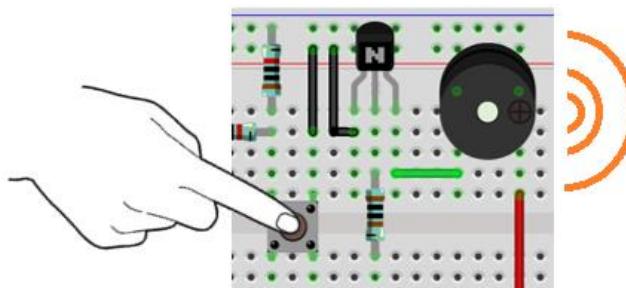


The screenshot shows the Arduino IDE interface with the following details:

- Title Bar:** Sketch\_07.2\_Aleror | Arduino IDE 2.3.2
- Menu Bar:** File Edit Sketch Tools Help
- Toolbar:** Includes icons for Save, Run, Stop, and a dropdown menu set to "Raspberry Pi Pico".
- Code Editor:** Displays the `Sketch_07.2_Aleror.ino` file content. The code uses `#define` statements to map pins to constants. It defines `PIN_BUZZER` as 15 and `PIN_BUTTON` as 16. The `setup()` function initializes the button pin as INPUT\_PULLUP and the buzzer pin as OUTPUT. The `loop()` function checks if the button is pressed. If it is, it calls the `alert()` function. Otherwise, it plays a sine wave at a frequency calculated based on the sine of the current angle (x) from 0 to 360 degrees. The `freq()` function is used to set the frequency of the PWM signal.

```
Sketch_07.2_Aleror.ino
7 #define PIN_BUZZER 15
8 #define PIN_BUTTON 16
9
10 void setup() {
11     pinMode(PIN_BUTTON, INPUT_PULLUP);
12     pinMode(PIN_BUZZER, OUTPUT);
13 }
14
15 void loop() {
16     if (digitalRead(PIN_BUTTON) == LOW) {
17         alert();
18     }else {
19         freq(PIN_BUZZER, 0, 10);
20     }
21 }
22
23 void alert() {
24     float sinVal;          // Define a variable to save sine value
25     int toneVal;           // Define a variable to save sound frequency
26     for (int x = 0; x < 360; x += 10) {    // X from 0 degree->360 degree
27         sinVal = sin(x * (PI / 180));      // Calculate the sine of x
28         toneVal = 2000 + sinVal * 500;       // Calculate sound frequency
29         freq(PIN_BUZZER, toneVal, 10);
30     }
31 }
```
- Output Panel:** Shows the message "indexing: 4/44" and "Ln 1, Col 1 Raspberry Pi Pico on COM15".

Download the code to Pico, press the button, and then alarm sounds; when the button is released, the alarm will stop sounding.



The following is the program code:

```
1 #define PIN_BUZZER 15
2 #define PIN_BUTTON 16
3
4 void setup() {
5     pinMode(PIN_BUTTON, INPUT_PULLUP);
6     pinMode(PIN_BUZZER, OUTPUT);
7 }
8
9 void loop() {
10    if (digitalRead(PIN_BUTTON) == LOW) {
11        alert();
12    }else {
13        freq(PIN_BUZZER, 0, 10);
14    }
15 }
16
17 void alert() {
18     float sinVal;          // Define a variable to save sine value
19     int toneVal;           // Define a variable to save sound frequency
20     for (int x = 0; x < 360; x += 10) { // X from 0 degree->360 degree
21         sinVal = sin(x * (PI / 180));      // Calculate the sine of x
22         toneVal = 2000 + sinVal * 500;      // Calculate sound frequency according to the sine of x
23         freq(PIN_BUZZER, toneVal, 10);
24     }
25 }
26
27 void freq(int PIN, int freqs, int times) {
28     if (freqs == 0) {
29         digitalWrite(PIN, LOW);
30     }
31     else {
32         for (int i = 0; i < times * freqs / 1000; i++) {
33             digitalWrite(PIN, HIGH);
```

```

34     delayMicroseconds(1000000 / freqs / 2);
35     digitalWrite(PIN, LOW);
36     delayMicroseconds(1000000 / freqs / 2);
37   }
38 }
39 }
```

Define the button and pin to control the passive buzzer.

```

1 #define PIN_BUZZER 15
2 #define PIN_BUTTON 16
```

Write a function to drive the passive buzzer with a duty cycle of 50%. The `delayMicroseconds()` function is in

1us.  $1 \text{ s} = 1000000 \text{ us}$ . By the formula  $T = \frac{1}{f}$ , when the frequency is fixed, the PWM period T is also fixed.

```

27 void freq(int PIN, int freqs, int times) {
28   if (freqs == 0) {
29     digitalWrite(PIN, LOW);
30   }
31   else {
32     for (int i = 0; i < times * freqs / 1000; i++) {
33       digitalWrite(PIN, HIGH);
34       delayMicroseconds(1000000 / freqs / 2);
35       digitalWrite(PIN, LOW);
36       delayMicroseconds(1000000 / freqs / 2);
37     }
38   }
39 }
```

The frequency curve of the alarm is based on the sine curve. We need to calculate the sine value from 0 to 360 degree and multiply a certain value (here it is 500) and plus the resonant frequency of buzzer.

```

17 void alert() {
18   float sinVal;           // Define a variable to save sine value
19   int toneVal;            // Define a variable to save sound frequency
20   for (int x = 0; x < 360; x += 10) { // X from 0 degree->360 degree
21     sinVal = sin(x * (PI / 180));      // Calculate the sine of x
22     toneVal = 2000 + sinVal * 500;      // Calculate sound frequency according to the sine of x
23     freq(PIN_BUZZER, toneVal, 10);
24   }
25 }
```

In the `loop()` function, when the button is pressed, subfunction `alert()` will be called and the alertor will issue a warning sound; otherwise, it stops the buzzer.

```

10 if (digitalRead(PIN_BUTTON) == LOW) {
11   alert();
12 }else {
13   freq(PIN_BUZZER, 0, 10);
14 }
```

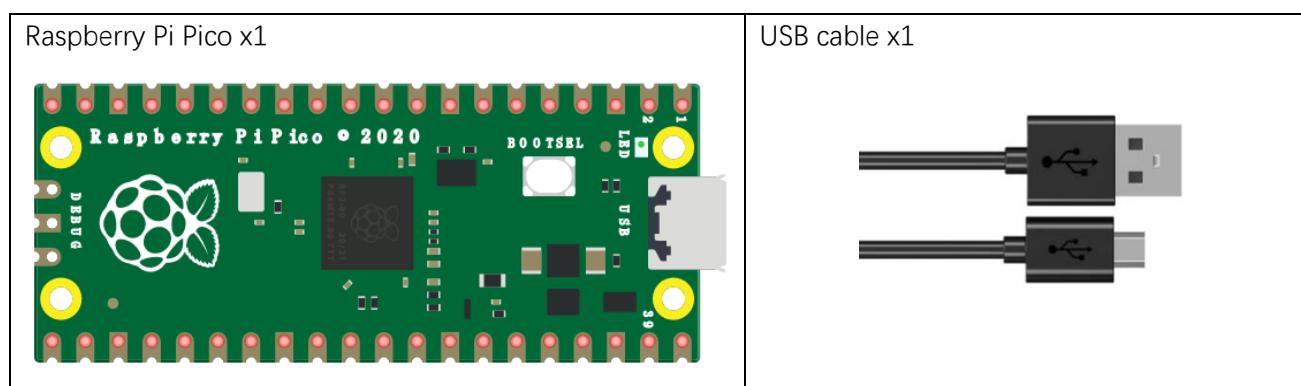
# Chapter 7 Serial Communication

Serial Communication is a means of Communication between different devices. This section describes Raspberry Pi Pico Serial Communication.

## Project 7.1 Serial Print

This project uses Raspberry Pi Pico serial communicator to send data to the computer and print it on the serial monitor.

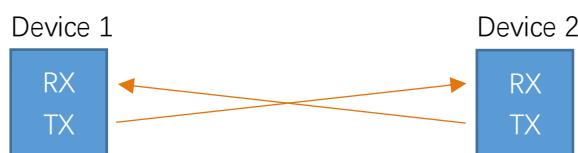
## Component List



## Related Knowledge

### Serial communication

Serial communication generally refers to the Universal Asynchronous Receiver/Transmitter (UART), which is commonly used in electronic circuit communication. It has two communication lines; one is responsible for sending data (TX line) and the other for receiving data (RX line). The serial communication connections two devices use is as follows:



Before serial communication starts, the baud rate of both sides must be the same. Communication between devices can work only if the same baud rate is used. The baud rates commonly used is 9600 and 115200.

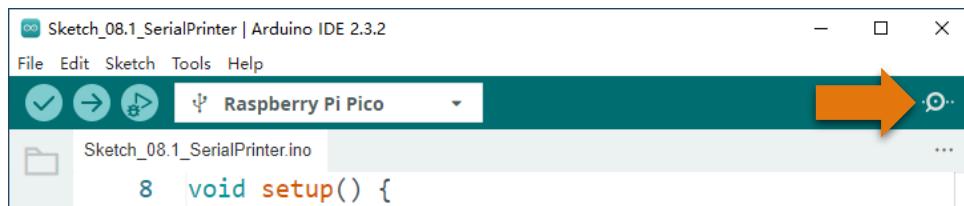
### Serial port on Raspberry Pi Pico

Raspberry Pi Pico has integrated USB to serial transfer, so it could communicate with computer connecting to USB cable.

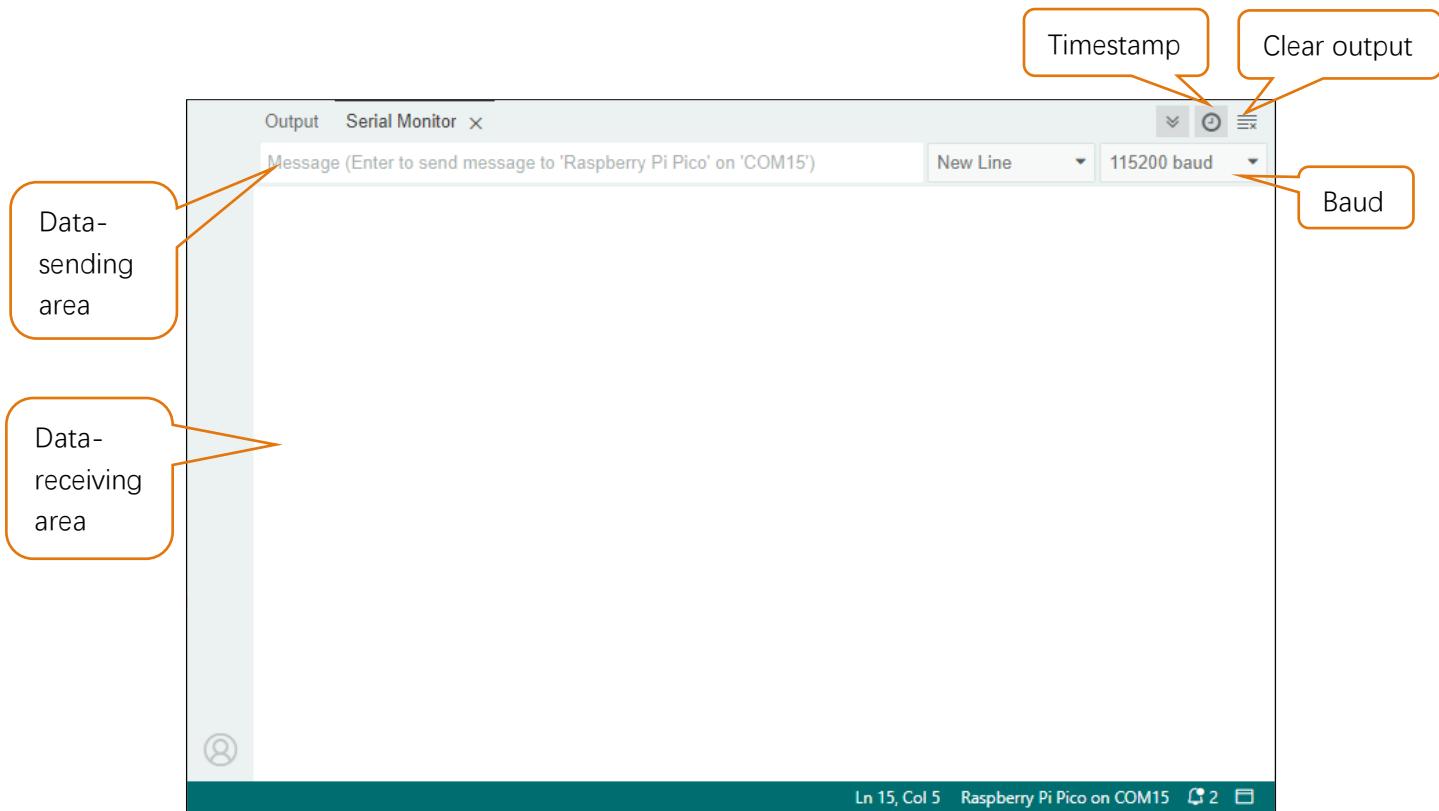


Arduino Software also uploads code to Pico through the serial connection.

Your computer identifies serial devices connecting to it as COMx. We can use the Serial Monitor window of Arduino Software to communicate with Pico, connect Pico to computer through the USB cable, choose the correct device, and then click the Serial Monitor icon to open the Serial Monitor window.

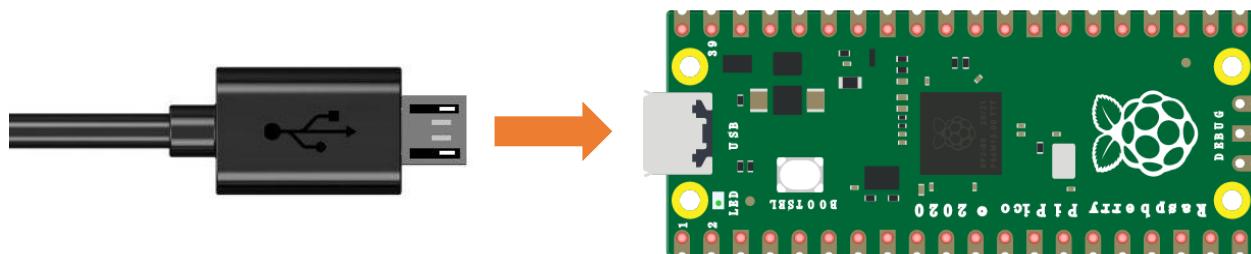


Interface of serial monitor window is as follows. If you cannot open it, make sure Pico has been connected to the computer, and choose the right serial port in the menu bar "Tools".



## Circuit

Connect Raspberry Pi Pico to the computer with USB cable.



## Sketch

### Sketch\_07.1\_SerialPrinter

The screenshot shows the Arduino IDE interface with the following details:

- Title Bar:** Sketch\_08.1\_SerialPrinter | Arduino IDE 2.3.2
- Menu Bar:** File Edit Sketch Tools Help
- Toolbox:** Includes icons for Save, Run, Stop, and others.
- Sketch Selection:** Raspberry Pi Pico
- Code Area:** Displays the following C++ code for `Sketch_08.1_SerialPrinter.ino`:

```
1 //*****
2 // Filename   : SerialPrinter
3 // Description : Use UART send some data to PC, and show them on serial
4 // monitor.
5 // Author     : www.freenove.com
6 // Modification: 2021/10/13
7 //*****
8 void setup() {
9     Serial.begin(115200);
10    delay(2000);
11    Serial.println("Raspberry Pi Pico initialization completed!");
12 }
13
14 void loop() {
15     Serial.println( millis() / 1000 % 60 );
16     delay(1000);
17 }
```

- Output Area:** Labeled "Output" at the top, currently empty.
- Status Bar:** Ln 18, Col 1 Raspberry Pi Pico on COM15 4 2

Download the code to Pico, open the serial port monitor, set the baud rate to 115200. As shown in the following picture:

```

15 | Serial.println( millis() / 1000 % 60 );
16 | delay(1000);
17 |

```

Set the relevant information

Output Serial Monitor x

Message (Enter to send message to 'Raspberry Pi Pico' on 'COM15') New Line 115200 baud

13:59:58.860 -> Raspberry Pi Pico initialization completed!  
13:59:58.860 -> 1  
13:59:59.831 -> 2  
14:00:00.830 -> 3  
14:00:01.830 -> 4  
14:00:02.868 -> 5  
14:00:03.864 -> 6  
14:00:04.830 -> 7  
14:00:05.851 -> 8

Above the trip is the system information, and below is the result of the code.

Ln 10, Col 10 Raspberry Pi Pico on COM15

As shown above, when the code runs, the data is printed every one second.

#### Reference

```
void begin(unsigned long baud, uint32_t config=SERIAL_8N1, int8_t rxPin=-1,
          int8_t txPin=-1, bool invert=false, unsigned long timeout_ms = 20000UL);
```

Initializes the serial port. Parameter baud is baud rate; other parameters generally use the default value.

```
size_t println( arg );
```

Print to the serial port and wrap. The parameter **arg** can be a number, a character, a string, an array of characters, etc.

```
size_t printf(const char * format, ...) __attribute__ ((format (printf, 2, 3)));
```

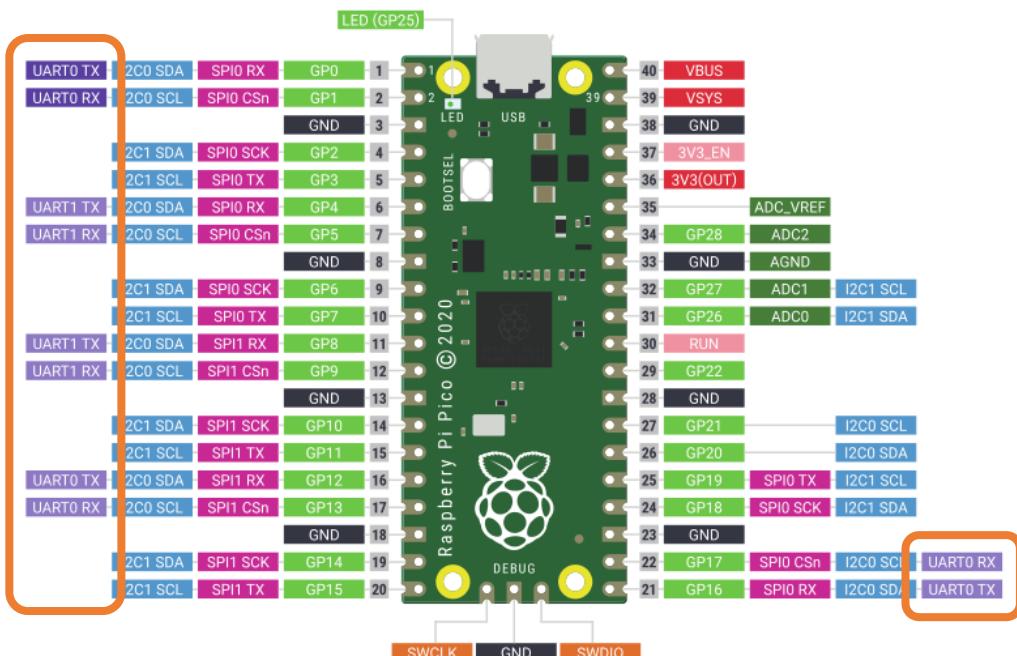
Print formatted content to the serial port in the same way as print in standard C.

```
unsigned long millis();
```

Returns the number of milliseconds since the current system was booted.

For details, please refer to [UART, I2C, SPI default pin](#).

You can change settings according to the distribution of pins.



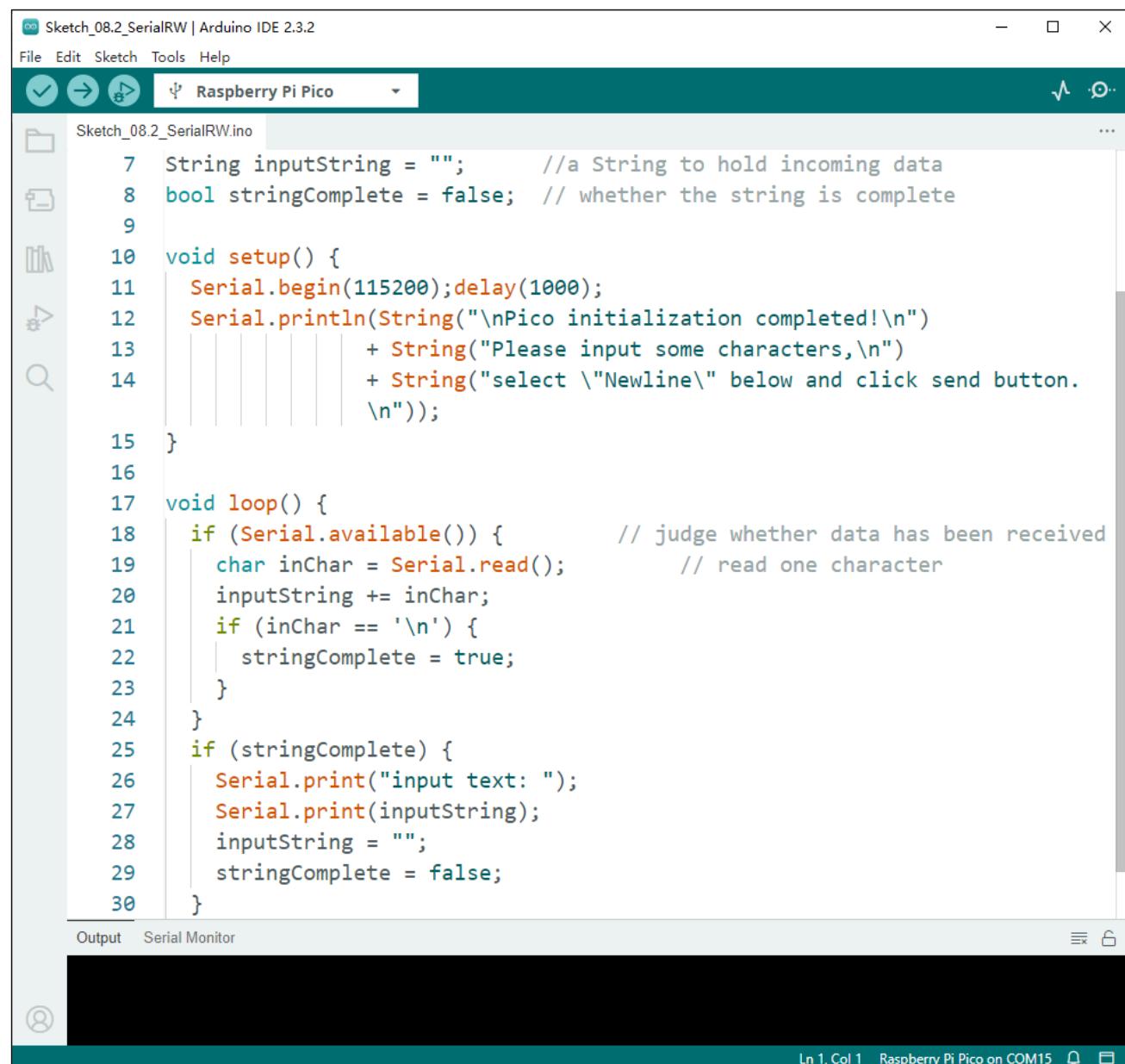
## Project 7.2 Serial Read and Write

From last section, we use serial port on Pico to send data to a computer, now we will use that to receive data from computer.

Component and circuit are the same as in the previous project.

### Sketch

#### Sketch\_07.2\_SerialRW



The screenshot shows the Arduino IDE interface with the following details:

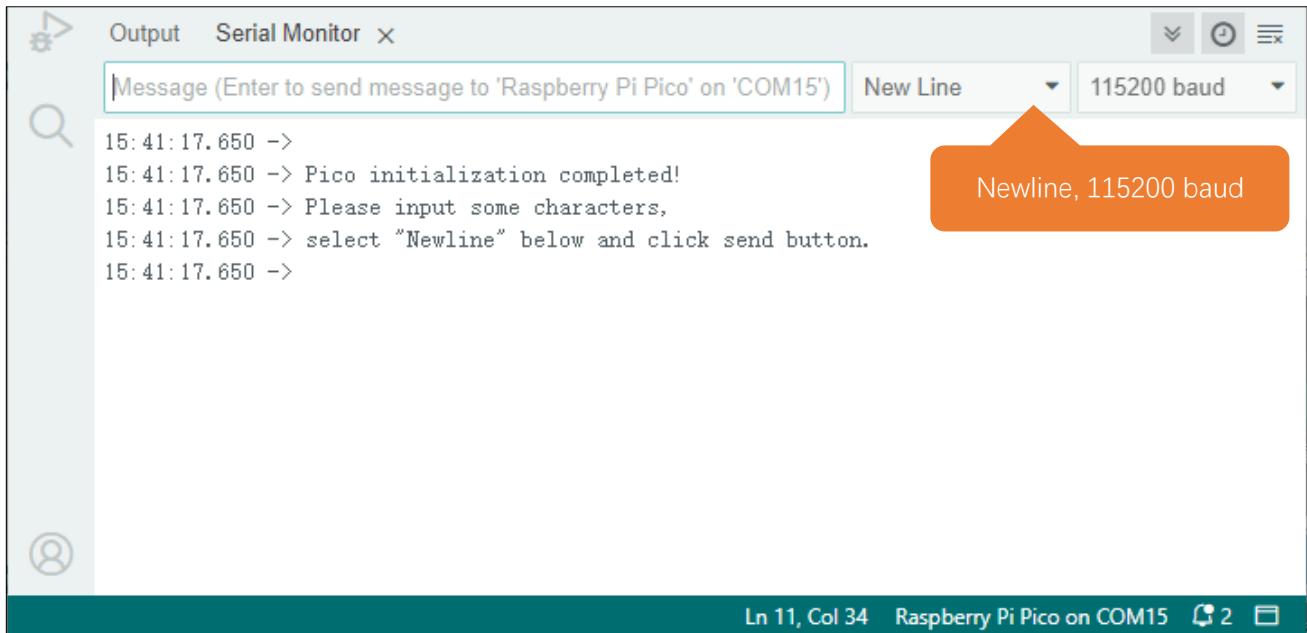
- Title Bar:** Sketch\_08.2\_SerialRW | Arduino IDE 2.3.2
- Menu Bar:** File Edit Sketch Tools Help
- Toolbar:** Includes icons for Save, Run, Stop, and a dropdown menu set to "Raspberry Pi Pico".
- Code Editor:** Displays the `Sketch_08.2_SerialRW.ino` file content. The code initializes the serial port at 115200 baud, prints a welcome message, and then enters a loop to read incoming characters. It checks if the received character is a newline (\n). If it is, it sets a flag `stringComplete` to true. Once `stringComplete` is true, it prints the received string and resets the variables.

```
String inputString = "";      //a String to hold incoming data
bool stringComplete = false; // whether the string is complete
void setup() {
    Serial.begin(115200);delay(1000);
    Serial.println(String("\nPico initialization completed!\n") +
                  + String("Please input some characters,\n") +
                  + String("select \"Newline\" below and click send button.\n"));
}
void loop() {
    if (Serial.available()) {      // judge whether data has been received
        char inChar = Serial.read(); // read one character
        inputString += inChar;
        if (inChar == '\n') {
            stringComplete = true;
        }
    }
    if (stringComplete) {
        Serial.print("input text: ");
        Serial.print(inputString);
        inputString = "";
        stringComplete = false;
    }
}
```

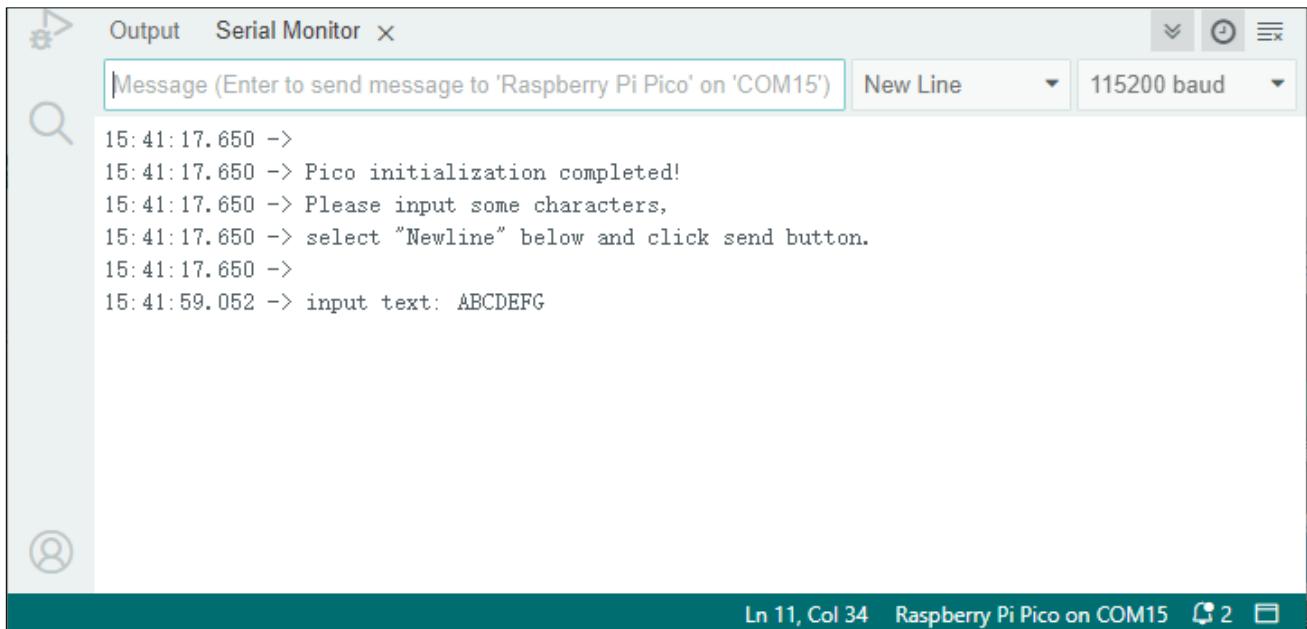
- Status Bar:** Shows "Ln 1, Col 1" and "Raspberry Pi Pico on COM15".



Download the code to Pico, open the serial monitor, and set the bottom to Newline, 115200, as shown in the following picture:



Input texts like "ABCDEFG" in the Message Bar and press Enter to print the data received by Pico.



The following is the program code:

```
1 String inputString = "";      //a String to hold incoming data
2 bool stringComplete = false; // whether the string is complete
3
4 void setup() {
5     Serial.begin(115200);delay(1000);
6     Serial.println(String("\nPico initialization completed!\n")
7                     + String("Please input some characters, \n")
8                     + String("select \"Newline\" below and click send button. \n"));
9 }
10
11 void loop() {
12     if (Serial.available()) {      // judge whether data has been received
13         char inChar = Serial.read();      // read one character
14         inputString += inChar;
15         if (inChar == '\n') {
16             stringComplete = true;
17         }
18     }
19     if (stringComplete) {
20         Serial.print("input text: ");
21         Serial.print(inputString);
22         inputString = "";
23         stringComplete = false;
24     }
25 }
```

In loop(), determine whether the serial port has data, if so, read and save the data, and if the newline character is read, print out all the data that has been read.

#### Reference

**String();**

Constructs an instance of the String class.

For more information, please visit

<https://www.arduino.cc/reference/en/language/variables/data-types/stringobject/>

**int available(void);**

Get the number of bytes (characters) available for reading from the serial port. This is data that has already arrived and stored in the serial receive buffer.

**Serial.read();**

Reads incoming serial data.

# Chapter 8 AD Converter

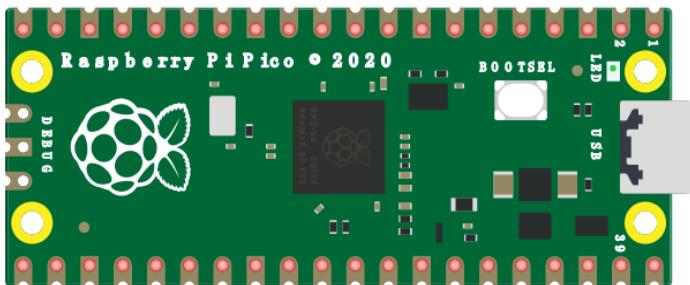
This chapter we learn to use the ADC function of Raspberry Pi Pico.

## Project 8.1 Read the Voltage of Potentiometer

In this chapter, we use ADC function of Pico to read the voltage output by potentiometer.

### Component List

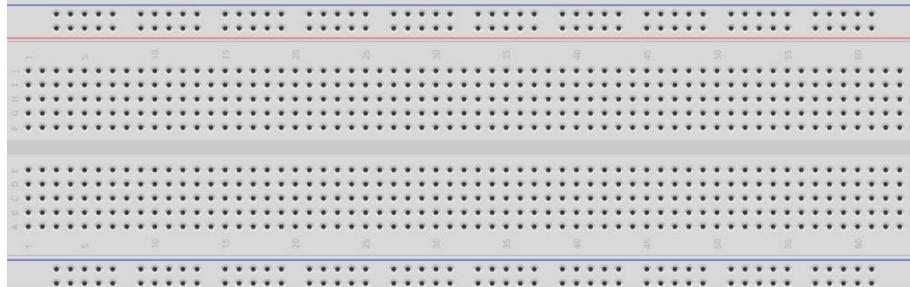
Raspberry Pi Pico x1



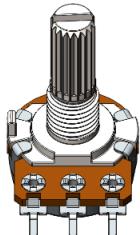
USB cable x1



Breadboard x1



Rotary potentiometer x1



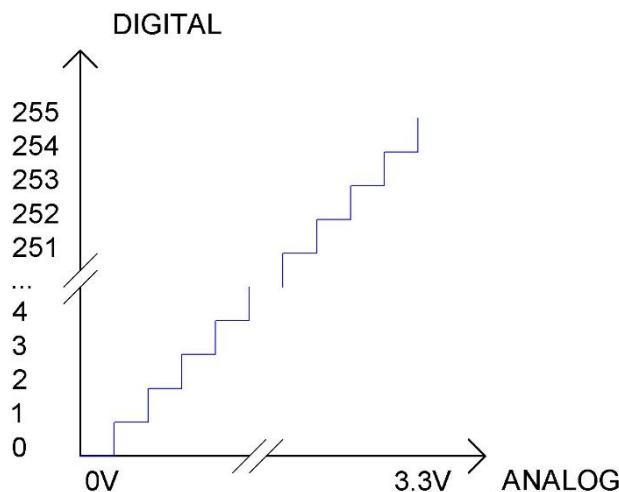
Jumper



## Related Knowledge

### ADC

An ADC is an electronic integrated circuit used to convert analog signals such as voltages to digital or binary form consisting of 1s and 0s. The range of our ADC on Pico is 10 bits, which means the resolution is  $2^{10}=1024$ , and it represents a range (at 3.3V) will be divided equally to 1024 parts. The range of analog values corresponds to ADC values. So the more bits the ADC has, the denser the partition of analog will be and the greater the precision of the resulting conversion.



Subsection 1: the analog in rang of 0V---3.3/1023 V corresponds to digital 0;

Subsection 2: the analog in rang of 3.3/1023V---2\*3.3/1023V corresponds to digital 1;

...

The following analog will be divided accordingly.

The conversion formula is as follows:

$$ADC\ Value = \frac{Analog\ Voltage}{3.3} * 1023$$

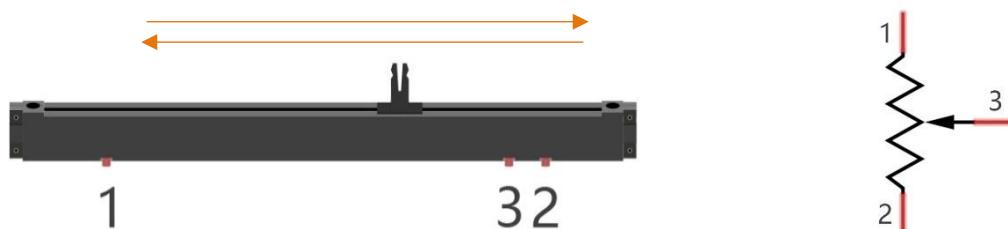
### ADC Channels Raspberry Pi Pico

Raspberry Pi Pico has four ADC channels, which are ADC0(GP26), ADC1(GP27), ADC2(GP28), ADC3(GP29). ADC3 used to measure VSYS on Pico board. Therefore, there are only three generic ADC channels that can be directly used, namely, ADC0, ADC1 and ADC2.

## Component Knowledge

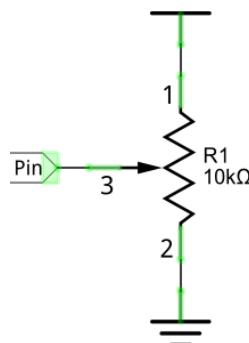
### Potentiometer

Potentiometer is a resistive element with three Terminal parts. Unlike the resistors that we have used thus far in our project which have a fixed resistance value, the resistance value of a potentiometer can be adjusted. A potentiometer is often made up by a resistive substance (a wire or carbon element) and movable contact brush. When the brush moves along the resistor element, there will be a change in the resistance of the potentiometer's output side (3) (or change in the voltage of the circuit that is a part). The illustration below represents a linear sliding potentiometer and its electronic symbol on the right.



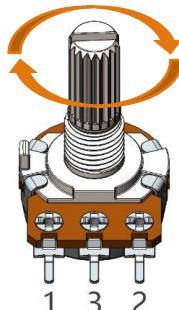
Between potentiometer pin 1 and pin 2 is the resistive element (a resistance wire or carbon) and pin 3 is connected to the brush that makes contact with the resistive element. In our illustration, when the brush moves from pin 1 to pin 2, the resistance value between pin 1 and pin 3 will increase linearly (until it reaches the highest value of the resistive element) and at the same time the resistance between pin 2 and pin 3 will decrease linearly and conversely down to zero. At the midpoint of the slider, the measured resistance values between pin 1 and 3 and between pin 2 and 3 will be the same.

In a circuit, both sides of resistive element are often connected to the positive and negative electrodes of power. When you slide the brush "pin 3", you can get variable voltage within the range of the power supply.



### Rotary potentiometer

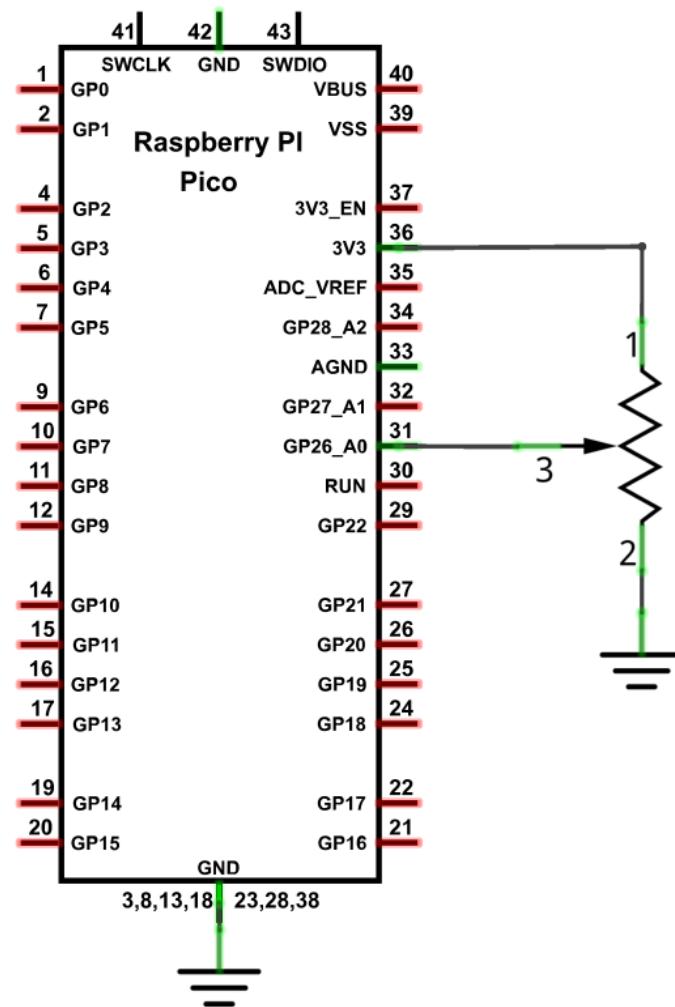
Rotary potentiometers and linear potentiometers have the same function; the only difference being the physical action being a rotational rather than a sliding movement.



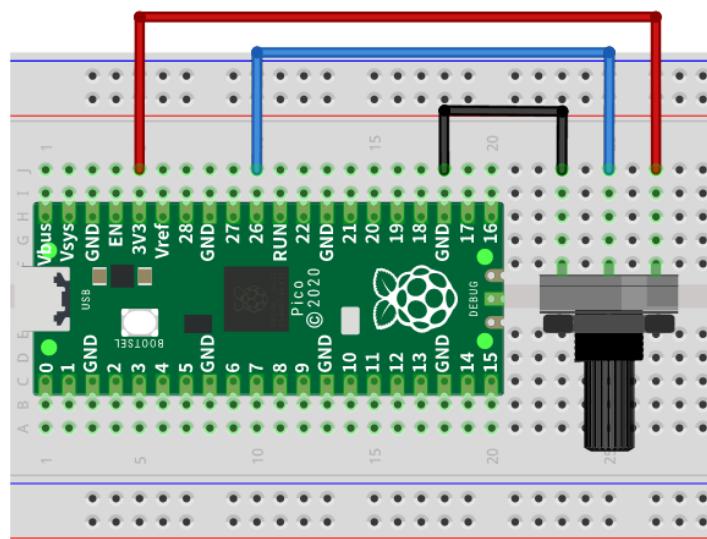
Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

## Circuit

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: [support@freenove.com](mailto:support@freenove.com)



Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)



## Sketch

### Sketch\_08.1\_ADC

```

Sketch_08.1_ADC | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Sketch_08.1_ADC.ino
 7 #define PIN_ANALOG_IN 26
 8
 9 void setup() {
10   Serial.begin(115200);
11 }
12
13 void loop() {
14   int adcVal = analogRead(PIN_ANALOG_IN);
15   double voltage = adcVal / 1023.0 * 3.3;
16   Serial.println("ADC Value: " + String(adcVal) + " --- Voltage Value: "
17     + String(voltage) + "V");
18   delay(500);
}

```

Output

Ln 1, Col 1 Raspberry Pi Pico on COM15 2

Download the code to Pico, open the serial monitor, and set the baud rate to 115200, as shown in the following picture,

Output Serial Monitor

Message (Enter to send message to 'Raspberry Pi Pico' on 'COM15')

New Line 115200 baud

```

14:21:22.183 -> ADC Value: 245 --- Voltage Value: 0.79V
14:21:22.725 -> ADC Value: 245 --- Voltage Value: 0.79V
14:21:23.183 -> ADC Value: 246 --- Voltage Value: 0.79V
14:21:23.719 -> ADC Value: 243 --- Voltage Value: 0.78V
14:21:24.198 -> ADC Value: 245 --- Voltage Value: 0.79V
14:21:24.711 -> ADC Value: 244 --- Voltage Value: 0.79V
14:21:25.206 -> ADC Value: 245 --- Voltage Value: 0.79V
14:21:25.684 -> ADC Value: 244 --- Voltage Value: 0.79V
14:21:26.184 -> ADC Value: 244 --- Voltage Value: 0.79V
14:21:26.717 -> ADC Value: 244 --- Voltage Value: 0.79V
14:21:27.228 -> ADC Value: 245 --- Voltage Value: 0.79V
14:21:27.685 -> ADC Value: 245 --- Voltage Value: 0.79V
14:21:28.231 -> ADC Value: 246 --- Voltage Value: 0.79V
14:21:28.685 -> ADC Value: 243 --- Voltage Value: 0.78V
14:21:29.223 -> ADC Value: 244 --- Voltage Value: 0.79V

```

Ln 1, Col 1 Raspberry Pi Pico on COM15 2

The following is the code:

```
1 #define PIN_ANALOG_IN 26
2
3 void setup() {
4     Serial.begin(115200);
5 }
6
7 void loop() {
8     int adcVal = analogRead(PIN_ANALOG_IN);
9     double voltage = adcVal / 1023.0 * 3.3;
10    Serial.println("ADC Value: " + String(adcVal) + " --- Voltage Value: " + String(voltage) +
11        "V");
12    delay(500);
13 }
```

In loop() function, analogRead is called to get the ADC value of ADC0 and assign it to adcVal. Calculate the measured voltage value through the formula, and print these data through the serial port monitor.

```
8 int adcVal = analogRead(PIN_ANALOG_IN);
9     double voltage = adcVal / 1023.0 * 3.3;
10    Serial.println("ADC Value: " + String(adcVal) + " --- Voltage Value: " + String(voltage) +
11        "V");
```

#### Reference

`uint16_t analogRead(uint8_t pin);`

Reads the value from the specified analog pin. Return the analog reading on the pin. (0-1023 for 10 bits).



# Chapter 9 Potentiometer & LED

We have learnt to use ADC in the previous chapter. In this chapter, we will combine PWM and ADC to use potentiometer to control LED, RGBLED and Neopixel.

## Project 9.1 Soft Light

In this project, we will make a soft light. We will use an ADC Module to read ADC values of a potentiometer and map it to duty cycle of the PWM used to control the brightness of an LED. Then you can change the brightness of an LED by adjusting the potentiometer.

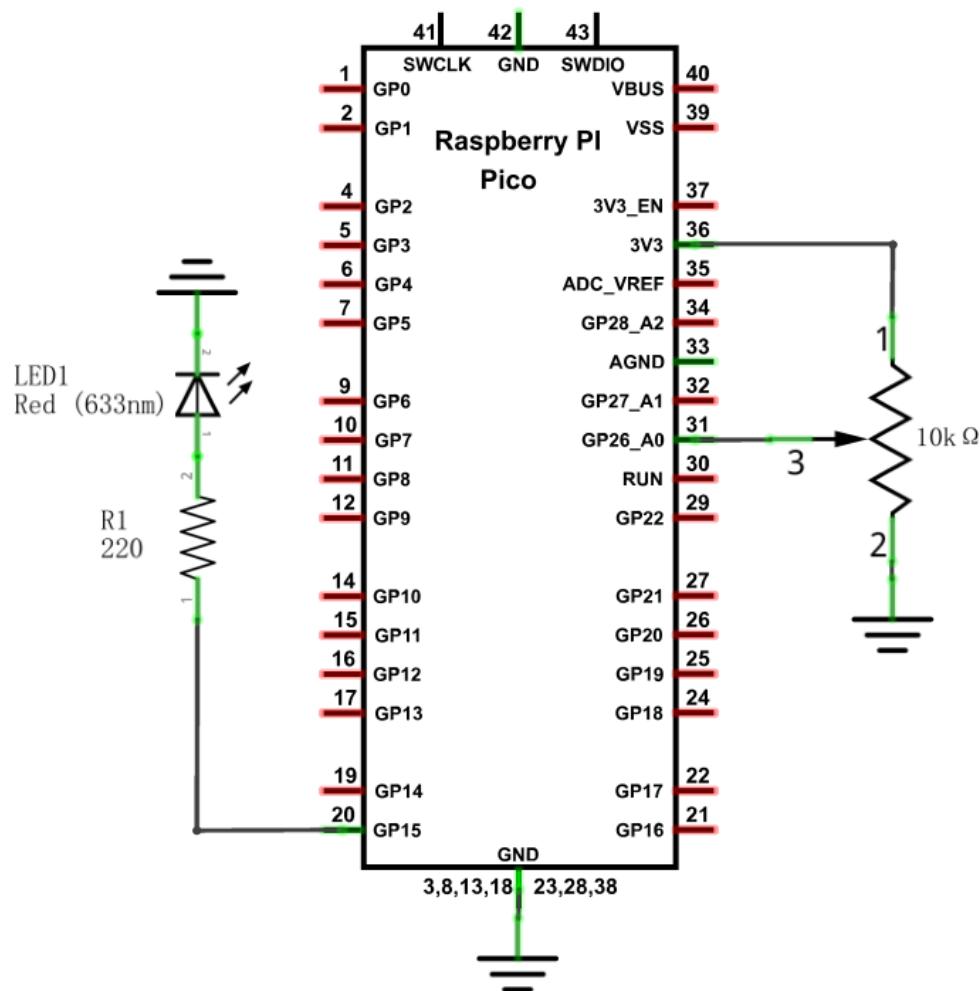
## Component List

Raspberry Pi Pico x1		USB cable x1	
Breadboard x1			
Rotary potentiometer x1	Resistor 220Ω x1	LED x1	Jumper
			

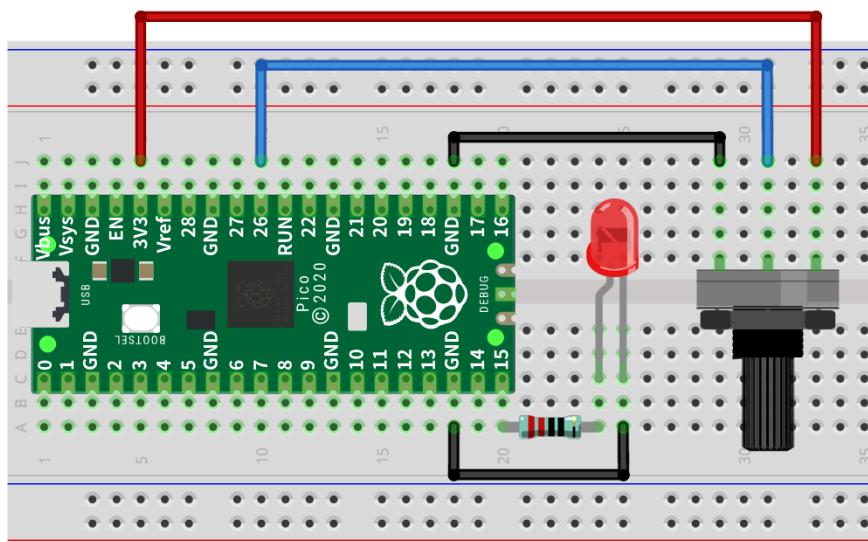
Any concerns? ✉ [support@freenove.com](mailto:support@freenove.com)

## Circuit

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: [support@freenove.com](mailto:support@freenove.com)



Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)



## Sketch

### Sketch\_09.1\_Softlight

The screenshot shows the Arduino IDE interface with the following details:

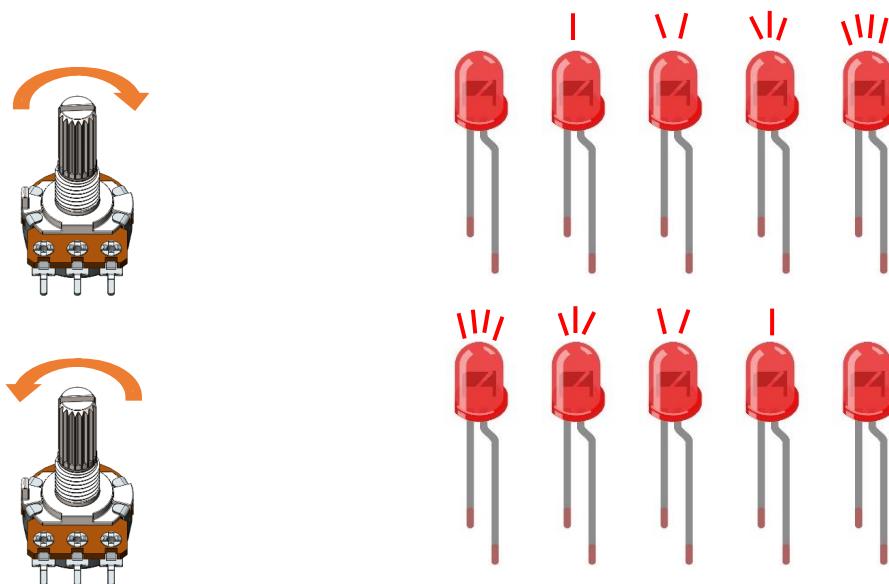
- Title Bar:** Sketch\_10.1\_SoftLight | Arduino IDE 2.3.2
- File Menu:** File Edit Sketch Tools Help
- Sketch Selection:** Sketch\_10.1\_SoftLight.ino
- Code Area:** The code is displayed in the main area, starting with #defines for PIN\_ADC0 (26) and PIN\_LED (15), followed by setup() and loop() functions.
- Output Area:** Below the code area, there is a large black box labeled "Output".
- Status Bar:** indexing: 2/44, Ln 1, Col 1, Raspberry Pi Pico on COM15, a refresh icon, and a close icon.

```
#define PIN_ADC0 26
#define PIN_LED 15

void setup() {
    pinMode(PIN_LED, OUTPUT);
}

void loop() {
    int adcVal = analogRead(PIN_ADC0); //read adc
    analogWrite(PIN_LED, map(adcVal, 0, 1023, 0, 255));
    delay(10);
}
```

Download the code to Pico, by turning the adjustable resistor to change the input voltage of GP26, Pico changes the output voltage of GP15 according to this voltage value, thus changing the brightness of the LED.



The following is the code:

```
1 #define PIN_ADC0      26
2 #define PIN_LED       15
3
4 void setup() {
5     pinMode(PIN_LED, OUTPUT);
6 }
7
8 void loop() {
9     int adcVal = analogRead(PIN_ADC0); //read adc
10    analogWrite(PIN_LED, map(adcVal, 0, 1023, 0, 255));
11    delay(10);
12 }
```

In the code, read the ADC value of potentiometer and map it to the duty cycle of PWM to control LED brightness.

If you have any concerns, please contact us via [support@freenove.com](mailto:support@freenove.com)



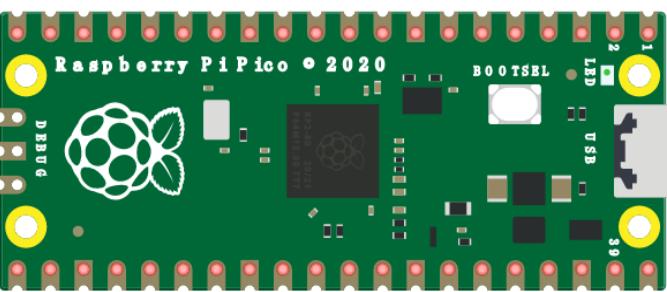
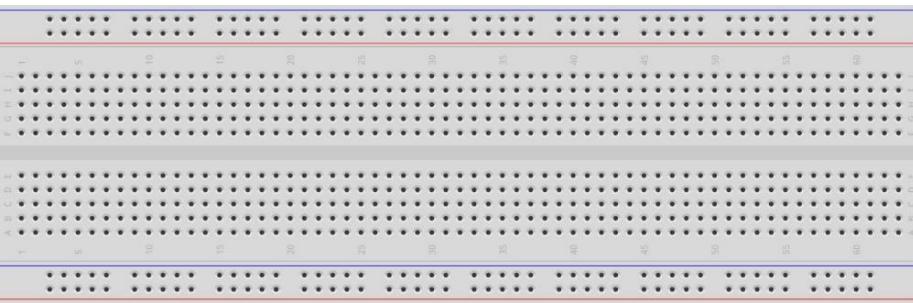
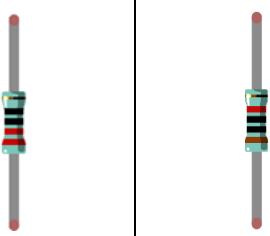
# Chapter 10 Photoresistor & LED

In this chapter, we will learn how to use photoresistor.

## Project 10.1 Control LED through Photoresistor

A Photoresistor is very sensitive to the amount of light present. We can take advantage of the characteristic to make a night lamp with the following function: when the ambient light is less (darker environment) the LED will automatically become brighter to compensate and when the ambient light is greater (brighter environment) the LED will automatically dim to compensate.

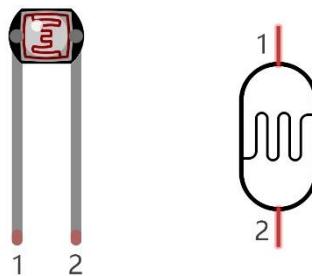
### Component List

Raspberry Pi Pico x1		USB cable x1
Breadboard x1		
Photoresistor x1 		Resistor 220Ω x1      10KΩ x1 
LED x1		Jumper

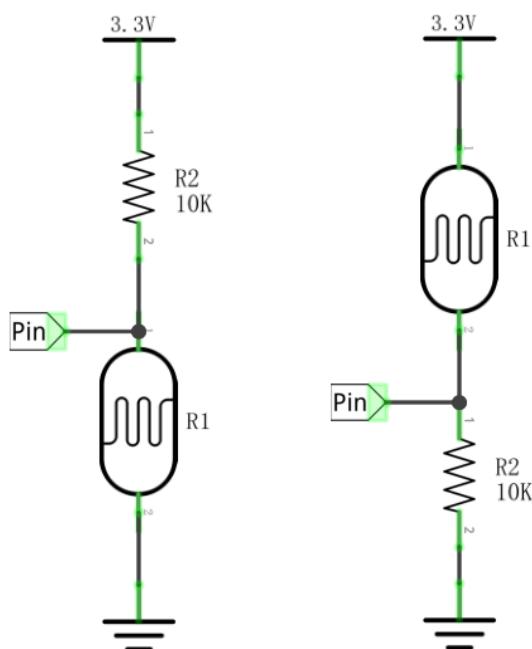
## Component Knowledge

### Photoresistor

Photoresistor is simply a light sensitive resistor. It is an active component that decreases resistance with respect to receiving luminosity (light) on the component's light sensitive surface. Photoresistor's resistance value will change in proportion to the ambient light detected. With this characteristic, we can use a Photoresistor to detect light intensity. The Photoresistor and its electronic symbol are as follows.



The circuit below is used to detect the change of a Photoresistor's resistance value:

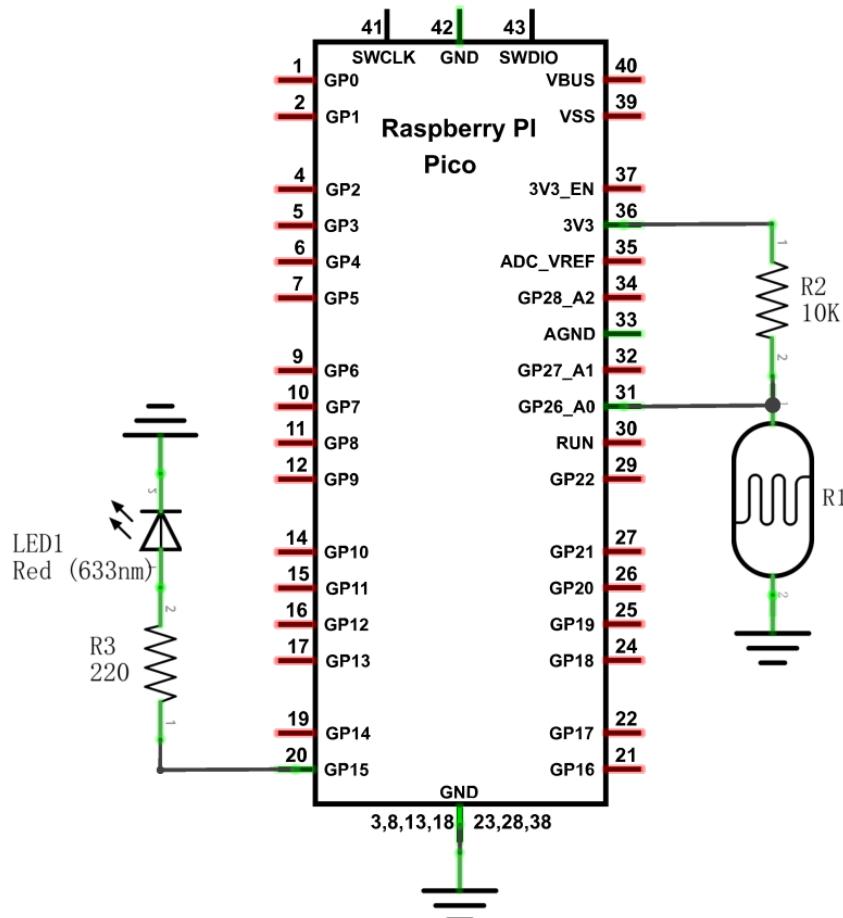


In the above circuit, when a Photoresistor's resistance value changes due to a change in light intensity, the voltage between the Photoresistor and Resistor R1 will also change. Therefore, the intensity of the light can be obtained by measuring this voltage.

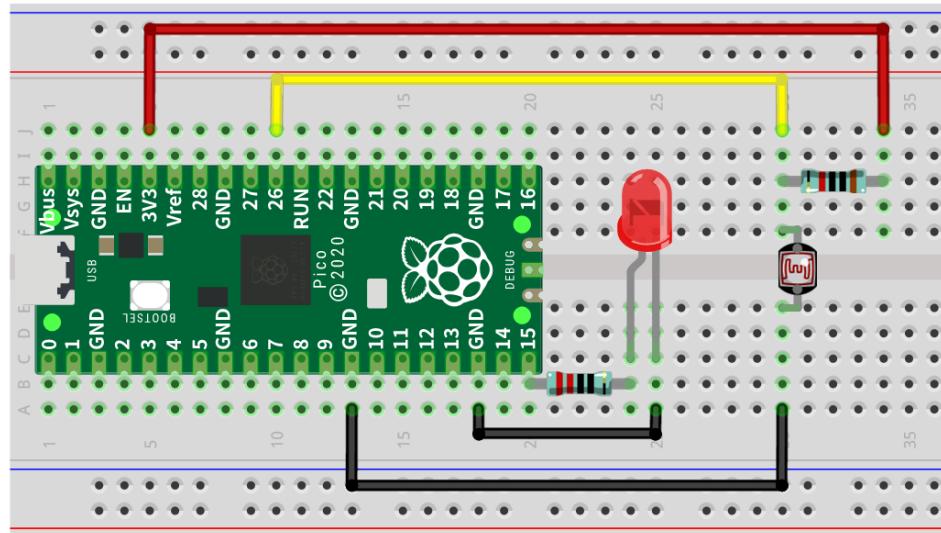
## Circuit

The circuit of this project is similar to SoftLight. The only difference is that the input signal is changed from a potentiometer to a combination of a photoresistor and a resistor.

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: [support@freenove.com](mailto:support@freenove.com)



Any concerns? ✉ [support@freenove.com](mailto:support@freenove.com)

## Sketch

The circuit used is similar to the project Soft Light. The only difference is that the input signal of the ADC0 pin of ADC changes from a potentiometer to a combination of a photoresistor and a resistor.

### Sketch\_10.1\_Nightlamp

```

Sketch_11.1_Photosensitive | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Sketch_11.1_Photosensitive.ino
7 #define PIN_ADC0      26
8 #define PIN_LED       15
9
10 void setup() {
11   pinMode(PIN_LED, OUTPUT);
12 }
13
14 void loop() {
15   int adcVal = analogRead(PIN_ADC0); //Read the voltage of the
                                         photoresistor
16   analogWrite(PIN_LED, map(adcVal, 0, 1023, 0, 255));
17   delay(10);
18 }

```

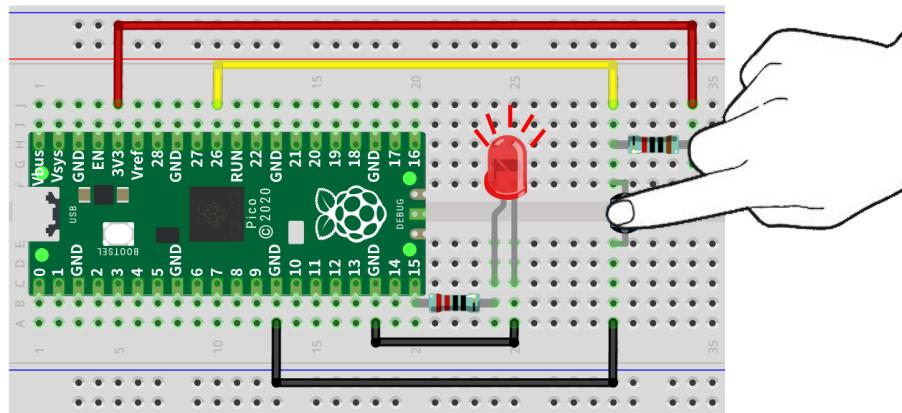
Output

Ln 1, Col 1 Raspberry Pi Pico on COM15

Download the code to Pico, if you cover the photoresistor or increase the light shining on it, the brightness of the LED changes accordingly.

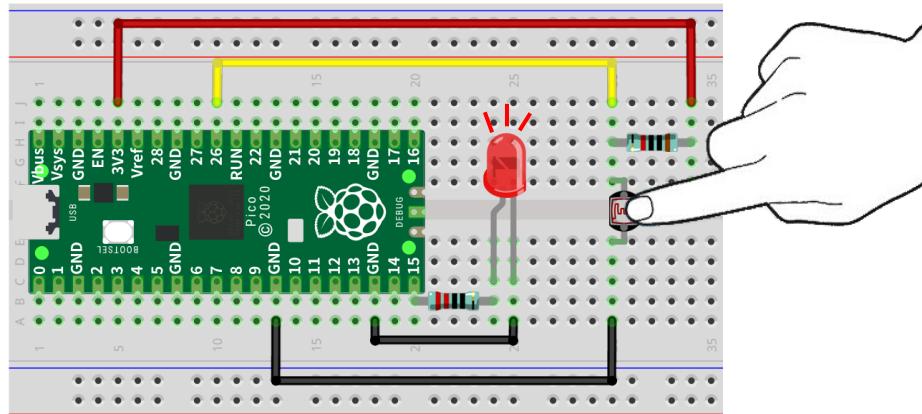
If you have any concerns, please contact us via: [support@freenove.com](mailto:support@freenove.com)

Fully cover the photoresistor:

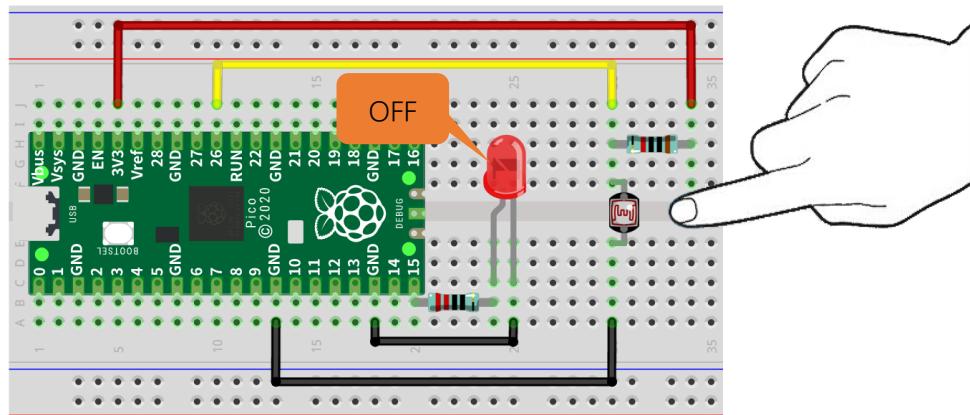


Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

Half cover the photoresistor:



Not cover the photoresistor:



The following is the program code:

```

1 #define PIN_ADC0      26
2 #define PIN_LED       15
3
4 void setup() {
5     pinMode(PIN_LED, OUTPUT);
6 }
7
8 void loop() {
9     int adcVal = analogRead(PIN_ADC0); //Read the voltage of the photoresistor
10    analogWrite(PIN_LED, map(adcVal, 0, 1023, 0, 255));
11    delay(10);
12 }
```

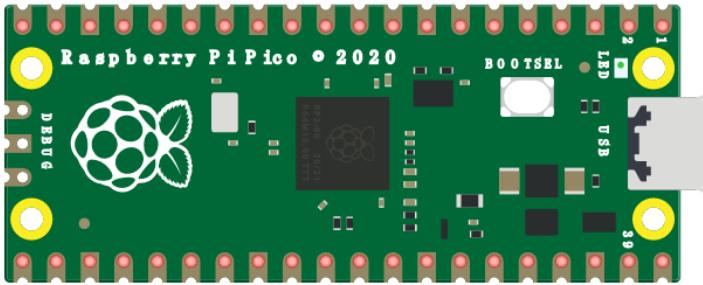
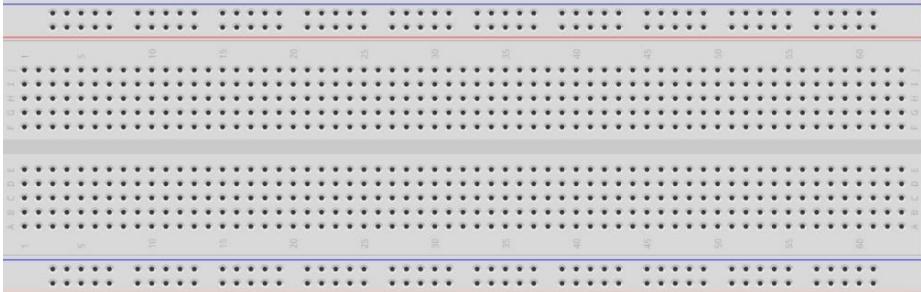
# Chapter 11 Thermistor

In this chapter, we will learn about Thermistors that are another kind of Resistor.

## Project 11.1 Thermometer

A Thermistor is a type of Resistor whose resistance value is dependent on temperature and changes in temperature. Therefore, we can take advantage of this characteristic to make a thermometer.

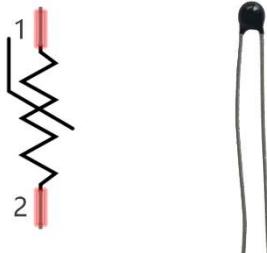
### Component List

Raspberry Pi Pico x1	USB cable x1	
		
Breadboard x1		
Thermistor x1	Resistor 10kΩ x1	Jumper

## Component Knowledge

### Thermistor

A Thermistor is a temperature sensitive resistor. When it senses a change in temperature, the resistance of the Thermistor will change. We can take advantage of this characteristic by using a Thermistor to detect temperature intensity. A Thermistor and its electronic symbol are shown below.



The relationship between resistance value and temperature of a thermistor is:

$$R_t = R * \text{EXP} \left[ B * \left( \frac{1}{T_2} - \frac{1}{T_1} \right) \right]$$

**Where:**

Rt is the thermistor resistance under T2 temperature;

R is the nominal resistance of thermistor under T1 temperature;

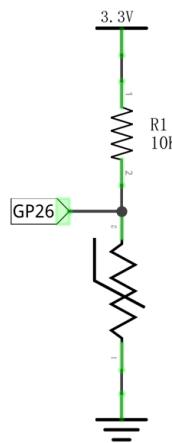
**EXP[n]** is nth power of e;

B is for thermal index;

T1, T2 is Kelvin temperature (absolute temperature). Kelvin temperature=273.15 + Celsius temperature.

For the parameters of the Thermistor, we use: B=3950, R=10kΩ, T1=25°C.

The circuit connection method of the Thermistor is similar to photoresistor, as the following:



We can use the value measured by the ADC converter to obtain the resistance value of Thermistor, and then we can use the formula to obtain the temperature value.

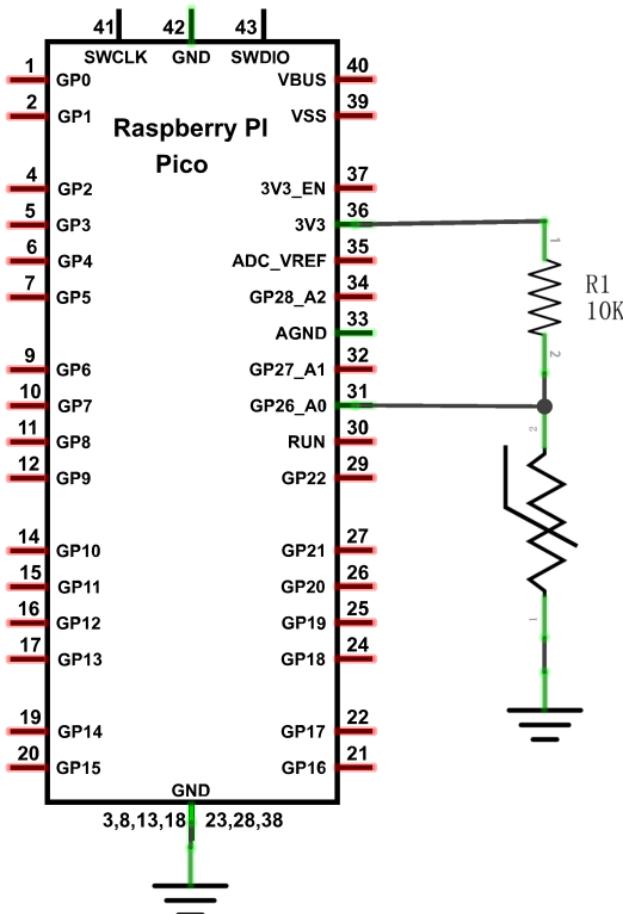
Therefore, the temperature formula can be derived as:

$$T_2 = 1 / \left( \frac{1}{T_1} + \ln \left( \frac{R_t}{R} \right) / B \right)$$

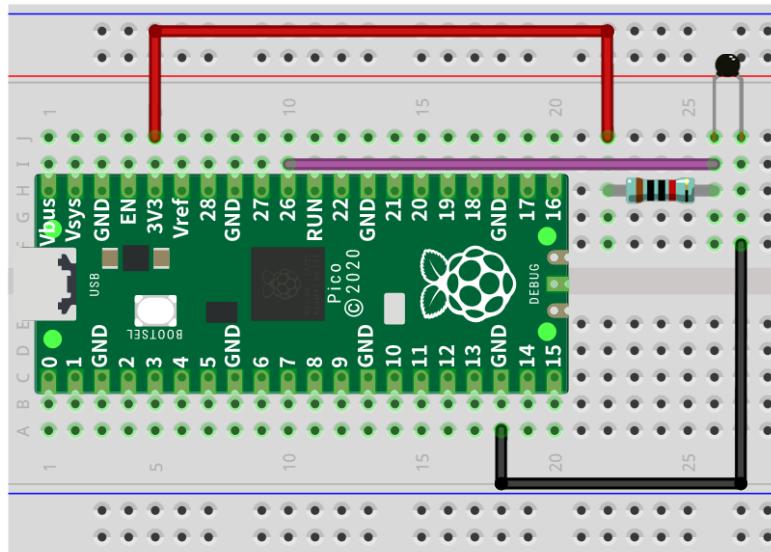
## Circuit

The circuit of this project is similar to the one in the previous chapter. The only difference is that the Photoresistor is replaced by a Thermistor.

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: [support@freenove.com](mailto:support@freenove.com)



Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)



## Sketch

### Sketch\_11.1\_Thermometer

```

Sketch_12.1_Thermometer | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Sketch_12.1_Thermometer.ino
1 #define PIN_ADC0 26
2 void setup() {
3     Serial.begin(115200);
4 }
5
6 void loop() {
7     int adcValue = analogRead(PIN_ADC0); //read ADC pin
8     double voltage = (float)adcValue / 1023.0 * 3.3; // calculate
9     voltage
10    double Rt = 10 * voltage / (3.3 - voltage); //calculate
11    resistance value of thermistor
12    double tempK = 1 / (1 / (273.15 + 25) + log(Rt / 10) / 3950.0); //calculate
13    temperature (Kelvin)
14    double tempC = tempK - 273.15; //calculate
15    temperature (Celsius)
16    Serial.println("Voltage: " + String(voltage) + "V,\t\t" + "Kelvins: " + String
17    (tempK) + "K,\t\t" + "Temperature: " + String(tempC) + "C");
18    delay(1000);
19 }
20

```

The screenshot shows the Arduino IDE interface with the sketch file open. The code reads the ADC value from pin 26, calculates the voltage, then uses the Steinhart-Hart formula to calculate the temperature in Kelvin and Celsius. It then prints these values to the serial monitor. The serial monitor window is visible at the bottom, showing the real-time data being sent from the Pico.

Upload the code to Pico and serial monitor will display the current ADC, voltage and temperature values. Try to “pinch” the thermistor (without touching the leads) with your index finger and thumb for a brief time, you should see that the temperature value increases.

Voltage (V)	Kelvins (K)	Temperature (C)
1.52V	301.86K	28.71C
1.53V	301.58K	28.43C
1.52V	301.86K	28.71C
1.54V	301.31K	28.16C
1.52V	301.67K	28.52C
1.52V	301.67K	28.52C
1.52V	301.67K	28.52C
1.53V	301.49K	28.34C
1.54V	301.22K	28.07C
1.52V	301.67K	28.52C
1.54V	301.31K	28.16C
1.53V	301.49K	28.34C
1.53V	301.58K	28.43C
1.52V	301.67K	28.52C
1.53V	301.58K	28.43C

The serial monitor output shows a series of measurements. The first few lines show stable values. Then, after a pinch, the voltage drops slightly, causing a significant increase in both kelvin and celsius temperatures. After the pinch ends, the values return to their previous levels.

If you have any concerns, please contact us via: [support@freenove.com](mailto:support@freenove.com)

Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

The following is the code:

```
1 #define PIN_ADC0 26
2 void setup() {
3     Serial.begin(115200);
4 }
5
6 void loop() {
7     int adcValue = analogRead(PIN_ADC0); //read ADC pin
8     double voltage = (float)adcValue / 1023.0 * 3.3;// calculate voltage
9     double Rt = 10 * voltage / (3.3 - voltage);//calculate resistance value of thermistor
10    double tempK = 1 / (1 / (273.15 + 25) + log(Rt / 10) / 3950.0); //calculate temperature
11    (Kelvin)
12    double tempC = tempK - 273.15; //calculate temperature (Celsius)
13    Serial.println("Voltage: " + String(voltage) + "V,\t\t" + "Kelvins: " + String(tempK) +
14    "K,\t" + "Temperature: " + String(tempC) + "C");
15    delay(1000);
}
```

In the code, the ADC value of ADC module A0 port is read, and then calculates the voltage and the resistance of thermistor according to Ohms Law. Finally, it calculates the temperature sensed by the thermistor, according to the formula.



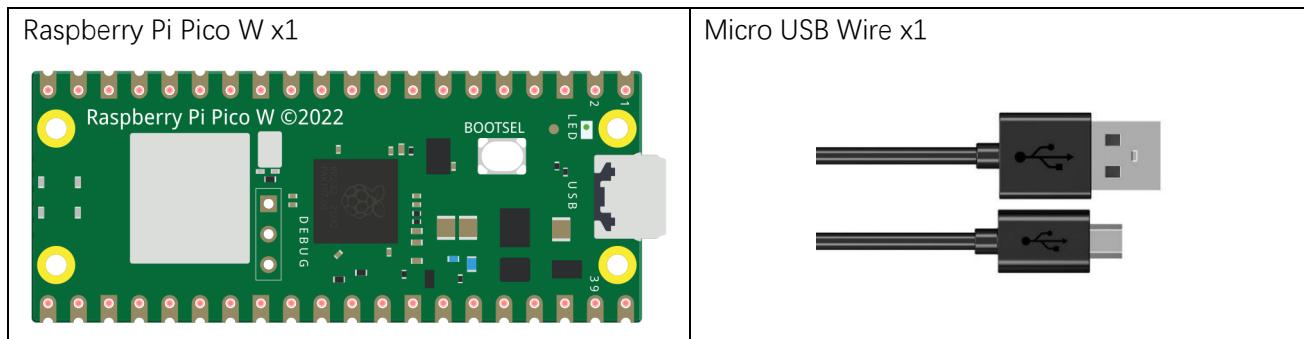
# Chapter 12 WiFi Working Modes (Only for Pico W)

The biggest difference between the raspberry pi pico and the raspberry pi Pico W is that the raspberry pi pico W is equipped with a WiFi function module. At the beginning of this chapter, we will learn about the WiFi function of Pico W of Raspberry Pi.

If you have Pico in your hand, please change it to Pico W before continuing to learn.

## Project 12.1 Station mode

### Component List



### Component knowledge

#### Wireless

Pico W has an on-board 2.4GHz wireless interface using an Infineon CYW43439. The antenna is an onboard antenna licensed from ABRACON (formerly ProAnt). The wireless interface is connected via SPI to the RP2040.

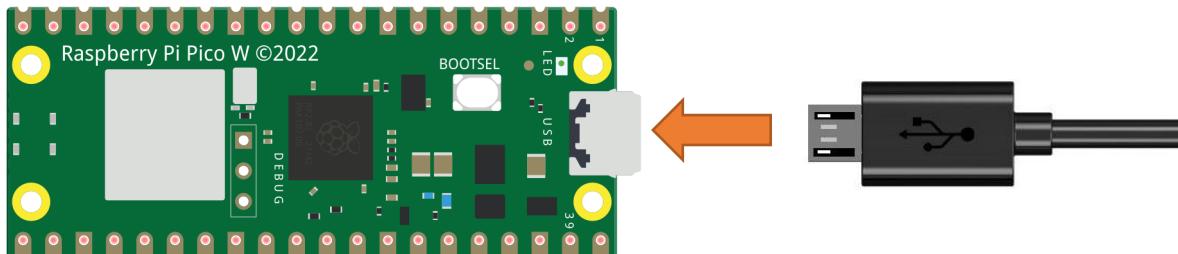
#### Station mode

When Pico W 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 Pico W wants to communicate with the PC, it needs to be connected to the router.



## Circuit

Connect Pico W to the computer using the USB cable.





## Sketch

### Sketch\_12.1\_Station\_mode

```
Sketch_30.1_WiFi_Station | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Sketch_30.1_WiFi_Station.ino
1
2
3
4
5
6
7 #include <WiFi.h>
8
9 const char *ssid_Router      = "*****"; //Enter the router name
10 const char *password_Router = "*****"; //Enter the router password
11
12 void setup(){
13   Serial.begin(115200);
14   delay(2000);
15   Serial.println("Setup start");
16   WiFi.begin(ssid_Router, password_Router);
17   Serial.println(String("Connecting to ") + ssid_Router);
18   while (WiFi.status() != WL_CONNECTED){
19     delay(500);
20     Serial.print(".");
21   }
--
```

Output

Ln 9, Col 31 Raspberry Pi Pico W on COM9 2

Enter the correct Router name and password.

Because the names and passwords of routers 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 Pico W, open serial monitor and set baud rate to 115200. Then it will display as follows:

The screenshot shows the Arduino Serial Monitor window. The title bar says "Output Serial Monitor X". The message area contains the following text:

```

Message (Enter to send message to 'Raspberry Pi Pico W' on 'COM9')
10:42:43.052 -> Setup start
10:42:50.076 -> Connecting to FYI_2.4G
10:42:50.076 ->
10:42:50.076 -> Connected, IP address:
10:42:50.125 -> 192.168.1.23
10:42:50.125 -> Setup End

```

The status bar at the bottom right shows "Ln 9, Col 42 Raspberry Pi Pico W on COM9" and a bell icon with the number 2.

When PICO W successfully connects to “ssid\_Router”, serial monitor will print out the IP address assigned to PICO W by the router.

The following is the program code:

```

1 #include <WiFi.h>
2
3 const char *ssid_Router      = "*****"; //Enter the router name
4 const char *password_Router = "*****"; //Enter the router password
5
6 void setup() {
7     Serial.begin(115200);
8     delay(2000);
9     Serial.println("Setup start");
10    WiFi.begin(ssid_Router, password_Router);
11    Serial.println(String("Connecting to ") + ssid_Router);
12    while (WiFi.status() != WL_CONNECTED) {
13        delay(500);
14        Serial.print(".");
15    }
16    Serial.println("\nConnected, IP address: ");
17    Serial.println(WiFi.localIP());
18    Serial.println("Setup End");
19 }
20 void loop() {
21 }
```

Include the WiFi Library header file of Pico W.

```
1 #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 Pico W in Station mode and connect it to your router.

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

Check whether Pico W 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 Pico W.

```
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):** PICO W 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:** boolean optional parameter, defaulting to true. If set as false, then PICO W 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():** obtain IP address in Station mode

**disconnect():** disconnect wifi

## Project 12.2 AP mode

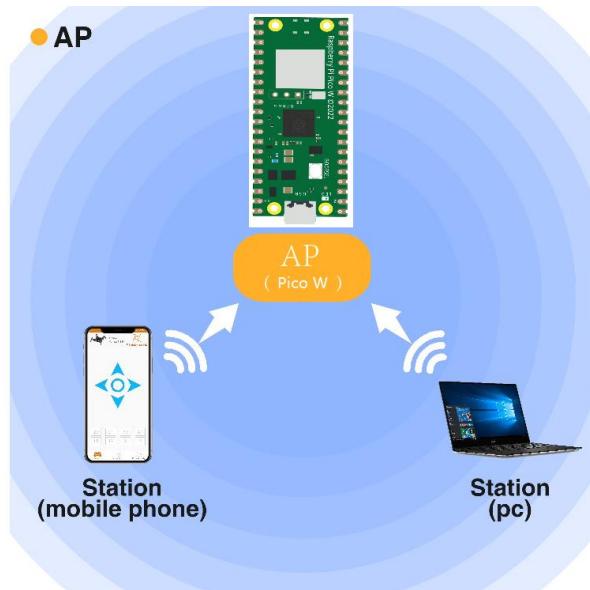
### Component List & Circuit

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

### Component knowledge

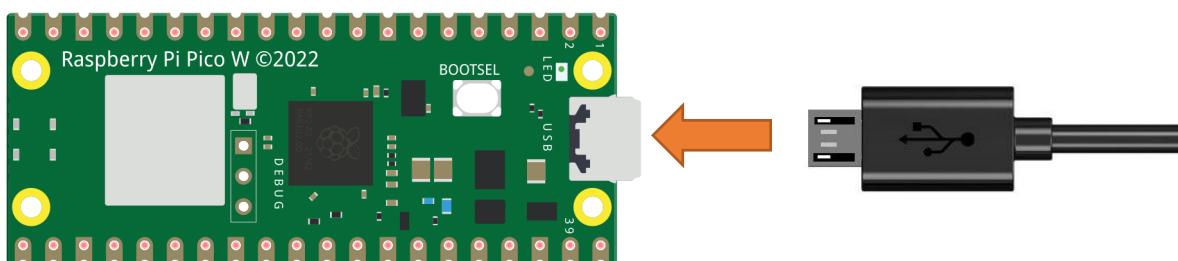
#### AP mode

When PICO W 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, PICO W is used as a hotspot. If a mobile phone or PC wants to communicate with PICO W, it must be connected to the hotspot of PICO W. Only after a connection is established with PICO W can they communicate.



### Circuit

Connect Pico W to the computer using the USB cable.





## Sketch

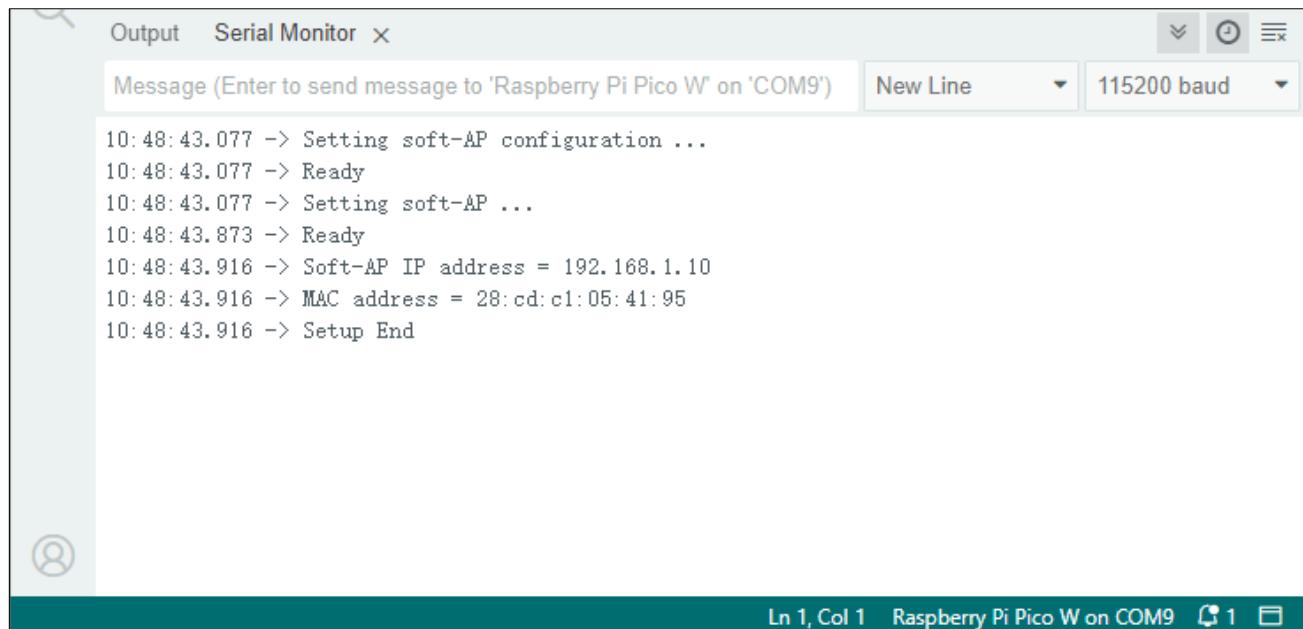
### Sketch\_12.2\_AP\_mode

```
Sketch_30.2_WiFi_AP | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Sketch_30.2_WiFi_AP.ino
7 #include <WiFi.h>
8
9 const char *ssid_AP      = "WiFi_Name"; //Enter the router name
10 const char *password_AP = "12345678"; //Enter the router password
11
12 IPAddress local_IP(192,168,1,100); //Set the IP address of ESP8266 itself
13 IPAddress gateway(192,168,1,10); //Set the gateway of ESP8266 itself
14 IPAddress subnet(255,255,255,0); //Set the subnet mask for itself
15
16 void setup(){
17     Serial.begin(115200);
18     delay(2000);
19     Serial.println("Setting soft-AP configuration ... ");
20     WiFi.disconnect();
```

Set a name and a password for PICO W AP mode.

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

Compile and upload codes to PICO W, open the serial monitor and set the baud rate to 115200. Then it will display as follows.



The screenshot shows the 'Serial Monitor' window with the title 'Output Serial Monitor'. The message area displays the following log entries:

```

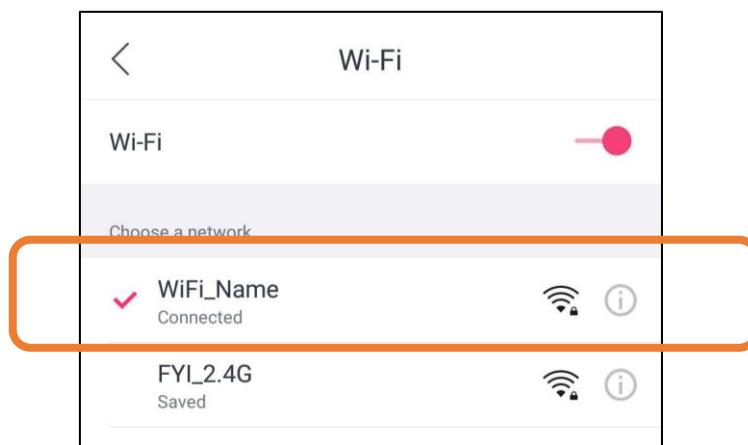
Message (Enter to send message to 'Raspberry Pi Pico W' on 'COM9')
New Line 115200 baud

10:48:43.077 -> Setting soft-AP configuration ...
10:48:43.077 -> Ready
10:48:43.077 -> Setting soft-AP ...
10:48:43.873 -> Ready
10:48:43.916 -> Soft-AP IP address = 192.168.1.10
10:48:43.916 -> MAC address = 28:cd:c1:05:41:95
10:48:43.916 -> Setup End

```

The status bar at the bottom indicates 'Ln 1, Col 1 Raspberry Pi Pico W on COM9'.

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 PICO W, 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.



Note:

1. Every time you change the WiFi name and password in the code, please power off and then on again, and then upload the code. It is possible that the WiFi name and password have not actually changed due to the direct uploading of code without power. This is because Pico W WiFi module and RP2040 chip are separated. Only when the power is cut off can the WiFi name and password be flashed to the WiFi module again.
2. Pico W executes this code only to open a WiFi hotspot, and does not configure the code related to online data transmission, so the mobile phone will display no data after connection.



The following is the program code:

```

1 #include <WiFi.h>
2
3 const char *ssid_AP      = "WiFi_Name"; //Enter the router name
4 const char *password_AP = "12345678"; //Enter the router password
5
6 IPAddress local_IP(192, 168, 1, 100); //Set the IP address of PICO W itself
7 IPAddress gateway(192, 168, 1, 10); //Set the gateway of PICO W itself
8 IPAddress subnet(255, 255, 255, 0); //Set the subnet mask for PICO W itself
9
10 void setup() {
11     Serial.begin(115200);
12     delay(2000);
13     Serial.println("Setting soft-AP configuration ... ");
14     WiFi.disconnect();
15     WiFi.mode(WIFI_AP);
16     Serial.println(WiFi.softAPConfig(local_IP, gateway, subnet) ? "Ready" : "Failed!");
17     Serial.println("Setting soft-AP ... ");
18     boolean result = WiFi.softAP(ssid_AP, password_AP);
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");
27 }
28
29 void loop() {
30 }
```

Include WiFi Library header file of PICO W.

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

Enter correct AP name and password.

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

Set PICO W in AP mode.

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

Configure IP address, gateway and subnet mask for PICO W.

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

Turn on an AP in PICO W, whose name is set by ssid\_AP and password is set by password\_AP.

```
18 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 PICO W. 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():** obtain IP address in AP mode

**softAPdisconnect ():** disconnect AP mode.



## Project 12.3 AP+Station mode

### Component List & Circuit

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

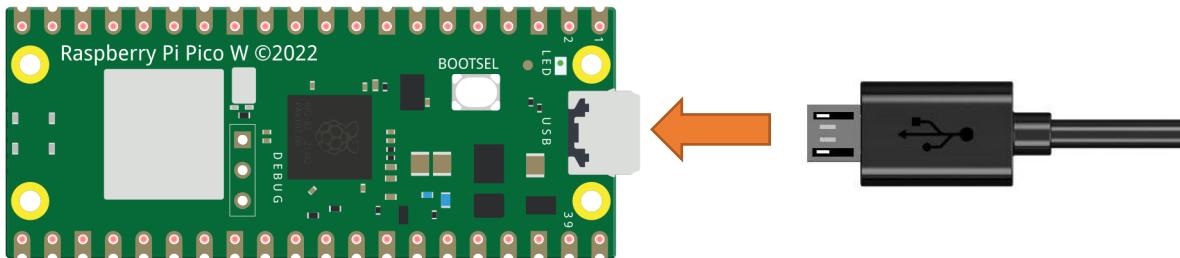
### Component knowledge

#### AP+Station mode

PICO W currently does not support simultaneous use of AP mode and Station mode, so this section can be skipped. In the actual mode configuration, the last configured mode shall prevail.

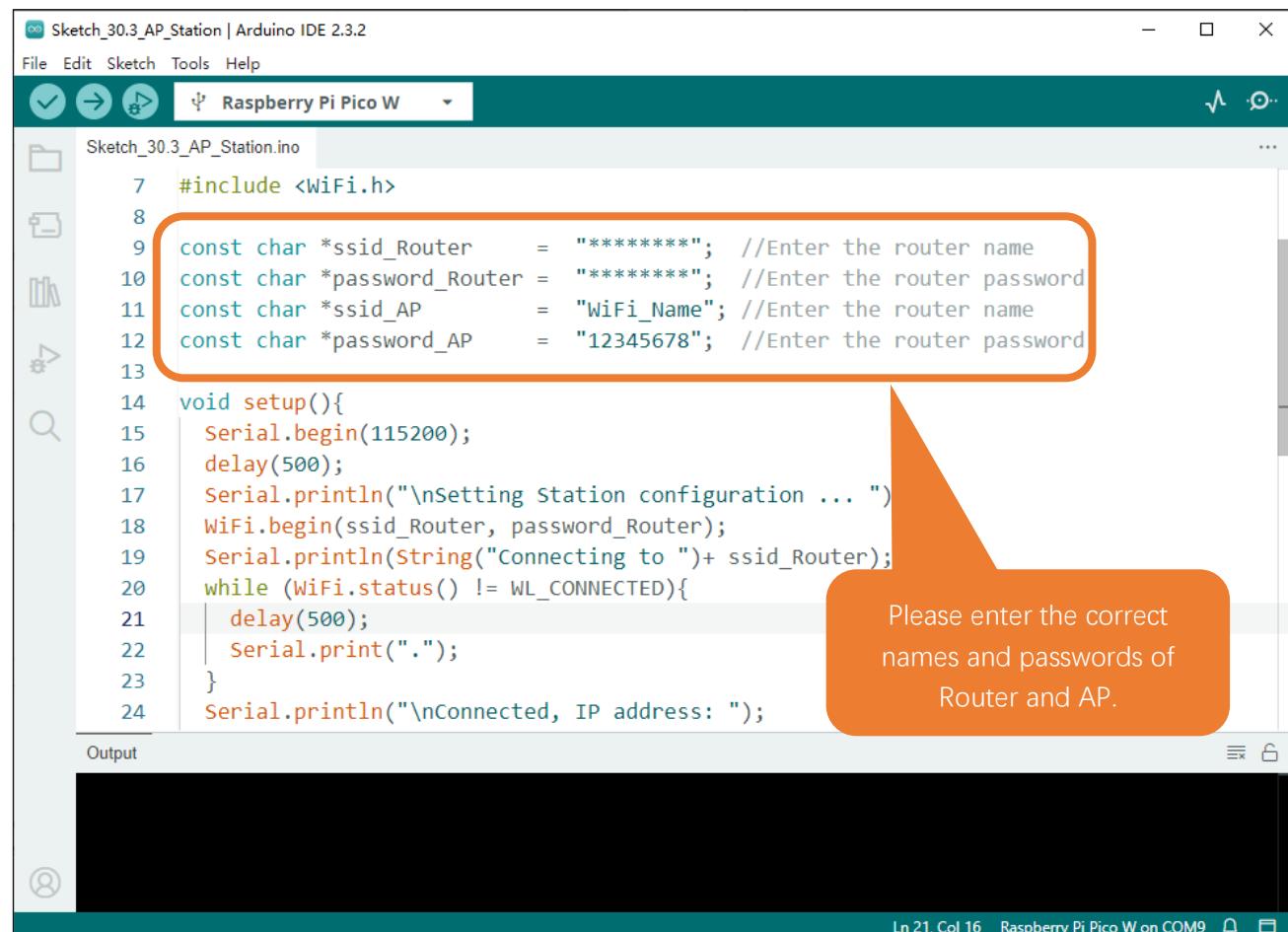
### Circuit

Connect Pico W to the computer using the USB cable.



## Sketch

### Sketch\_12.3\_AP\_Station\_mode



The screenshot shows the Arduino IDE interface with the sketch `Sketch_12.3_AP_Station_mode`. The code is as follows:

```
7 #include <WiFi.h>
8
9 const char *ssid_Router      = "*****"; //Enter the router name
10 const char *password_Router = "*****"; //Enter the router password
11 const char *ssid_AP         = "WiFi_Name"; //Enter the router name
12 const char *password_AP     = "12345678"; //Enter the router password
13
14 void setup(){
15     Serial.begin(115200);
16     delay(500);
17     Serial.println("\nSetting Station configuration ... ")
18     WiFi.begin(ssid_Router, password_Router);
19     Serial.println(String("Connecting to ")+ ssid_Router);
20     while (WiFi.status() != WL_CONNECTED){
21         delay(500);
22         Serial.print(".");
23     }
24     Serial.println("\nConnected, IP address: ");
```

A callout bubble with an orange border and arrow points to the configuration parameters from line 9 to line 12. The text inside the bubble reads:

Please enter the correct names and passwords of Router and AP.

It is analogous to Project 12.1 and Project 12.2. 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 PICO W, open serial monitor and set baud rate to 115200. Then it will display as follows:

The screenshot shows the Arduino Serial Monitor window. The title bar says "Output Serial Monitor". The message area contains the following log output:

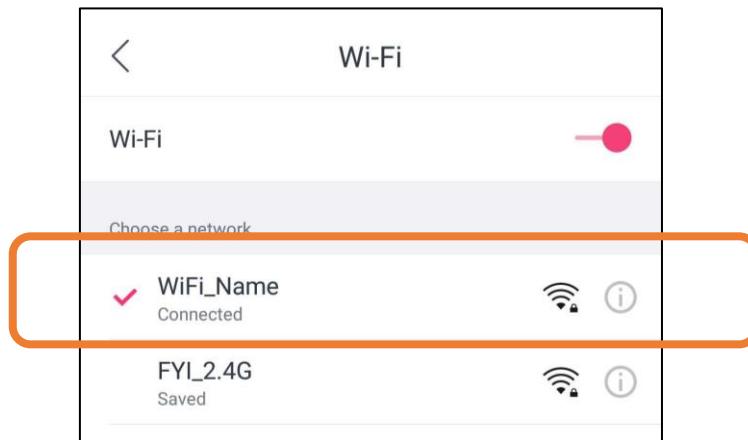
```

Message (Enter to send message to 'Raspberry Pi Pico W' on 'COM9')
10:53:45.024 -> Connecting to FYI_2.4G
10:53:45.024 ->
10:53:45.024 -> Connected, IP address:
10:53:45.024 -> 192.168.1.23
10:53:45.518 -> Setting soft-AP configuration ...
10:53:45.518 -> Setting soft-AP ...
10:53:46.357 -> Ready
10:53:46.357 -> Soft-AP IP address = 192.168.1.1
10:53:46.357 -> MAC address = 28:cd:c1:05:41:95
10:53:46.357 -> Setup End

```

The status bar at the bottom right shows "Ln 15, Col 24 Raspberry Pi Pico W on COM9" and icons for a bell and a refresh.

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 PICO W.



The following is the program code:

```

1 #include <WiFi.h>
2
3 const char *ssid_Router      = "*****"; //Enter the router name
4 const char *password_Router = "*****"; //Enter the router password
5 const char *ssid_AP          = "WiFi_Name"; //Enter the AP name
6 const char *password_AP      = "12345678"; //Enter the AP password
7
8 void setup() {
9     Serial.begin(115200);

```

```
10 Serial.println("Setting soft-AP configuration ... ");
11 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 }
```



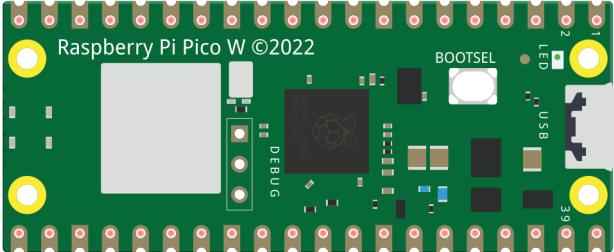
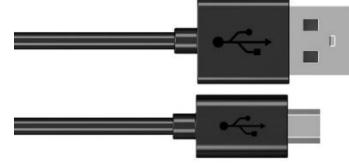
# Chapter 13 TCP/IP (Only for Pico W)

In this chapter, we will introduce how PICO W 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 13.1 as Client

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

### Component List

Raspberry Pi Pico W x1	Micro USB Wire x1
	

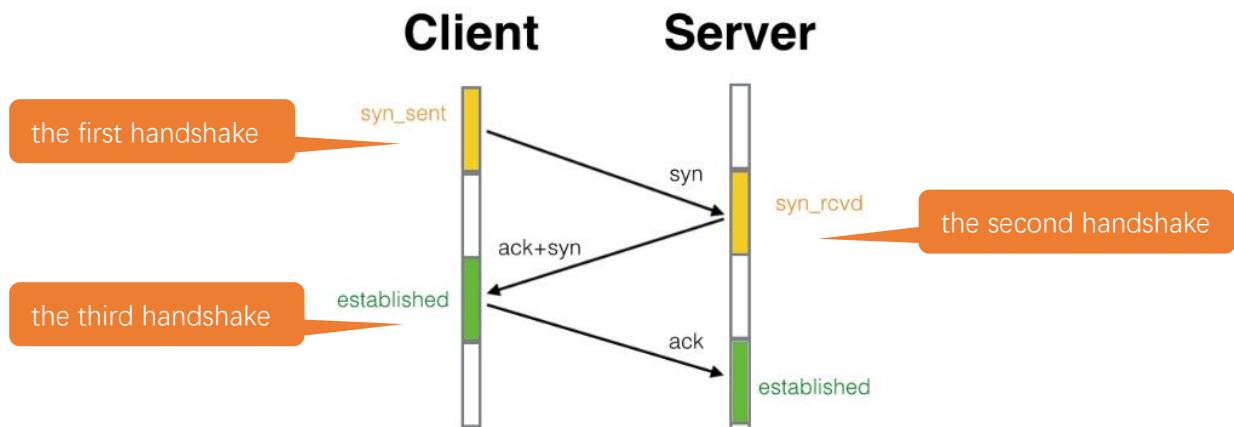
### Component knowledge

#### TCP connection

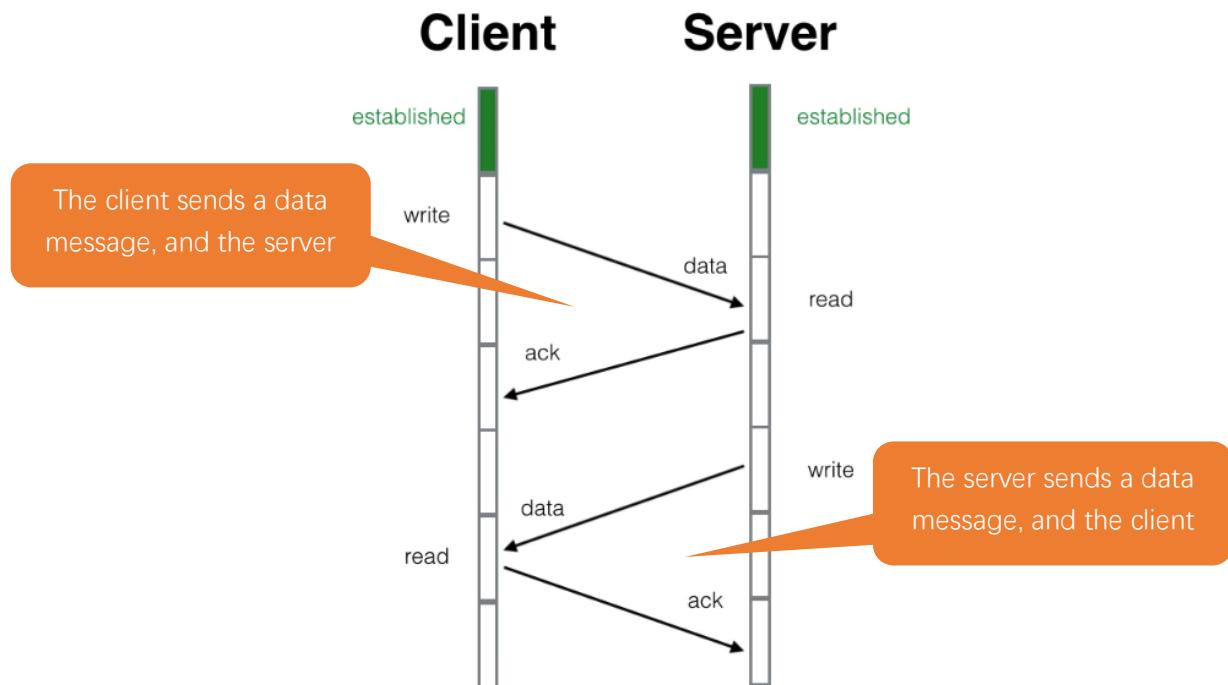
Before transmitting data, TCP needs to establish a logical connection between the sending end and the receiving end. It provides reliable and error-free data transmission between the two computers. In the TCP connection, the client and the server must be clarified. The client sends a connection request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

Three-time 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 have 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.

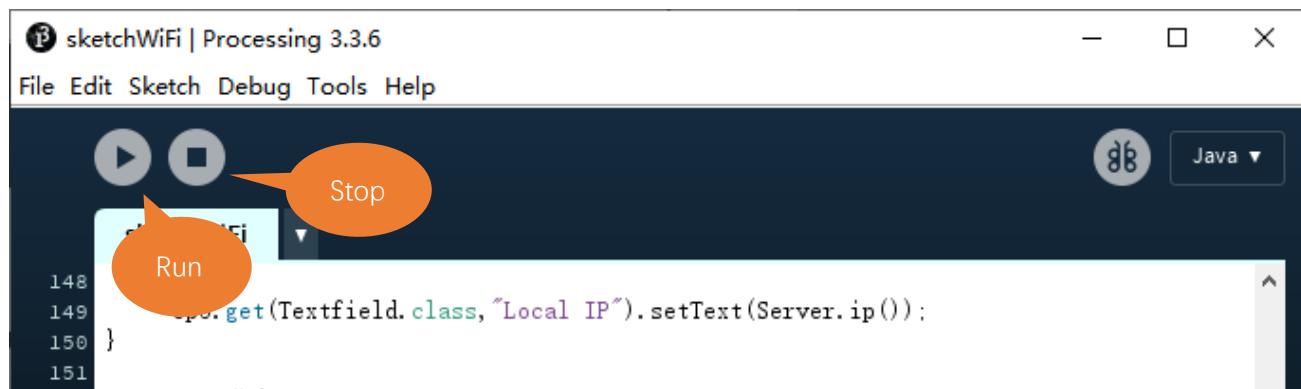
The screenshot shows the official Processing website. At the top, there's a navigation bar with links for 'Processing', 'p5.js', 'Processing.py', 'Processing for Android', 'Processing for Pi', and 'Processing Foundation'. Below the navigation bar is a large banner featuring the word 'Processing' in a stylized font over a background of abstract geometric shapes. To the right of the banner is a search bar with a magnifying glass icon. On the left side of the main content area, there's a sidebar with links: 'Cover', 'Download', 'Donate', 'Exhibition', 'Reference', 'Libraries', 'Tools', 'Environment', 'Tutorials', 'Examples', 'Books', 'Overview', and 'People'. In the center, under the heading 'Download Processing', it says 'Processing is available for Linux, Mac OS X, and Windows. Select your choice to download the software below.' Below this, it shows '3.5.4 (17 January 2020)' and download links for 'Windows 64-bit', 'Windows 32-bit', 'Linux 64-bit', and 'Mac OS X'. To the left of the download links is a large circular logo with a stylized 'P' inside. At the bottom of the sidebar, there are links for '» Github', '» Report Bugs', '» Wiki', '» Supported Platforms', and a note about 'Read about the changes in 3.0. The list of revisions covers the differences between releases in detail.'

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

core	2020/1/17 12:16
java	2020/1/17 12:17
lib	2020/1/17 12:16
modes	2020/1/17 12:16
tools	2020/1/17 12:16
processing.exe	2020/1/17 12:16
processing-java.exe	2020/1/17 12:16
revisions.txt	2020/1/17 12:16

Use Server mode for communication

Open the “Freenove\_Basic\_Starter\_Kit\_for\_Raspberry\_Pi\_Pico\Sketches\Sketch\_13.1\_WiFiClient\sketchWiFi\sketchWiFi.pde”, and click “Run”.

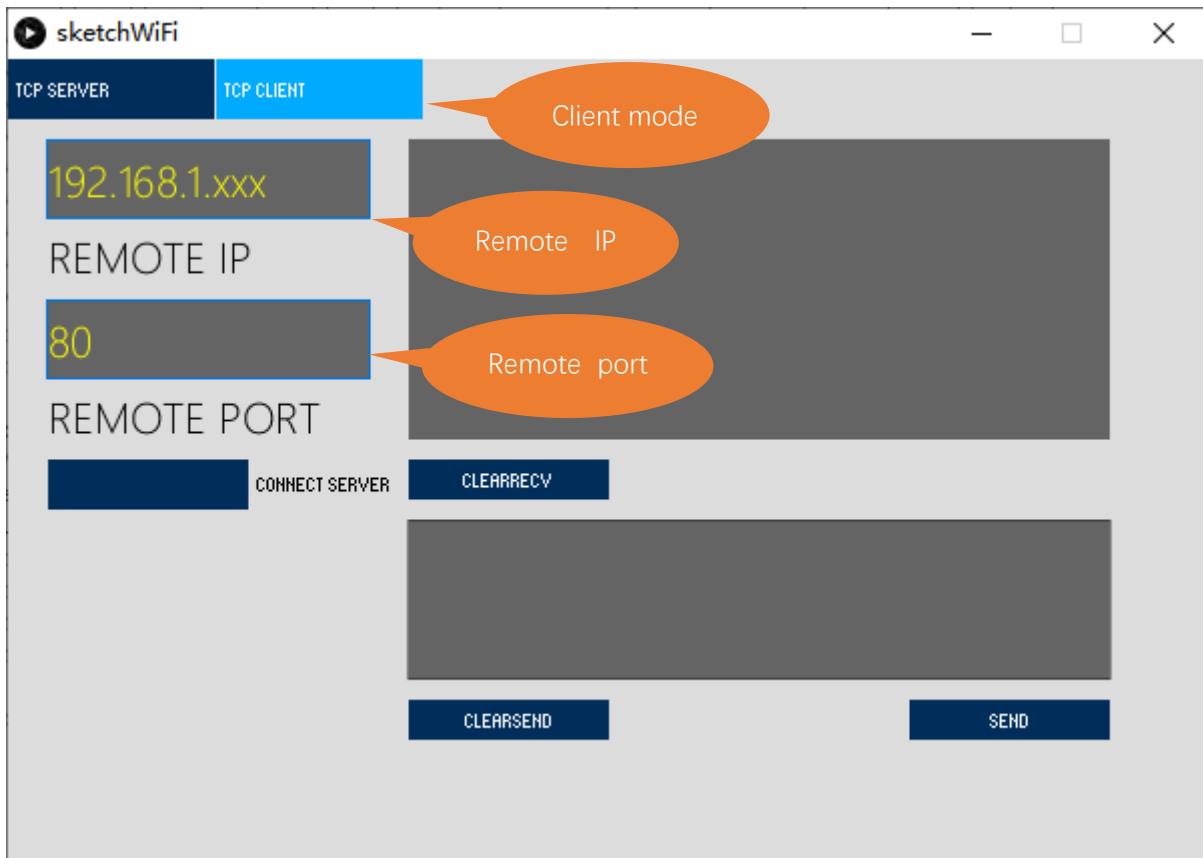


The new pop-up interface is as follows. If PICO W is used as client, select TCP SERVER mode for sketchWiFi.



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

If PICO W 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.

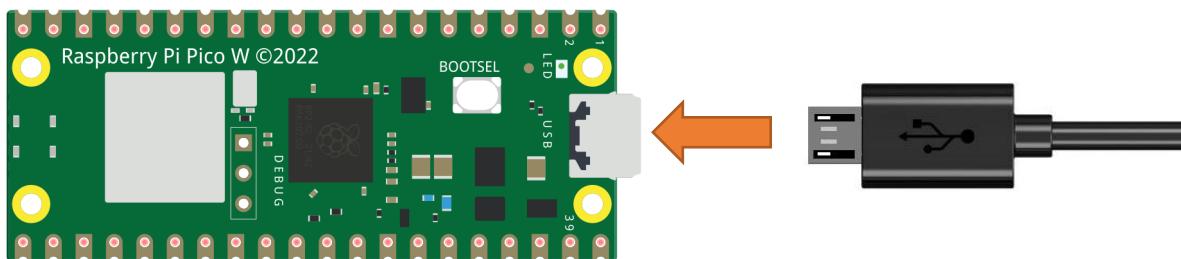
**clear receive:** clear out the content in the receiving text box

**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

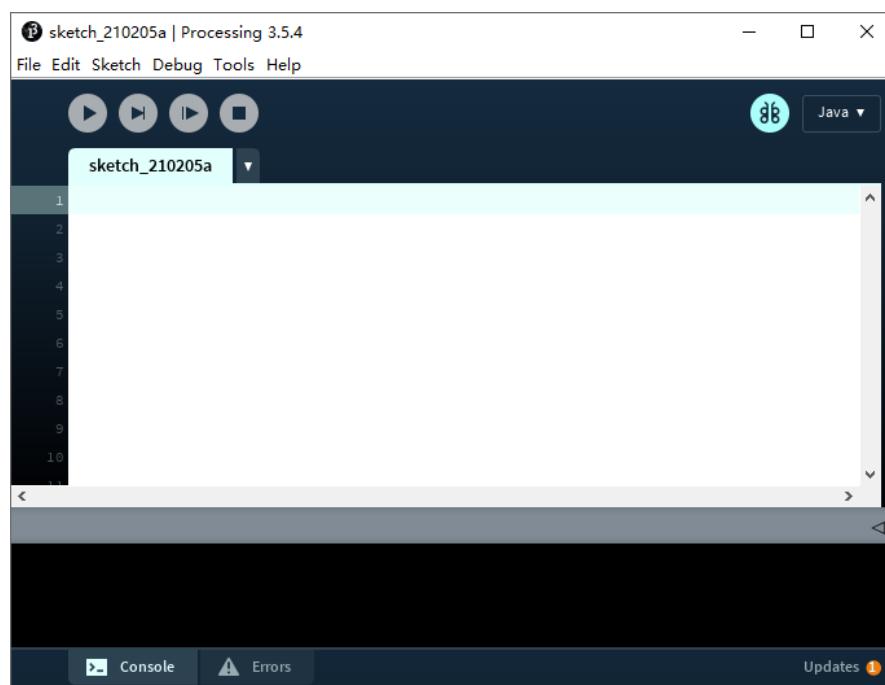
Connect Pico W to the computer using the USB cable.



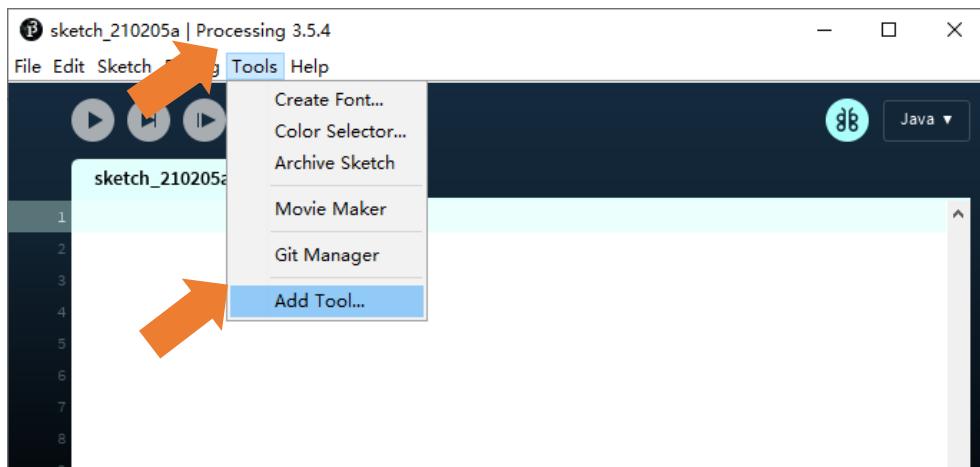
## Sketch

If you have not installed “ControlIP5”, please follow the following steps to continue the installation, if you have installed, please skip this section.

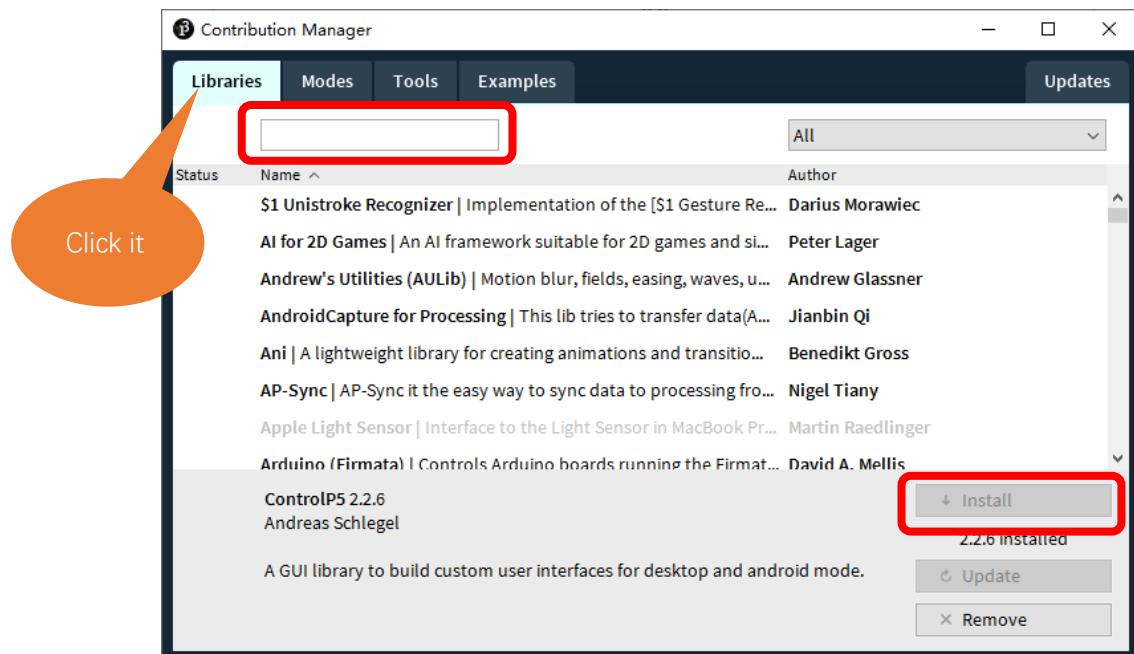
Open Processing.



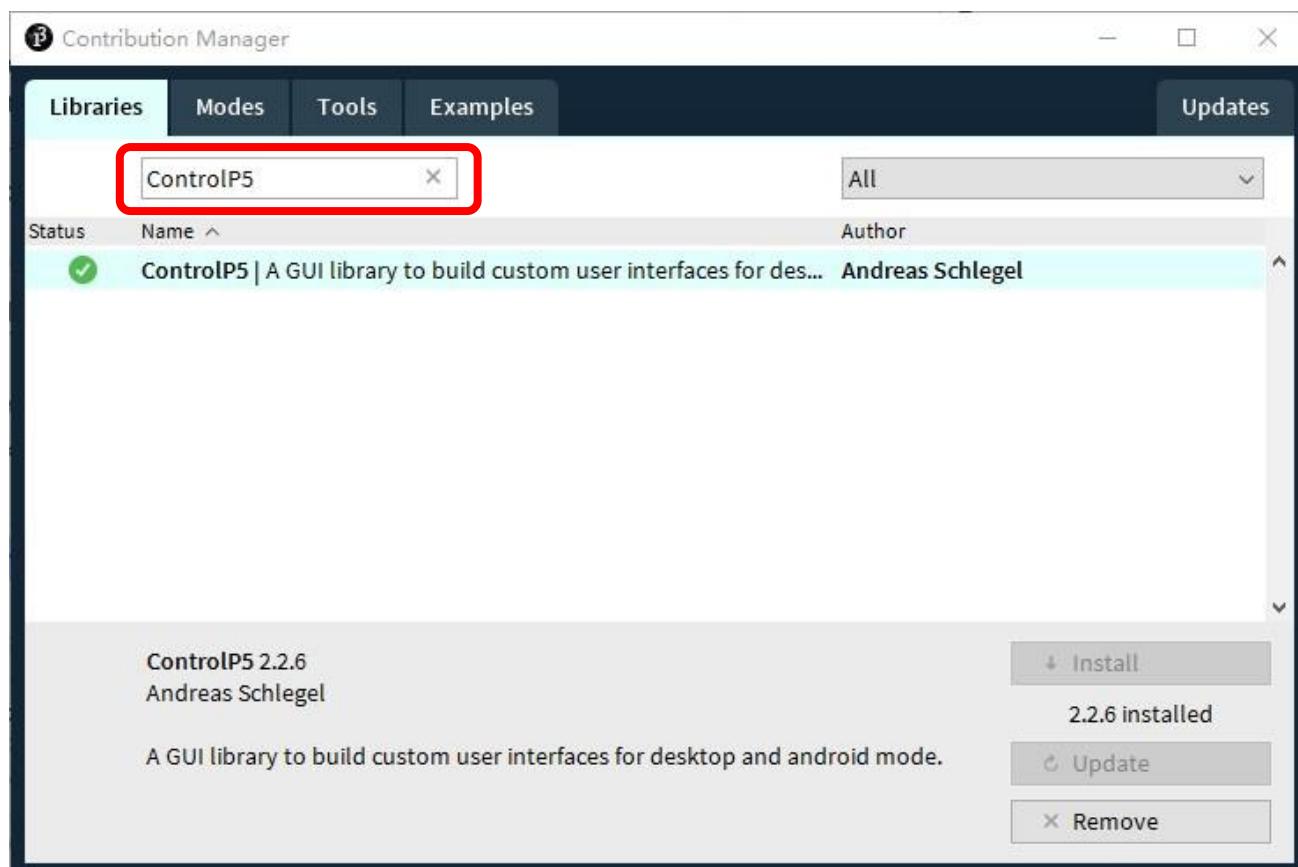
Click Add Tool under Tools.



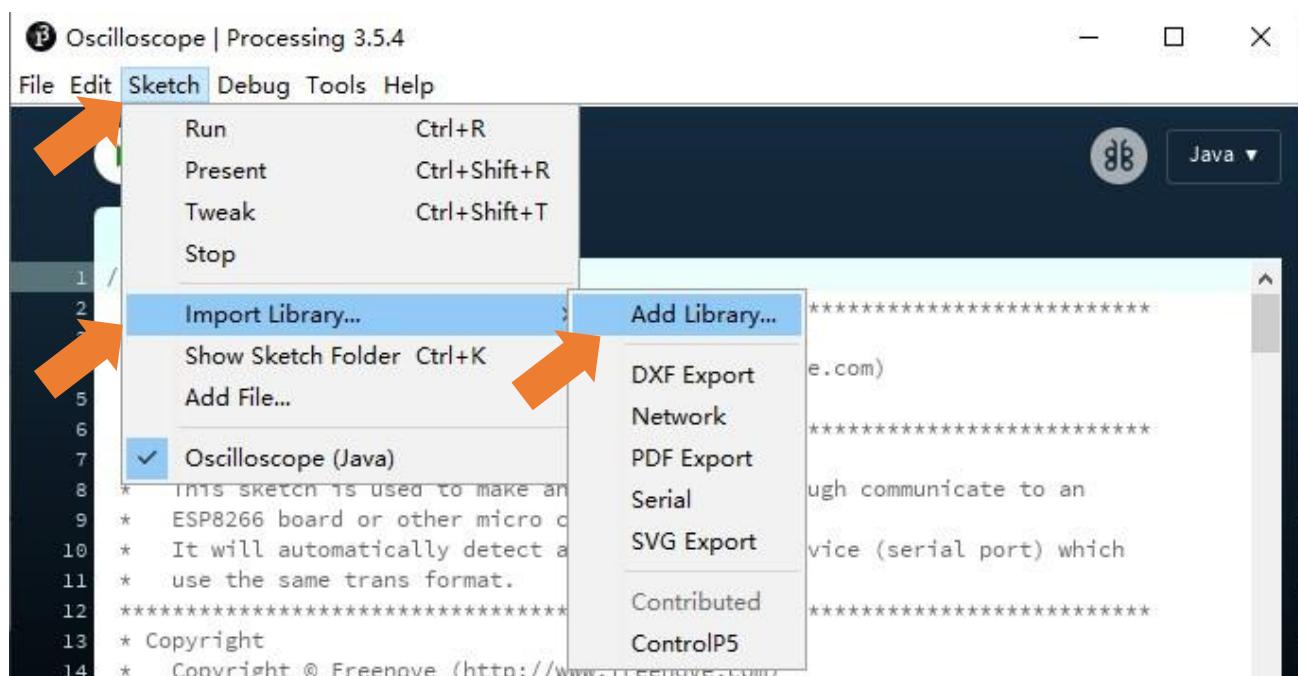
Select Libraries in the pop-up window.



Input "ControlP5" in the searching box, and then select the option as below. Click "Install" and wait for the installation to finish.

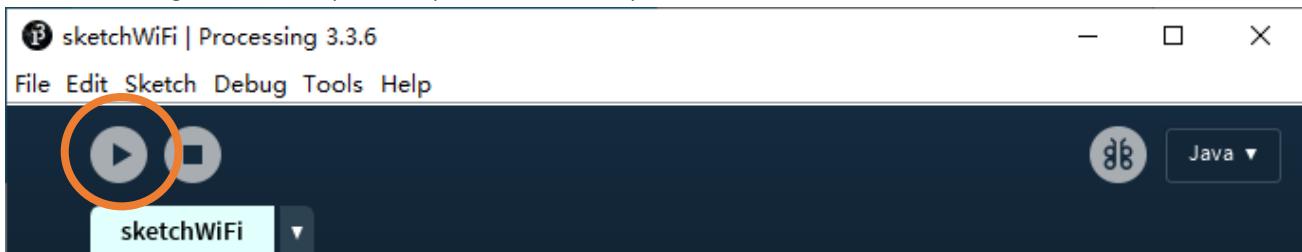


You can also click Add Library under 'Import Library' under 'Sketch'.

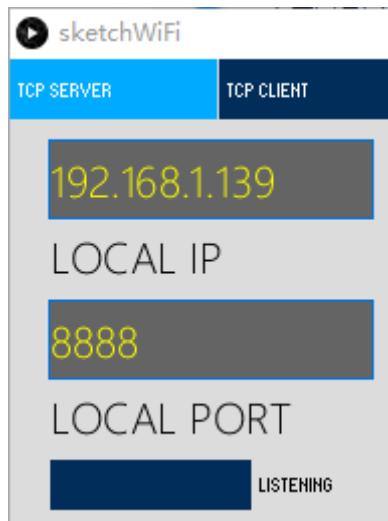


### Sketch\_13.1\_As\_Client

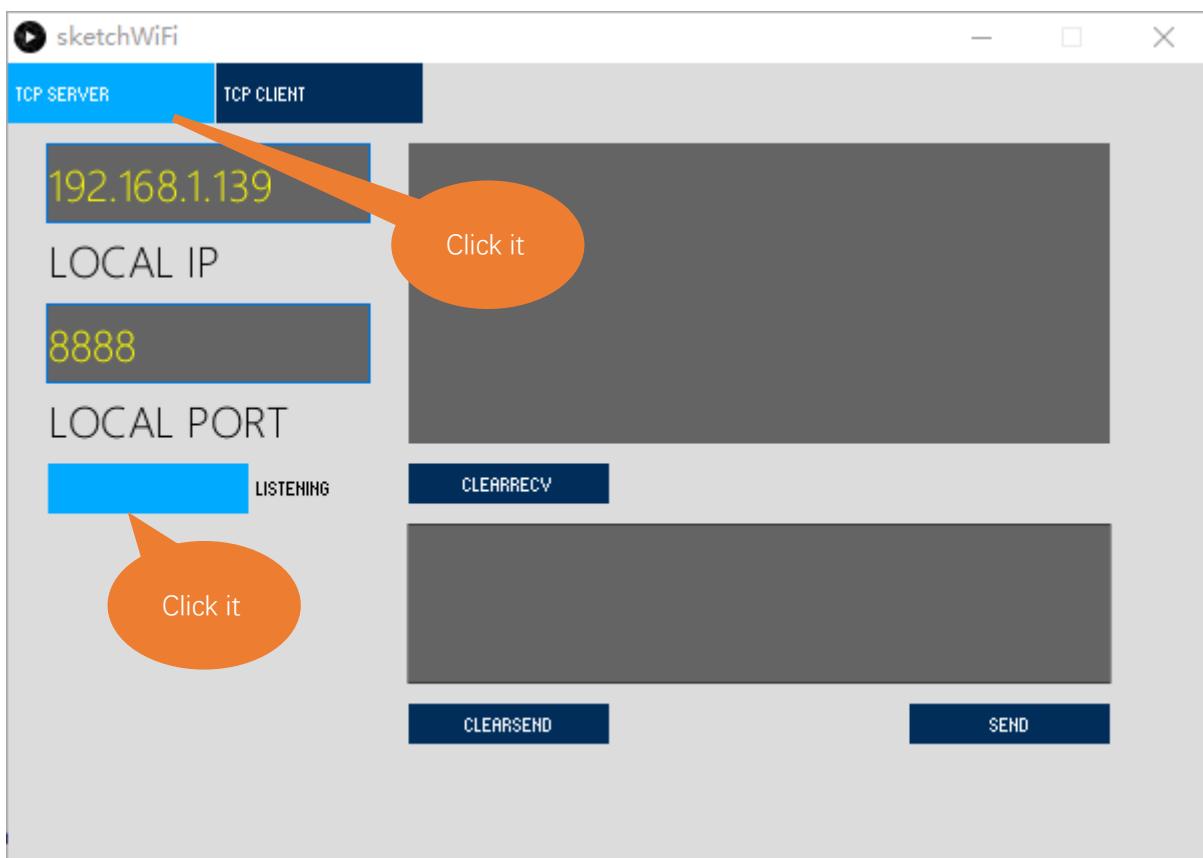
Before running the Sketch, please open “sketchWiFi.pde.” first, and click “Run”.



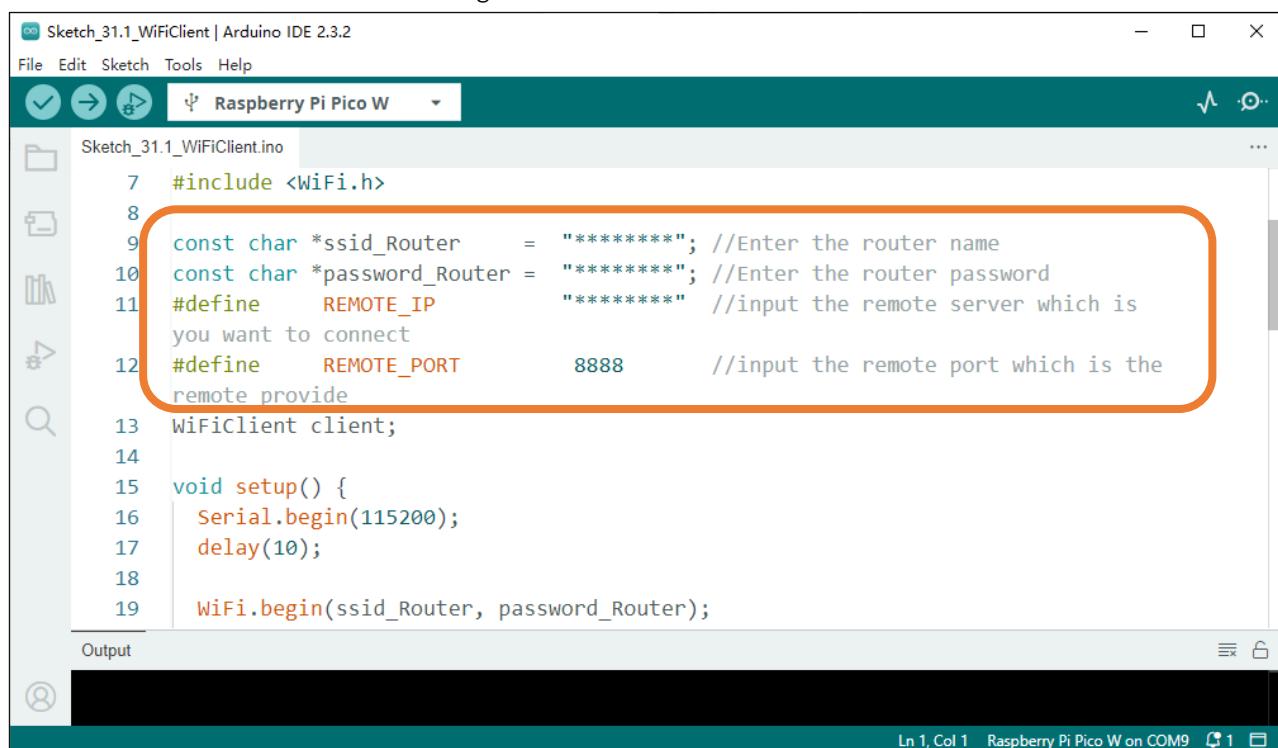
The newly pop up window will use the computer's IP address by default and open a data monitor port.



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



Next, open Sketch\_13.1\_WiFiClient.ino. Before running it, please change the following information based on "LOCAL IP" and "LOCAL PORT" in the figure above.



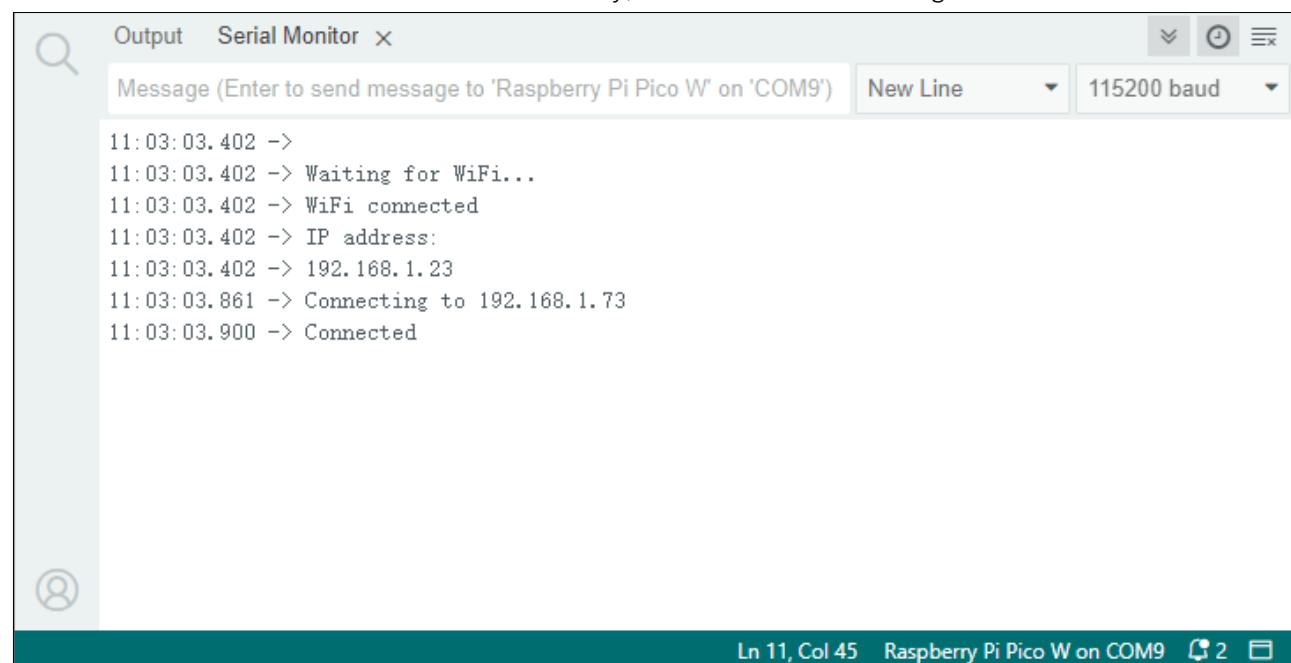
```

Sketch_13.1_WiFiClient | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Raspberry Pi Pico W
Sketch_31.1_WiFiClient.ino
7 #include <WiFi.h>
8
9 const char *ssid_Router      = "*****"; //Enter the router name
10 const char *password_Router = "*****"; //Enter the router password
11 #define    REMOTE_IP          "*****" //input the remote server which is
you want to connect
12 #define    REMOTE_PORT        8888     //input the remote port which is the
remote provide
13 WiFiClient client;
14
15 void setup() {
16   Serial.begin(115200);
17   delay(10);
18
19   WiFi.begin(ssid_Router, password_Router);

```

REMOTE\_IP needs to be filled in according to the interface of sketchWiFi.pde. Taking this tutorial as an example, its REMOTE\_IP is "192.168.1.73". Generally, by default, the ports do not need to change its value.

Compile and upload code to PICO W, open the serial monitor and set the baud rate to 115200. PICO W 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, PICO W can send messages to server.



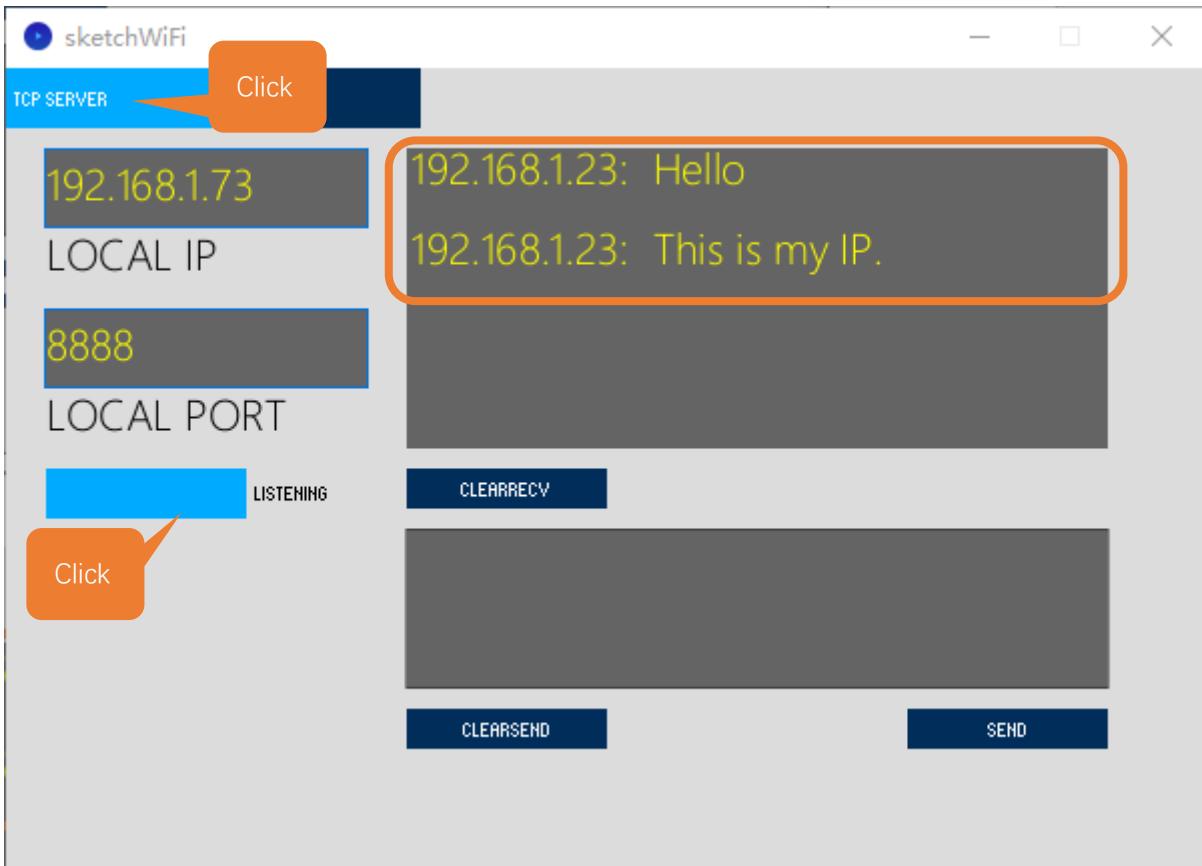
```

Output Serial Monitor
Message (Enter to send message to 'Raspberry Pi Pico W' on 'COM9') New Line 115200 baud
11:03:03.402 ->
11:03:03.402 -> Waiting for WiFi...
11:03:03.402 -> WiFi connected
11:03:03.402 -> IP address:
11:03:03.402 -> 192.168.1.23
11:03:03.861 -> Connecting to 192.168.1.73
11:03:03.900 -> Connected

```



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



At this point, you can send data to Pico W through sketchWiFi. Pico W will send the received data back to sketchWiFi after receiving it.

The following is the program code:

```
1 #include <WiFi.h>
2
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
7 WiFiClient client;
8
9 void setup() {
10   Serial.begin(115200);
11   delay(10);
12
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   }
19   Serial.println("");
20   Serial.println("WiFi connected");
21   Serial.println("IP address: ");
22   Serial.println(WiFi.localIP());
23   delay(500);
24
25   Serial.print("Connecting to ");
26   Serial.println(REMOTE_IP);
27
28   while (!client.connect(REMOTE_IP, REMOTE_PORT)) {
29     Serial.println("Connection failed.");
30     Serial.println("Waiting a moment before retrying... ");
31   }
32   Serial.println("Connected");
33   client.print("Hello\n");
34   client.print("This is my IP.\n");
35
36 void loop() {
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   }
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 disconnect the power supply and try again several times.

```

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 PICO W 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) {
```

**Any concerns? ✉ support@freenove.com**

```
44     delay(20);  
45     String line = Serial.readString();  
46     client.print(line);  
47 }
```

If the server is disconnected, turn off WiFi of PICO W.

```
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

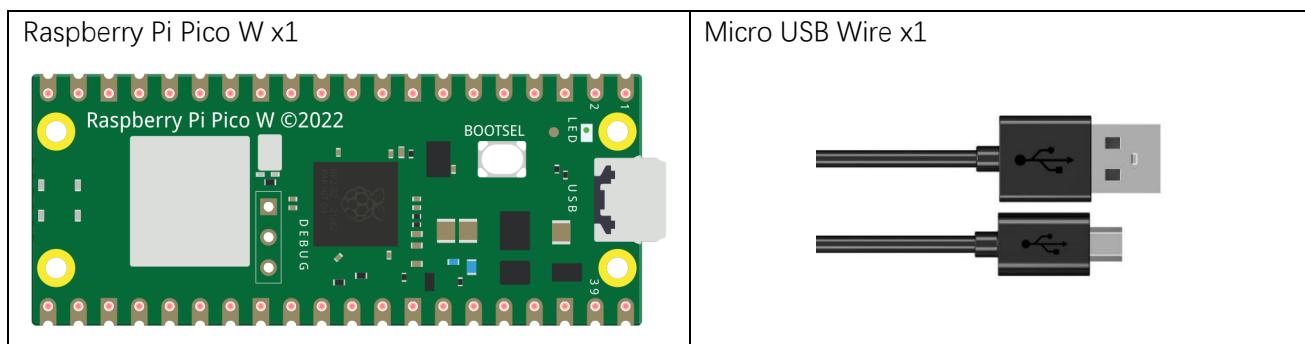
**readString()**: read string in receive buffer



## Project 13.2 as Server

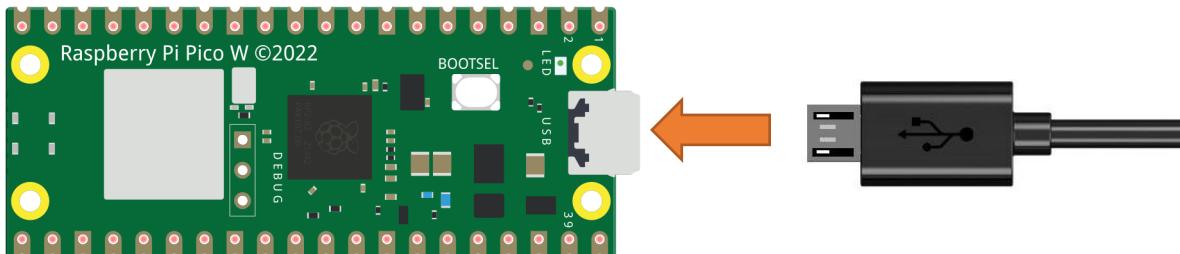
In this section, PICO W is used as a server to wait for the connection and communication of client on the same LAN.

### Component List



### Circuit

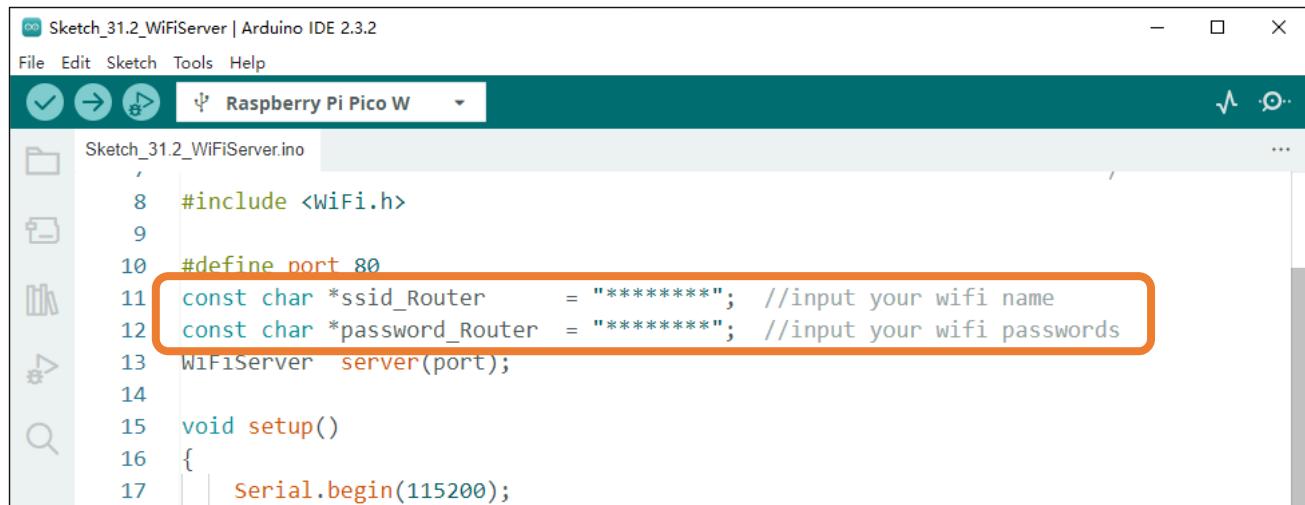
Connect Pico W to the computer using the USB cable.



## Sketch

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

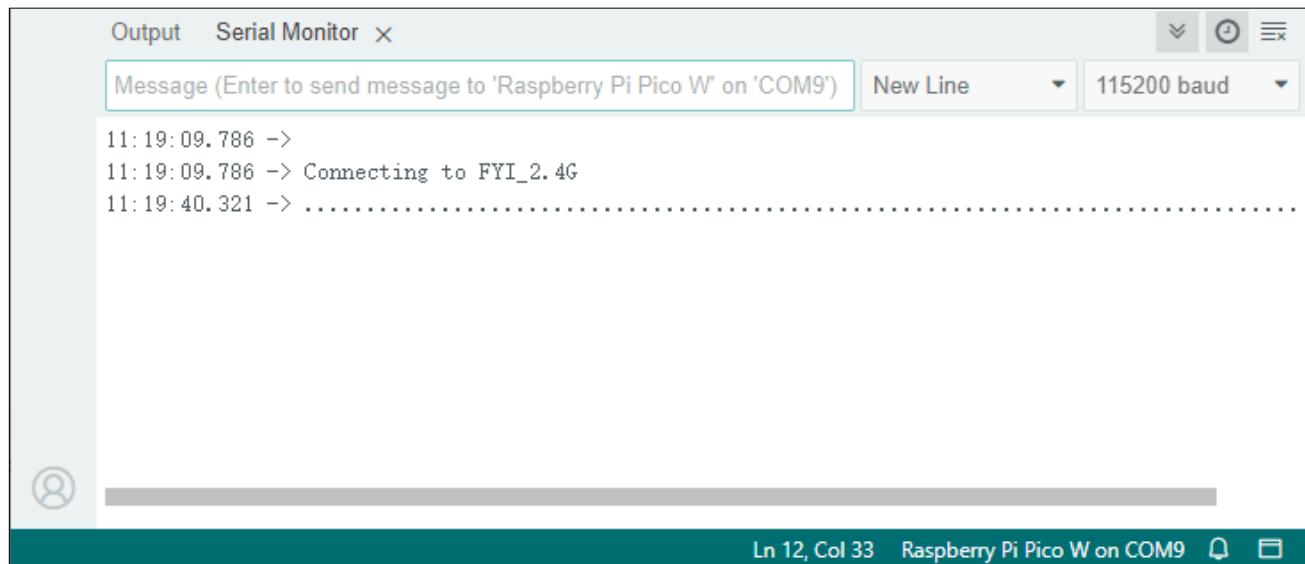
### Sketch\_13.2\_As\_Server



```
Sketch_31.2_WiFiServer | Arduino IDE 2.3.2
File Edit Sketch Tools Help
Sketch_31.2_WiFiServer.ino
1
2   #include <WiFi.h>
3
4
5 #define port 80
6 const char *ssid_Router      = "*****"; //input your wifi name
7 const char *password_Router = "*****"; //input your wifi passwords
8 WiFiServer server(port);
9
10 void setup()
11 {
12     Serial.begin(115200);
```

Compile and upload code to PICO W board, open the serial monitor and set the baud rate to 115200. Turn on server mode for PICO W, 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 Pico W fails to connect to router, please disconnect the power supply and try again several times.



```
Output Serial Monitor ×
Message (Enter to send message to 'Raspberry Pi Pico W' on 'COM9') New Line 115200 baud
11:19:09.786 ->
11:19:09.786 -> Connecting to FYI_2.4G
11:19:40.321 -> .....
```



## Serial Monitor

Output   Serial Monitor ×

Message (Enter to send message to 'Raspberry Pi Pico W' on 'COM9')   New Line   115200 baud

```

11:34:47.195 ->
11:34:47.195 -> Connecting to FYI_2.4G
11:34:55.192 ->
11:34:55.192 -> WiFi connected.
11:34:55.192 -> IP address: 192.168.1.23
11:34:55.235 -> IP port: 80
11:35:14.535 -> Client connected.
11:35:47.831 -> nice to meet you

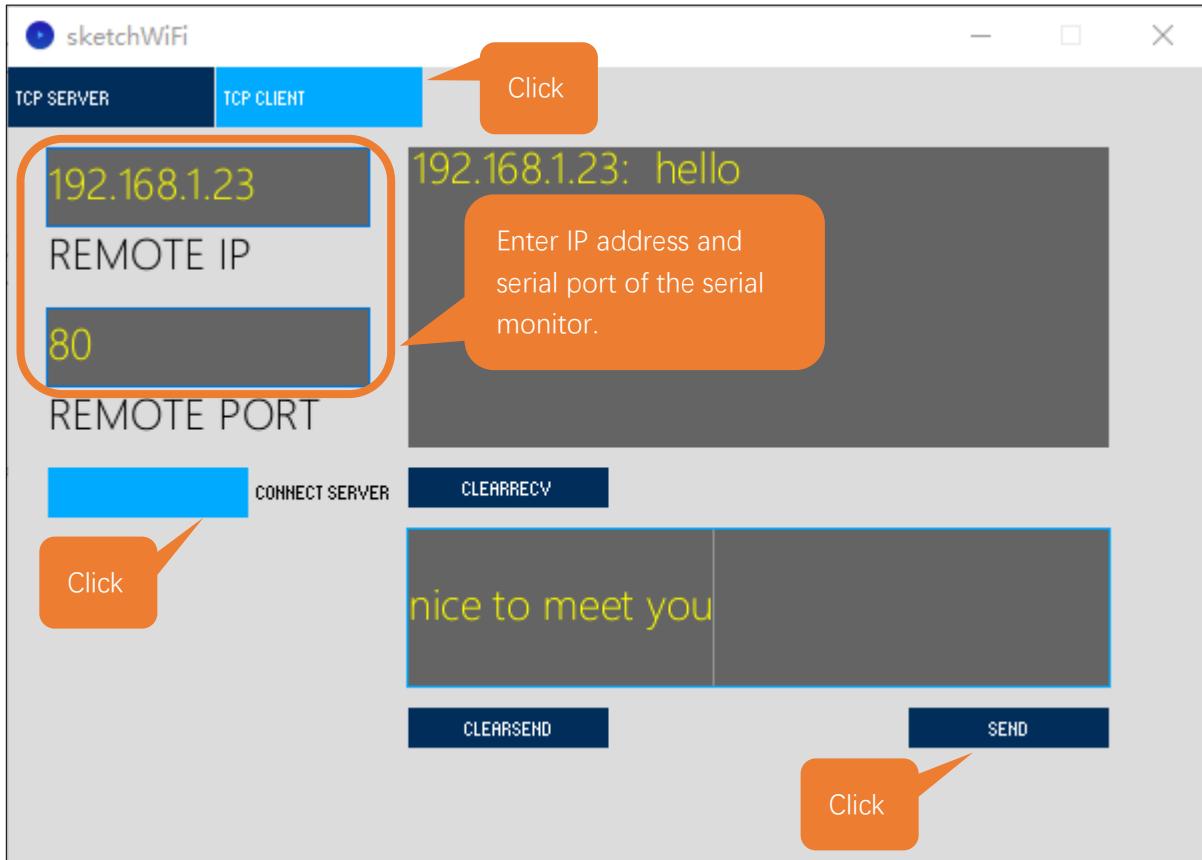
```

Ln 35, Col 1   Raspberry Pi Pico W on COM9   1  

Processing:

Open the “**Freenove\_Basic\_Starter\_Kit\_for\_Raspberry\_Pi\_Pico\C\Sketches\Sketch\_13.2\_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:

```
1 #include <WiFi.h>
2
3 #define port 80
4 const char *ssid_Router      = "*****"; //input your wifi name
5 const char *password_Router  = "*****"; //input your wifi passwords
6 WiFiServer server(port);
7
8 void setup()
9 {
10    Serial.begin(115200);
11    Serial.printf("\nConnecting to ");
12    Serial.println(ssid_Router);
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.");
22    Serial.print("IP address: ");
23    Serial.println(WiFi.localIP());
24    Serial.printf("IP port: %d\n", port);
25    server.begin(port);
26 }
27
28 void loop() {
29    WiFiClient client = server.available();           // listen for incoming clients
30    if (client) {                                     // if you get a client
31        Serial.println("Client connected.");
32        while (client.connected()) {                  // loop while the client's connected
33            if (client.available()) {                 // if there's bytes to read from the
34                Serial.println(client.readStringUntil('\n'));// print it out the serial monitor
35                while (client.read() > 0);               // clear the wifi receive area cache
36            }
37            if (Serial.available()) {                  // if there's bytes to read from the
38                client.print(Serial.readStringUntil('\n'));// print it out the client.
39                while (Serial.read() > 0);              // clear the wifi receive area cache
40            }
41        }
42    }
43 }
```

```

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

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 disconnect the power supply and try again several times.

```

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 PICO W.

```

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

Turn on server mode of PICO W.

```

25 server.begin();                         //Turn ON PICO W as Server mode
```

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

```

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

## 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**: port 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.



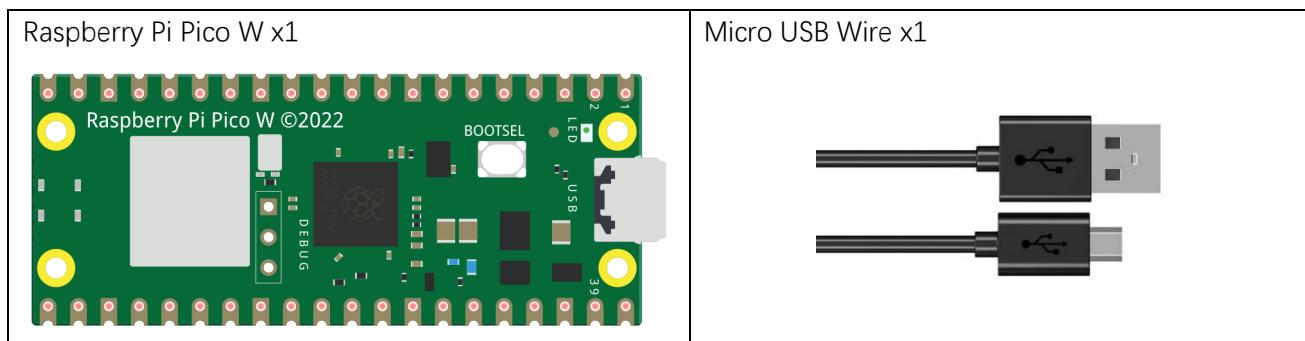
# Chapter 14 Control LED with Web (Only for Pico W)

In this chapter, we will use PICO W to make a simple smart home. We will learn how to control LED lights through web pages.

## Project 14.1 Control the LED with Web

In this project, we need to build a Web Service and then use PICO W to control the LED through the Web browser of the phone or PC. Through this example, you can remotely control the appliances in your home to achieve smart home.

### Component List



## Component knowledge

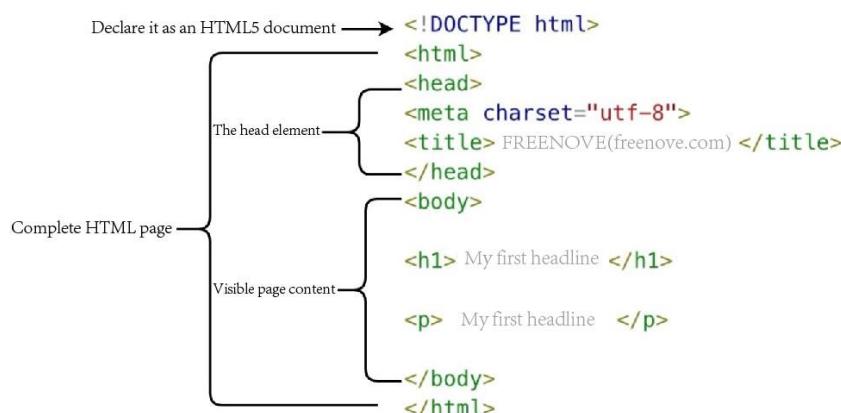
### HTML

Hypertext Markup Language (HTML) is a standard Markup Language for creating web pages. It includes a set of tags that unify documents on the network and connect disparate Internet resources into a logical whole. HTML text is descriptive text composed of HTML commands that describe text, graphics, animations, sounds, tables, links, etc. The extension of the HTML file is HTM or HTML. Hypertext is a way to organize information. It uses hyperlinks to associate words and charts in Text with other information media. These related information media may be in the same Text, other files, or files located on a remote computer. This way of organizing information connects the information resources distributed in different places, which is convenient for people to search and retrieve information.

The nature of the Web is hypertext Markup Language (HTML), which can be combined with other Web technologies (e.g., scripting languages, common gateway interfaces, components, etc.) to create powerful Web pages. Thus, Hypertext Markup Language (HTML) is the foundation of World Wide Web (Web) programming, that is, the World Wide Web is based on hypertext. Hypertext Markup Language is called hypertext Markup language because the text contains so-called "hyperlink" points.

You can build your own WEB site using HTML, which runs on the browser and is parsed by the browser.

Example analysis is shown in the figure below:



**<!DOCTYPE html>**: Declare it as an HTML5 document

**<html>**: Is the root element of an HTML page

**<head>**: Contains meta data for the document, such as &lt; meta charset="utf-8"&gt; Define the web page encoding format to UTF-8.

**<title>**: Notes the title of the document

**<body>**: Contains visible page content

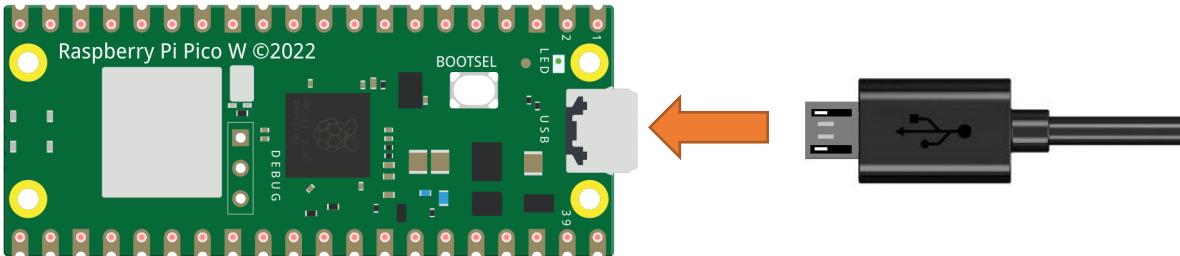
**<h1>**: Define a big heading

**<p>**: Define a paragraph

For more information, please visit: <https://developer.mozilla.org/en-US/docs/Web/HTML>

## Circuit

Connect Pico W to the computer using the USB cable.



## Sketch

### Sketch\_14.1\_Control\_the\_LED\_with\_Web

 A screenshot of the Arduino IDE interface. The title bar says "Sketch\_32.1\_Control\_the\_LED\_with\_Web | Arduino IDE 2.3.2". The menu bar includes File, Edit, Sketch, Tools, and Help. The central workspace shows the following C++ code for a sketch named "Sketch\_32.1\_Control\_the\_LED\_with\_Web.ino":
 

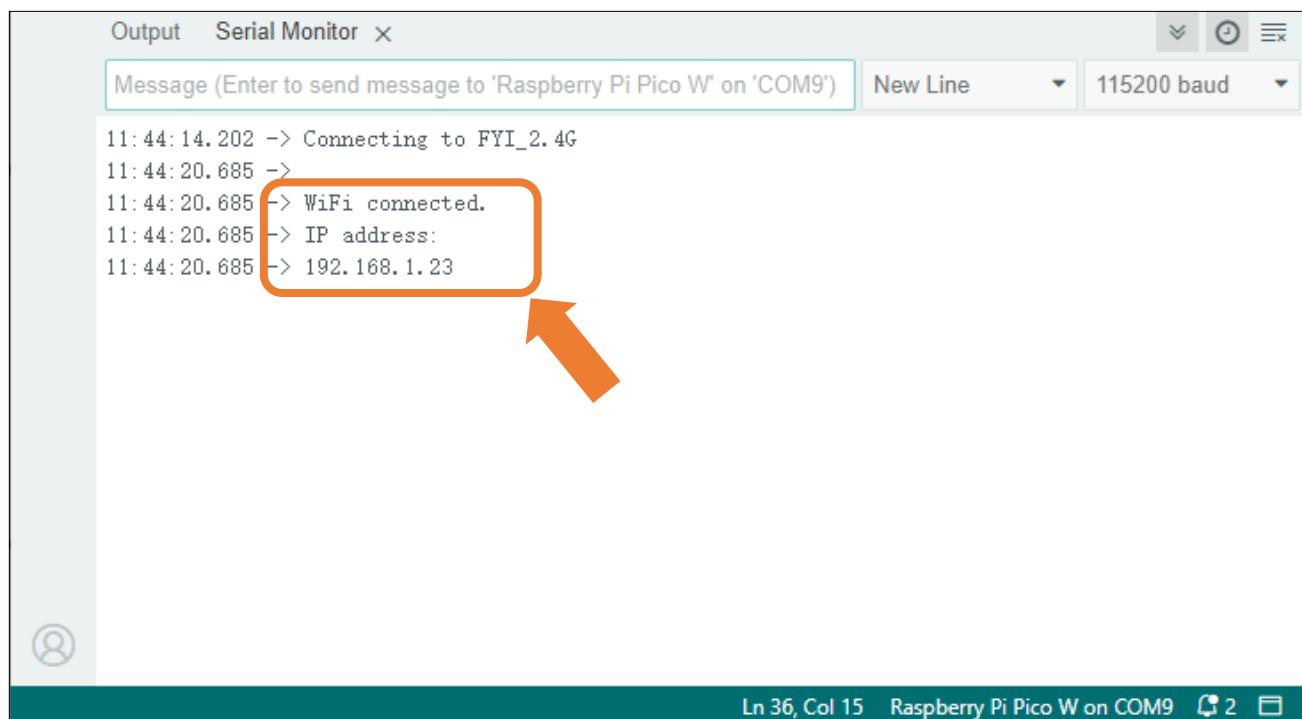
```

11 #include <WiFi.h>
12
13 // Replace with your network credentials
14 const char* ssid      = "*****";
15 const char* password = "*****";
16
17 // Set web server port number to 80
18 WiFiServer server(80);
19 // Variable to store the HTTP request
20 String header;
21 // Auxiliar variables to store the current output state
22 String PIN_LEDstate = "OFF";
23
24 // Current time
25 unsigned long currentTime = millis();
26 // Previous time
27 unsigned long previousTime = 0;
28 // Define timeout time in milliseconds (example: 2000ms = 2s)
29 const long timeoutTime = 2000;
30
31 void setup() {
32   Serial.begin(115200);
33   // Initialize the output variables as outputs
34   pinMode(LED_BUILTIN, OUTPUT);
35   digitalWrite(LED_BUILTIN, LOW);
36
37   // Connect to Wi-Fi network with SSID and password
38   Serial.print("Connecting to ");
  
```

 A callout bubble with an orange border and rounded corners points from the text "Enter the correct Router name and password." to the two lines of code: "const char\* ssid = \"\*\*\*\*\*\";" and "const char\* password = \"\*\*\*\*\*\";". The text inside the bubble is "Enter the correct Router name and password.".
 

Enter the correct Router name and password.

Download the code to PICO W, open the serial port monitor, set the baud rate to 115200, and you can use it to measure the distance between the ultrasonic module and the object. As shown in the following figure:



The screenshot shows the 'Serial Monitor' window with the title 'Output' and 'Serial Monitor'. The message area contains the following text:

```
11:44:14.202 -> Connecting to FYI_2.4G  
11:44:20.685 ->  
11:44:20.685 -> WiFi connected.  
11:44:20.685 -> IP address:  
11:44:20.685 -> 192.168.1.23
```

An orange arrow points from the text 'As shown in the following figure:' to the line '11:44:20.685 -> WiFi connected.' and the line '11:44:20.685 -> IP address: 192.168.1.23'.

When PICO W successfully connects to "ssid\_Router", serial monitor will print out the IP address assigned to PICO W by the router. Access <http://192.168.1.23> in a computer browser on the LAN. [As shown in the following figure:](#)



You can click the corresponding button to control the LED on and off.



The following is the program code:

```
1 #include <WiFi.h>
2
3 // Replace with your network credentials
4 const char* ssid      = "*****";
5 const char* password = "*****";
6
7 // Set web server port number to 80
8 WiFiServer server(80);
9 // Variable to store the HTTP request
10 String header;
11 // Auxiliar variables to store the current output state
12 String PIN_LEDState = "OFF";
13
14 // Current time
15 unsigned long currentTime = millis();
16 // Previous time
17 unsigned long previousTime = 0;
18 // Define timeout time in milliseconds (example: 2000ms = 2s)
19 const long timeoutTime = 2000;
20
21 void setup() {
22     Serial.begin(115200);
23     // Initialize the output variables as outputs
24     pinMode(LED_BUILTIN, OUTPUT);
25     digitalWrite(LED_BUILTIN, LOW);
26
27     // Connect to Wi-Fi network with SSID and password
28     Serial.print("Connecting to ");
29     Serial.println(ssid);
30     WiFi.begin(ssid, password);
31     while (WiFi.status() != WL_CONNECTED) {
32         delay(500);
33         Serial.print(".");
34     }
35     // Print local IP address and start web server
36     Serial.println("");
37     Serial.println("WiFi connected.");
38     Serial.println("IP address: ");
39     Serial.println(WiFi.localIP());
40     server.begin();
41 }
42 void loop() {
43     WiFiClient client = server.available(); // Listen for incoming clients
```

Any concerns? ✉ support@freenove.com

```
44  if (client) {                                // If a new client connects,
45      Serial.println("New Client.");           // print a message out in the serial port
46      String currentLine = "";                 // make a String to hold incoming data from the
47      client
48      currentTime = millis();
49      previousTime = currentTime;
50      while (client.connected() && currentTime - previousTime <= timeoutTime) { // loop while
the client's connected
51          currentTime = millis();
52          if (client.available()) { // if there's bytes to read from the client,
53              char c = client.read(); // read a byte, then
54              Serial.write(c);       // print it out the serial monitor
55              header += c;
56              if (c == '\n') { // if the byte is a newline character
57                  // if the current line is blank, you got two newline characters in a row.
58                  // that's the end of the client HTTP request, so send a response:
59                  if (currentLine.length() == 0) {
60                      // HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)
61                      // and a content-type so the client knows what's coming, then a blank line:
62                      client.println("HTTP/1.1 200 OK");
63                      client.println("Content-type:text/html");
64                      client.println("Connection: close");
65                      client.println();
66                      // turns the GPIOs on and off
67                      if (header.indexOf("GET /LED_BUILTIN/ON") >= 0) {
68                          Serial.println("LED_BUILTIN ON");
69                          PIN_LEDState = "ON";
70                          digitalWrite(LED_BUILTIN, HIGH);
71                      } else if (header.indexOf("GET /LED_BUILTIN/OFF") >= 0) {
72                          Serial.println("LED_BUILTIN OFF");
73                          PIN_LEDState = "OFF";
74                          digitalWrite(LED_BUILTIN, LOW);
75                      }
76                      // Display the HTML web page
77                      client.println("<!DOCTYPE html><html>");
78                      client.println("<head> <title>Pico W Web Server</title> <meta name=\"viewport\""
content="width=device-width, initial-scale=1">");
79                      client.println("<link rel=\"icon\" href=\"data:, \">");
80                      // CSS to style the on/off buttons
81                      // Feel free to change the background-color and font-size attributes to fit your
preferences
82                      client.println("<style>html {font-family: Helvetica; display:inline-block; margin:
0px auto; text-align: center;}</style>");
```

```

83         client.println(".button{background-color: #4286f4; display: inline-block; border:
84             none; border-radius: 4px; color: white; padding: 16px 40px;text-decoration: none; font-size:
85             30px; margin: 2px; cursor: pointer;}");
86         client.println(".button2{background-color: #4286f4;display: inline-block; border:
87             none; border-radius: 4px; color: white; padding: 16px 40px;text-decoration: none; font-size:
88             30px; margin: 2px; cursor: pointer;}</style></head>");
89         // Web Page Heading
90         client.println("<body><h1>Pico W Web Server</h1>");
91         client.println("<p>GPIO state: " + PIN_LEDState + "</p>"); 
92         client.println("<p><a href=\"/LED_BUILTIN/ON\"><button class=\"button
button2\">ON</button></a></p>"); 
93         client.println("<p><a href=\"/LED_BUILTIN/OFF\"><button class=\"button
button2\">OFF</button></a></p>"); 
94         client.println("</body></html>"); 
95         // The HTTP response ends with another blank line
96         client.println(); 
97         // Break out of the while loop
98         break;
99     } else { // if you got a newline, then clear currentLine
100        currentLine = "";
101    }
102  }
103  // Clear the header variable
104  header = "";
105  // Close the connection
106  client.stop();
107  Serial.println("Client disconnected.");
108  Serial.println("");
109 }
110 }
```

Include the WiFi Library header file of PICO W.

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

Enter correct router name and password.

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

Set PICO W in Station mode and connect it to your router.

```
30 WiFi.begin(ssid, password);
```

Check whether PICO W has connected to router successfully every 0.5s.

```
31 while (WiFi.status() != WL_CONNECTED) {
32     delay(500);
```

**Any concerns? ✉ support@freenove.com**

```
33     Serial.print(".");
34 }
```

Serial monitor prints out the IP address assigned to PICO W.

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

Click the button on the web page to control the LED light on and off.

```
65 // turns the GPIOs on and off
66 if (header.indexOf("GET /LED_BUILTIN/ON") >= 0) {
67     Serial.println("LED_BUILTIN ON");
68     PIN_LEDState = "ON";
69     digitalWrite(LED_BUILTIN, HIGH);
70 } else if (header.indexOf("GET /LED_BUILTIN/OFF") >= 0) {
71     Serial.println("LED_BUILTIN OFF");
72     PIN_LEDState = "OFF";
73     digitalWrite(LED_BUILTIN, LOW);
74 }
```



# Chapter 15 Bluetooth (Only for Pico W)

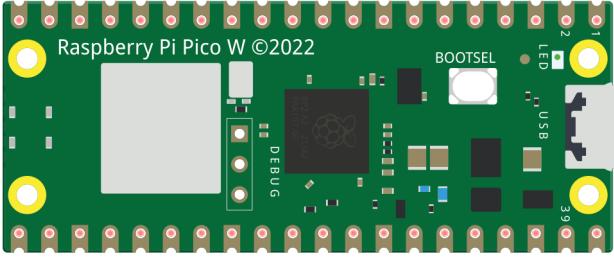
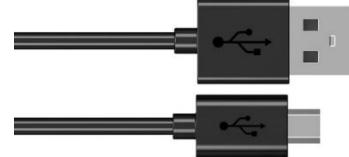
In June 2023, Raspberry Pi Official has updated to add Bluetooth support to the Pico W.

Pico W's Bluetooth 5.2 supports Bluetooth Classic and Bluetooth Low Energy (BLE) functionality. At the beginning of this chapter, we will learn the Pico W's Bluetooth function.

If you have Pico in your hand, please change it to Pico W before continuing to learn.

## Project 15.1 Bluetooth Passthrough

### Component List

Raspberry Pi Pico W x1	Micro USB Wire x1
	

### Component knowledge

Pico W's wireless functionality is provided by the Infineon CYW43439 device, which contains a 2.4 GHz radio providing both 802.11n Wi-Fi and Bluetooth 5.2, supporting Bluetooth Classic and Bluetooth Low Energy (BLE) functionality.

For simple data transfer, there are two modes:

#### Master mode

A device works in master mode can connect to one or more slave devices.

We can search and select the slave devices nearby to connect.

When a device initiates a connection request in master mode, it requires information about other Bluetooth devices, including their addresses and pairing keys.

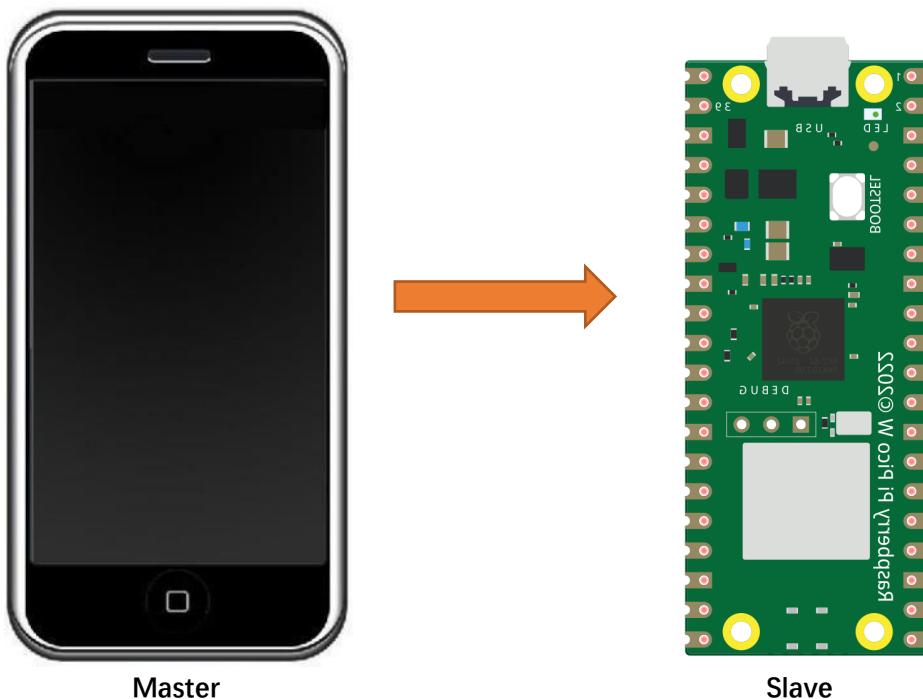
Once the devices are paired, a direct connection can be established.

#### Slave mode

A Bluetooth module operating in slave mode can only receive connection requests from a master device and cannot actively initiate connections.

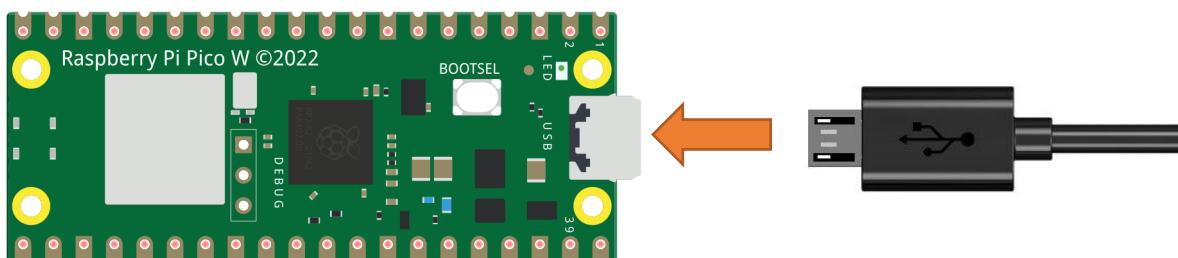
After establishing a connection with a master device, it can either send or receive data.

Bluetooth devices can interact with each other, with one in master mode and the other in slave mode. During data communication, the master device searches for and selects nearby devices to connect with. Once a connection is made, data can be exchanged between the devices. In the case of data exchange between a smartphone and a Raspberry Pi Pico W, the smartphone typically operates in master mode, while the Raspberry Pi Pico W functions in slave mode.



## Circuit

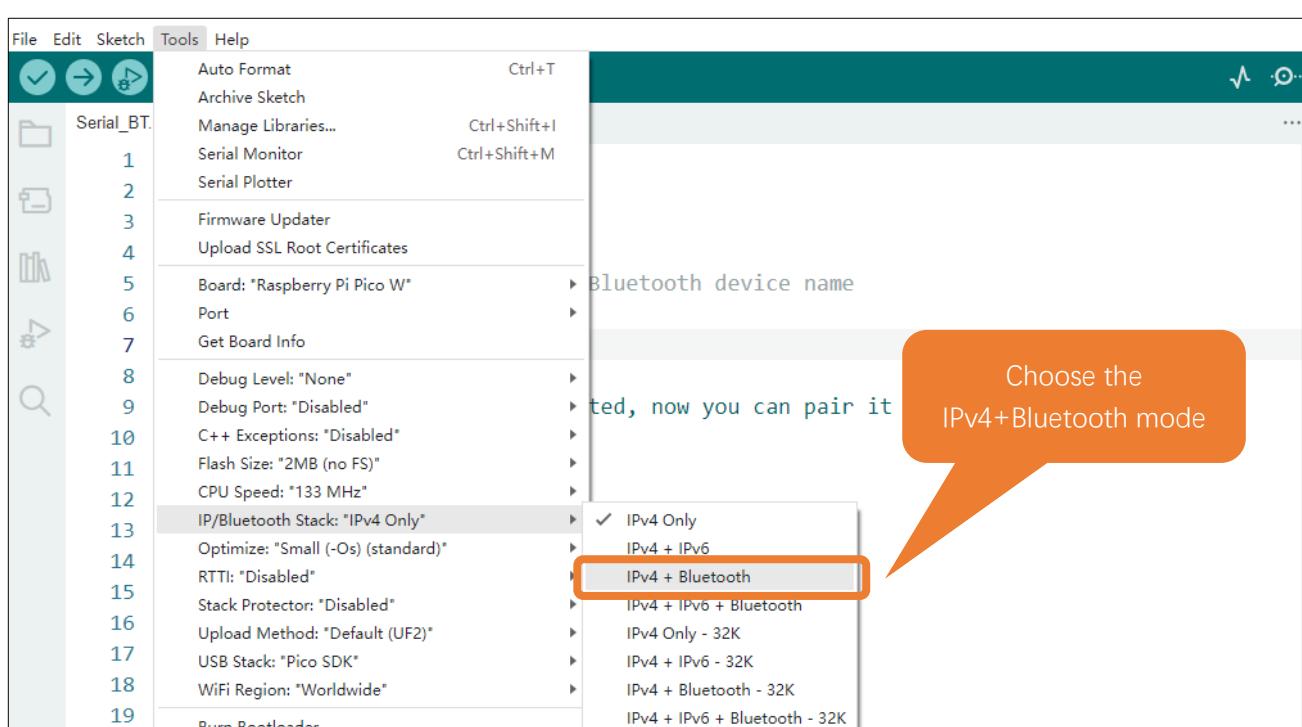
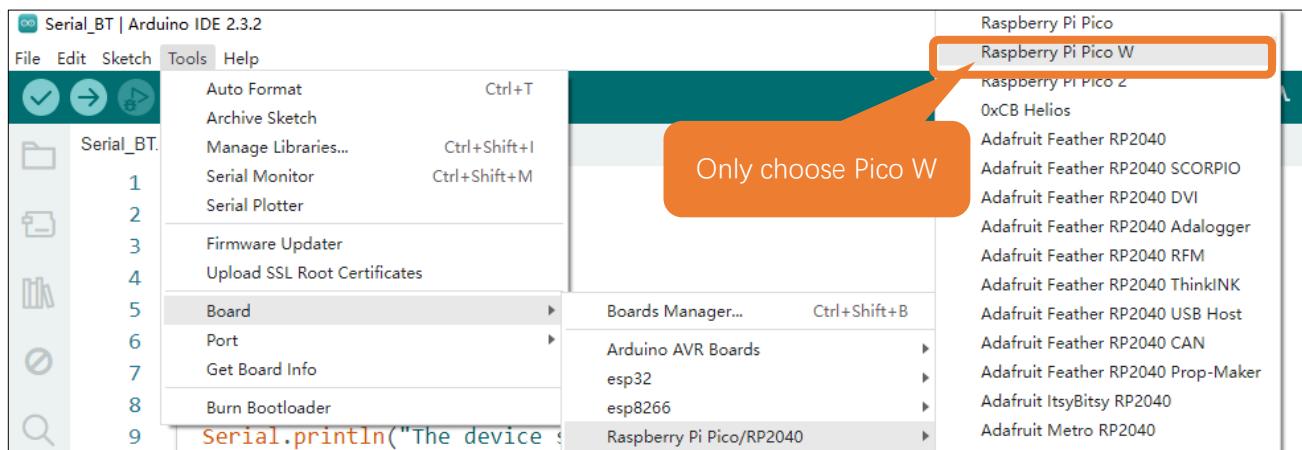
Connect Pico W to the computer using the USB cable.



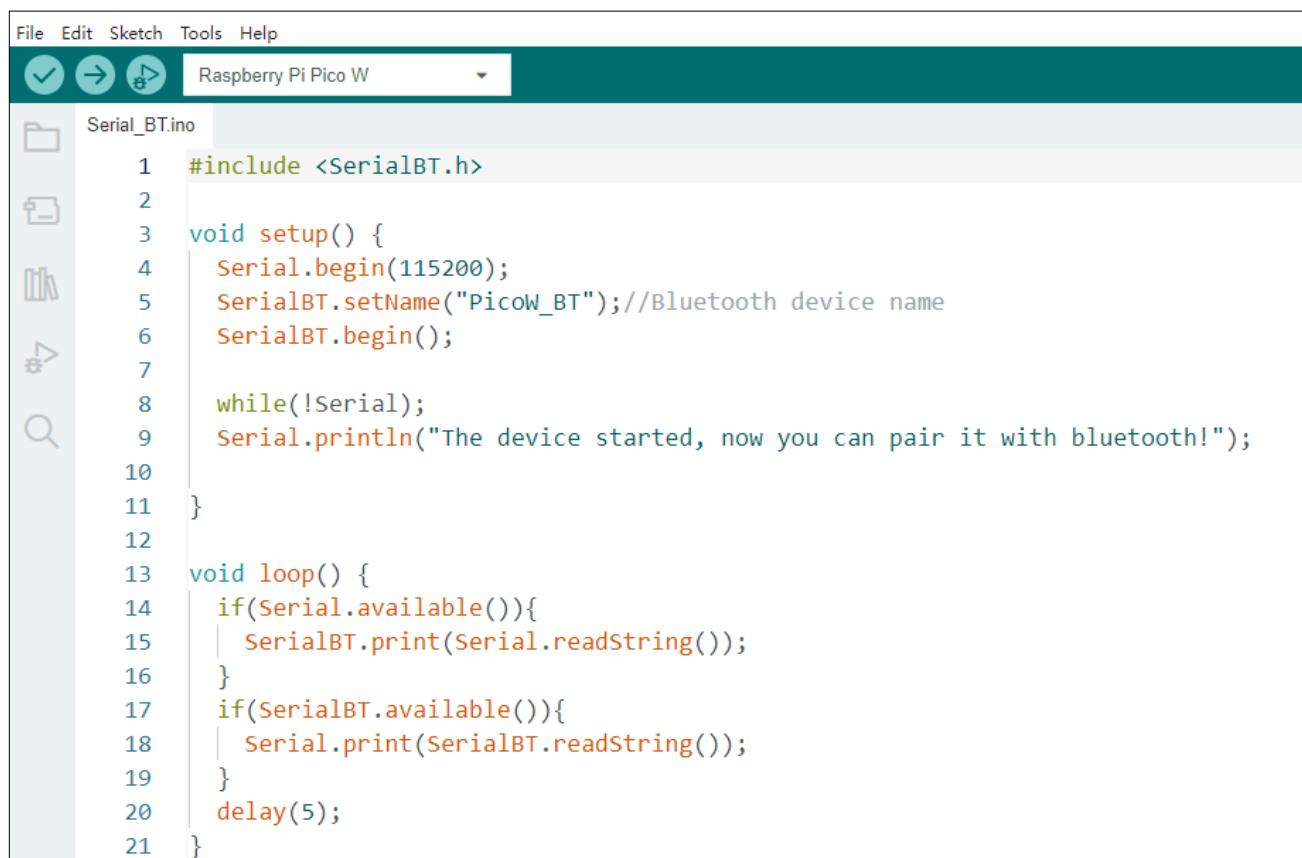
## Sketch

### How to enable Pico W's Bluetooth

In default setting, Pico W only enables the ipv4 function of its WiFi. To enable Bluetooth, please refer to the following steps:

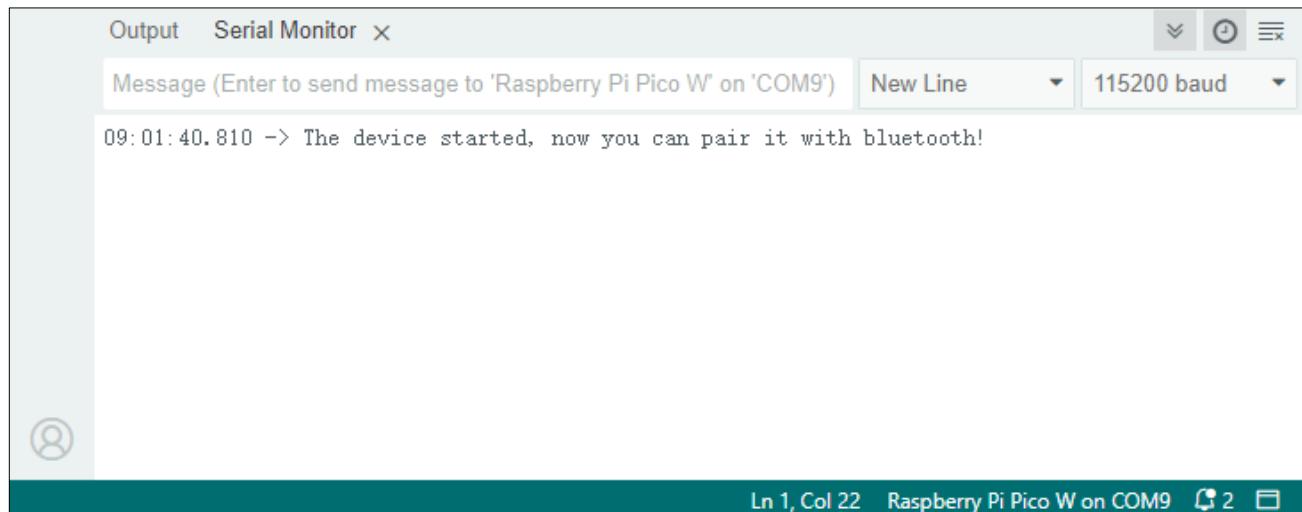


### Sketch\_15.1\_Serial\_BT



```
File Edit Sketch Tools Help
Serial_BT.ino
1 #include <SerialBT.h>
2
3 void setup() {
4     Serial.begin(115200);
5     SerialBT.setName("PicoW_BT"); //Bluetooth device name
6     SerialBT.begin();
7
8     while(!Serial);
9     Serial.println("The device started, now you can pair it with bluetooth!");
10 }
11
12 void loop() {
13     if(Serial.available()){
14         SerialBT.print(Serial.readString());
15     }
16     if(SerialBT.available()){
17         Serial.print(SerialBT.readString());
18     }
19     delay(5);
20 }
```

Upload the sketch to Pico W and open the serial monitor, set the baud rate to 115200. When you see the message as shown below, it indicates that, the Bluetooth of Pico W is ready.



Output Serial Monitor X

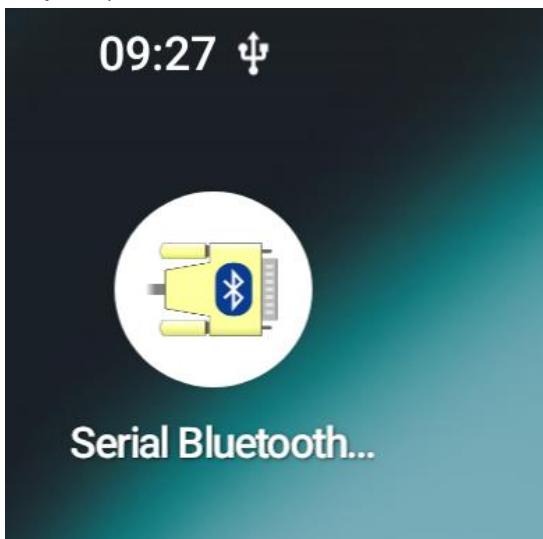
Message (Enter to send message to 'Raspberry Pi Pico W' on 'COM9') New Line 115200 baud

09:01:40.810 -> The device started, now you can pair it with bluetooth!

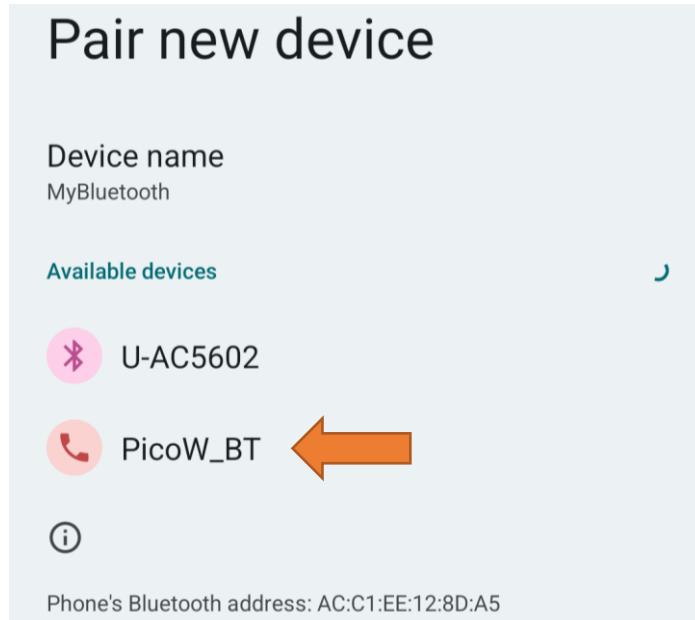
Ln 1, Col 22 Raspberry Pi Pico W on COM9 2



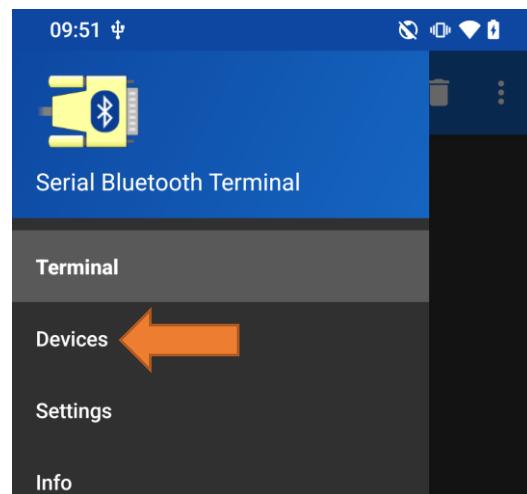
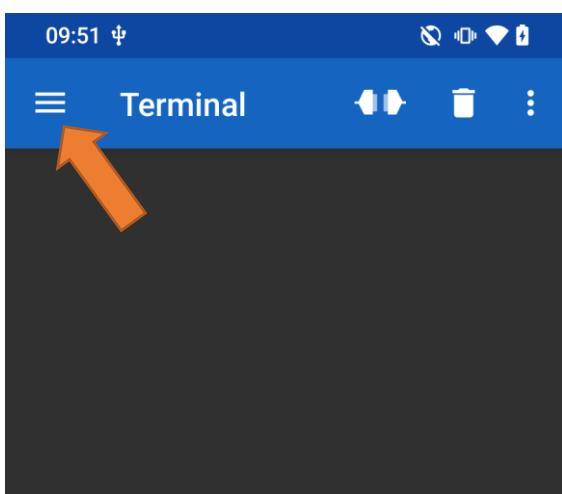
Please make sure the Bluetooth of your phone is turn ON and the App Serial Bluetooth Terminal has installed on your phone.



Click Pair new device and select "PicoW\_BT" to connect.

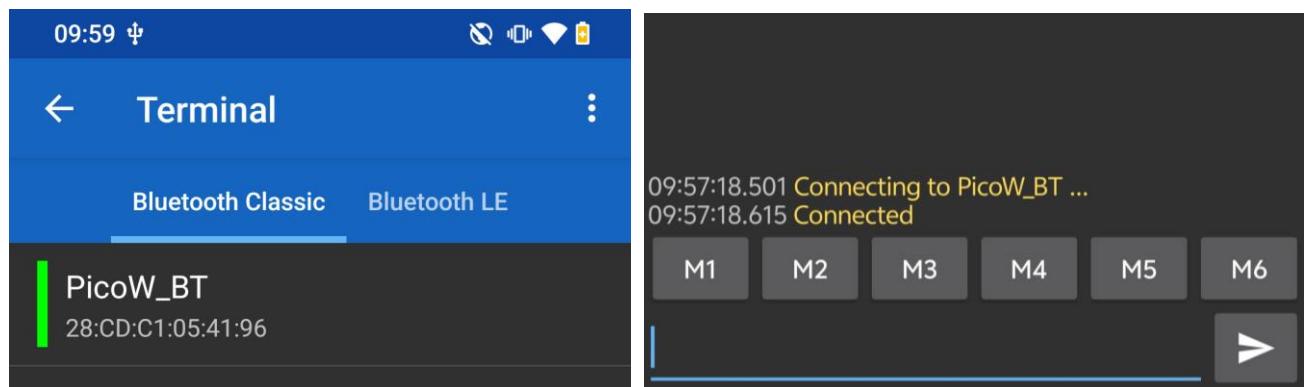


Open the App "Serial Bluetooth Terminal", expand the menu and select "Devices".



Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

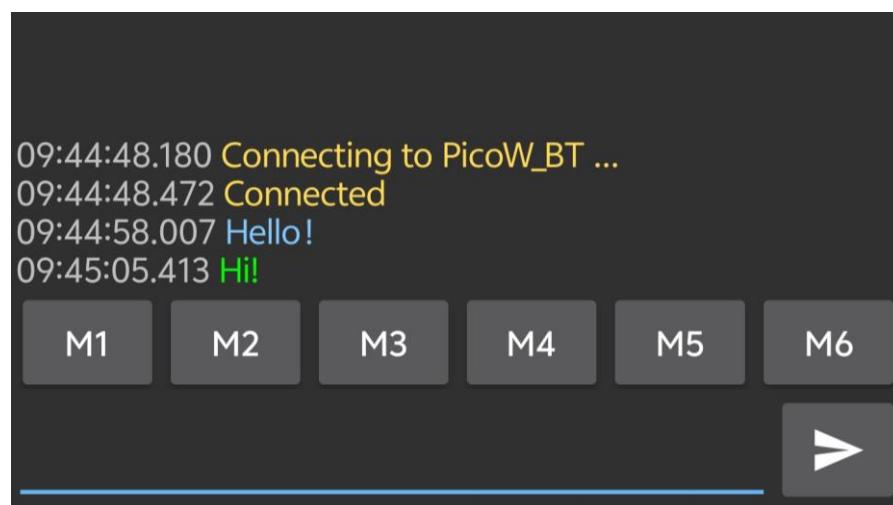
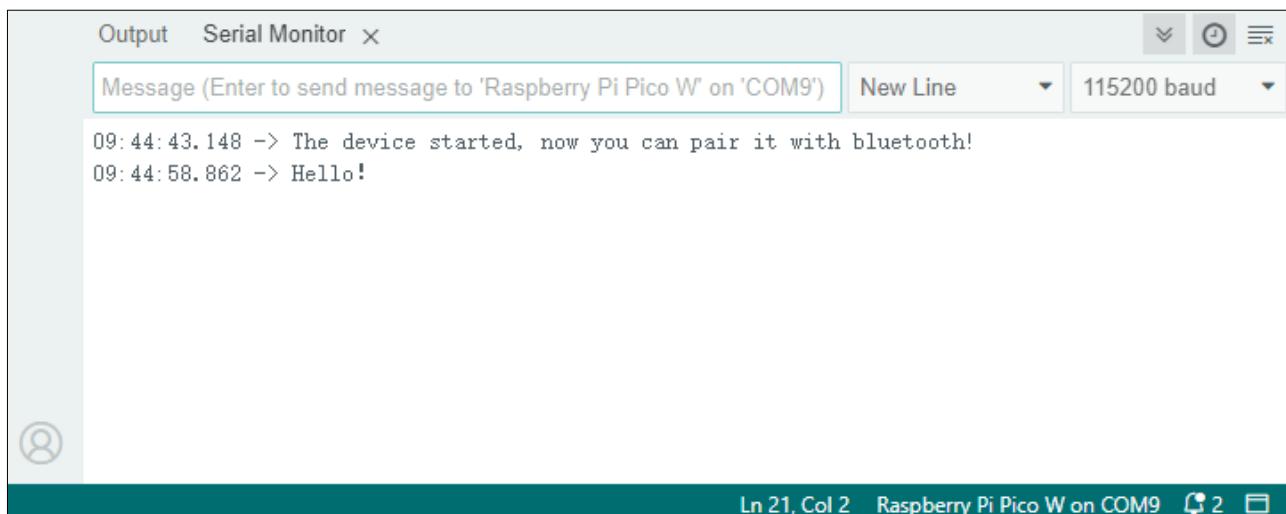
In Bluetooth Classic mode, select PicoW\_BT, and you will see "Connected", which indicates a successful connection.



Now, the data can be transferred between your phone and computer (Pico W).

Input Hello at the sending bar on the phone app and tap the send icon to send message to pico W.

When the computer receives it, input "Hi" at the Message bar and hit Enter key to send message to your phone.





The following is the program code:

```
1 #include <SerialBT.h>
2
3 void setup() {
4     Serial.begin(115200);
5     SerialBT.setName("PicoW_BT"); //Bluetooth device name
6     SerialBT.begin();
7     while(!Serial);
8     Serial.println("The device started, now you can pair it with bluetooth!")
9 }
10
11 void loop() {
12     if(Serial.available()) {
13         SerialBT.print(Serial.readString());
14     }
15     if(SerialBT.available()) {
16         Serial.print(SerialBT.readString());
17     }
18     delay(5);
19 }
20 }
```

## Reference

### Class SerialBT

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

**SetName(const char \*name)**: Sets the Bluetooth module name.

**begin()**: Initializes the Bluetooth functionality.

**available()**: Retrieves data sent from the buffer; if none, returns 0.

**read()**: Reads data from Bluetooth, returns data as an int type.

**readString()**: Reads data from Bluetooth, returns data as a String type.

**write(uint8\_t c)**: Sends a single uint8\_t type of data to Bluetooth.

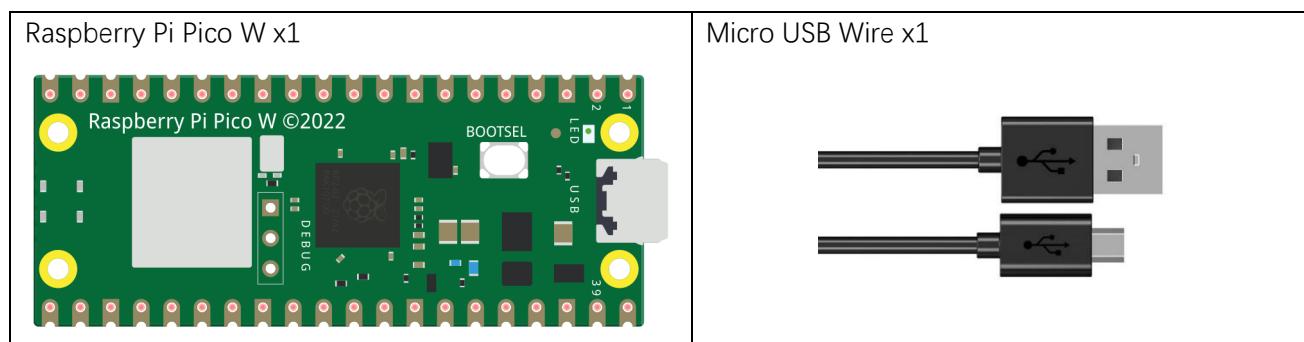
**write(const uint8\_t \*p, size\_t len)**: Sends the first len bytes of data stored at pointer address p to Bluetooth.

**print()**: Sends all types of data to Bluetooth for printing.

**end()**: Disconnects all Bluetooth devices and turns off Bluetooth, freeing up all occupied space.

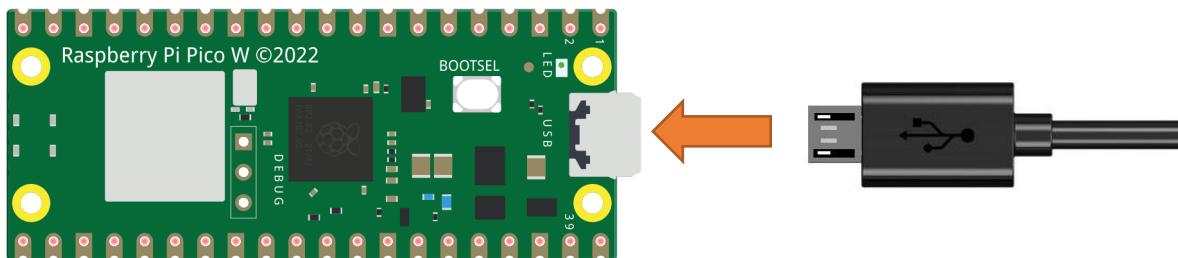
## Project 15.2 Bluetooth Low Energy Data Passthrough

### Component List

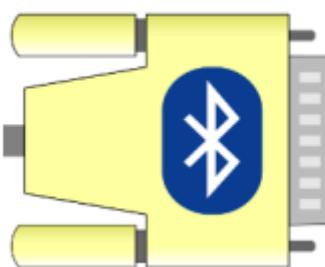


### Circuit

Connect Pico W to the computer using the USB cable.



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.





## Sketch

### Sketch\_15.2\_Serial\_BT

```

File Edit Sketch Tools Help
Serial_BLE.ino
42 void setup(void) {
43   Serial.begin(115200);
44   while(!Serial);
45
46   setupBLE("PicoW_BLE");
47 }
48
49 void loop(void) {
50   BTstack.loop();
51
52   if (Serial.available() > 0) {           // Check if there is serial
53     String input = Serial.readStringUntil('\n'); // Read until '\n'
54     input.trim();                         // Remove front and back blanks
55
56     size_t input_length = input.length();    // Get String length
57
58     // Copy input to characteristic_data and add line breaks
59     memcpy(characteristic_data, input.c_str(), input_length);
60     characteristic_data[input_length] = '\n'; // add linefeeds
61     characteristic_data[input_length + 1] = '\0'; // End of string
62
63     Serial.print("input data: ");
64     Serial.print(characteristic_data);

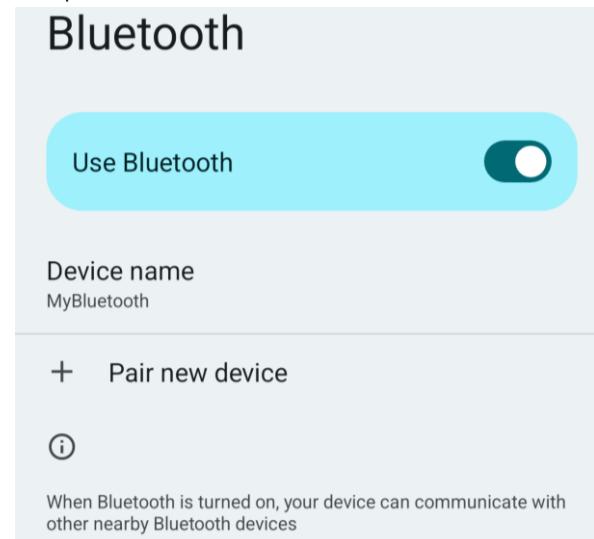
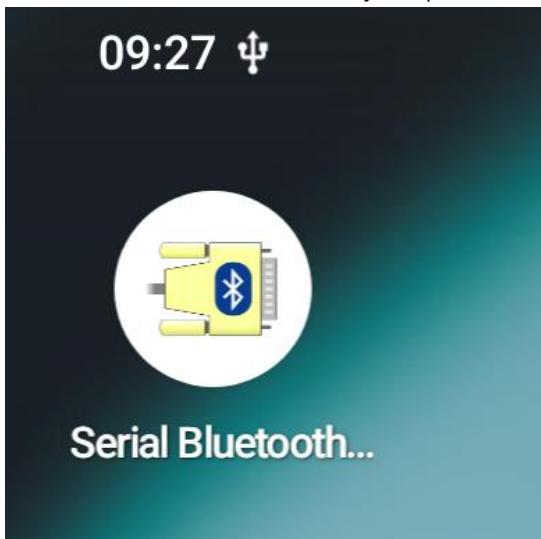
```

Before uploading the sketch, please make sure the Bluetooth of Pico W is enabled.

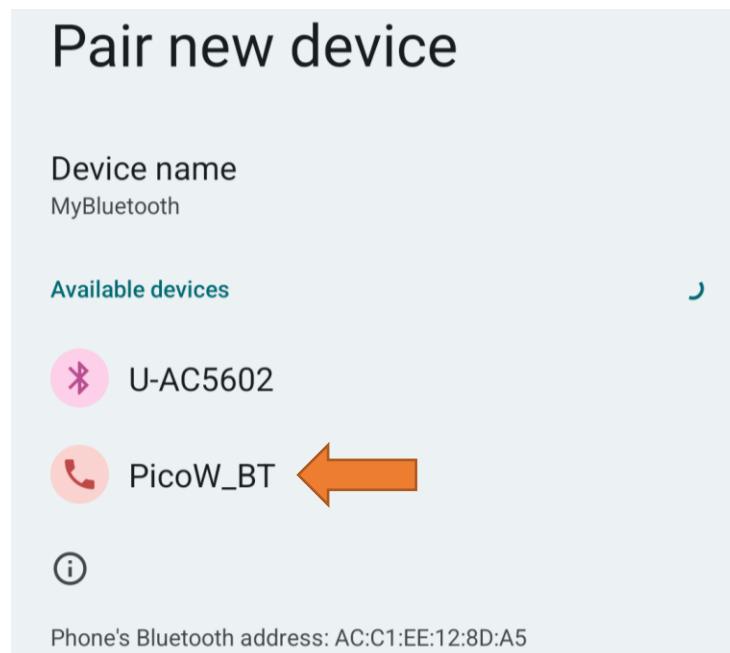
#### Serial Bluetooth

Compile and upload sketch th Pico W. The opration is similar to that in the previous section.

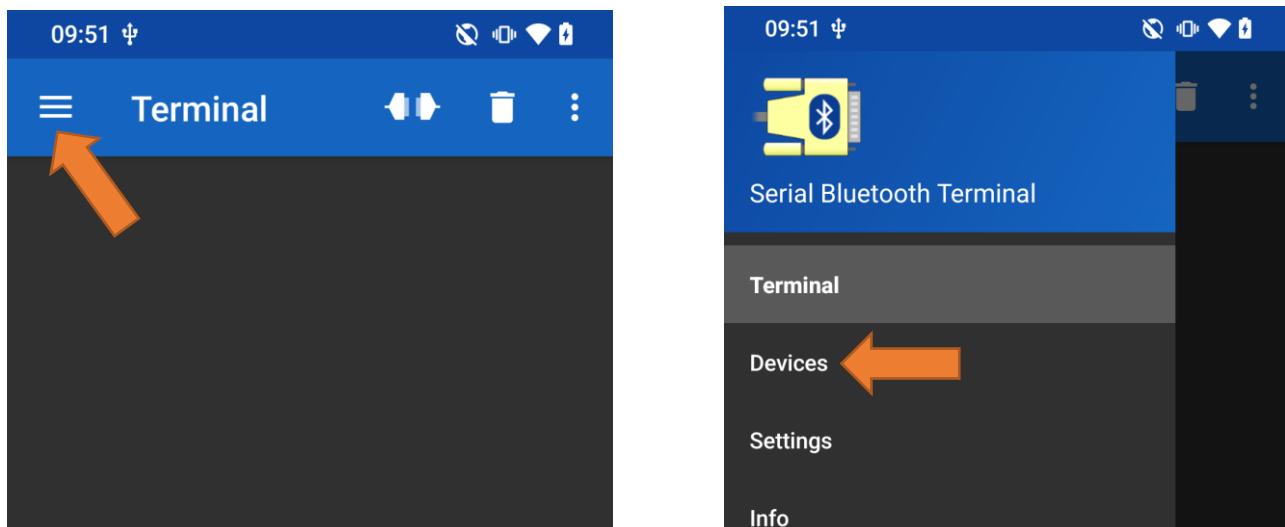
First, make sure Bluetooth on your phone is turned ON, and open Serial Bluetooth Terminal.



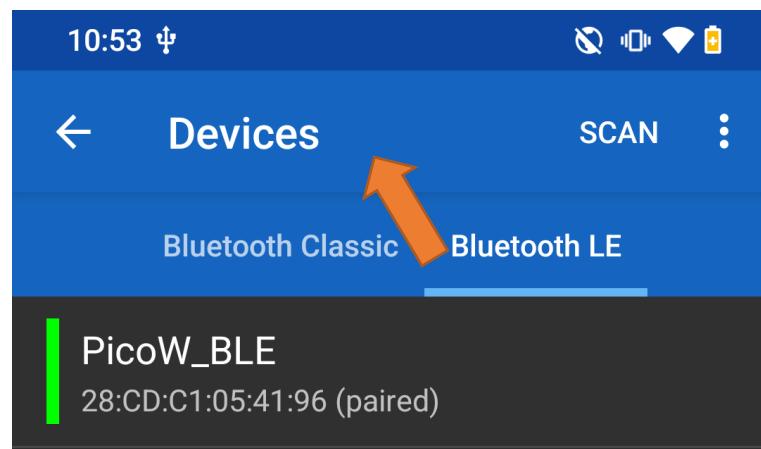
Click Pair new device, select "PicoW\_BT" to connect.



Expand the menu at the left of the app, and select "Devices".

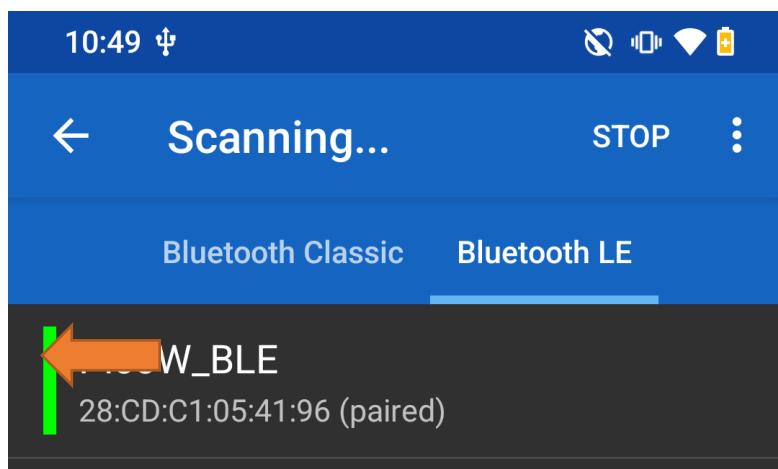


Select BLUETOOTH LE and click scan to search for BLE devices nearby.





Select “PicoW\_BLE”.



### LightBlue

If Serial Bluetooth cannot be installed on your phone, try LightBlue. If you do not have this software installed on your phone, you can refer to this link:

iPhone: <https://apps.apple.com/us/app/lightblue/id557428110?platform=iphone>



**LightBlue®** 4+

The go-to BLE development tool  
Punch Through

★★★★★ 4.0 • 4 Ratings

Free

### Screenshots Mac iPhone iPad

The screenshots demonstrate the LightBlue app's functionality across different platforms (Mac, iPhone, iPad).  
 - The first screenshot shows a list of nearby peripherals, including "Health Monitor", "Matt's Mug", "Thed's Arduino Uno", "Gretchen's Fitbit Blaze", "Heart Rate Monitor", "Colin's iPhone Xs", "Sous Vide Bean", "Mike's MacBook Pro", "PT DevKit", and "Health Monitor".  
 - The second screenshot shows the "Health Monitor" peripheral selected. It displays "ADVERTISEMENT DATA" and "Device Information" sections.  
 - The third screenshot shows the "Temperature Measurement" service for the "Health Monitor" peripheral. It includes sections for "INDICATED VALUES", "Cloud Connect" (disabled), "DESCRIPTORS", and "PROPERTIES".  
 - The fourth screenshot shows the "Characteristic Format" settings for the "Temperature Measurement" service. It lists binary values (e.g., 123456, 04432126) and their corresponding UTF-8 strings ('4V'). It also includes "Byte Count" and "Endianness" settings.

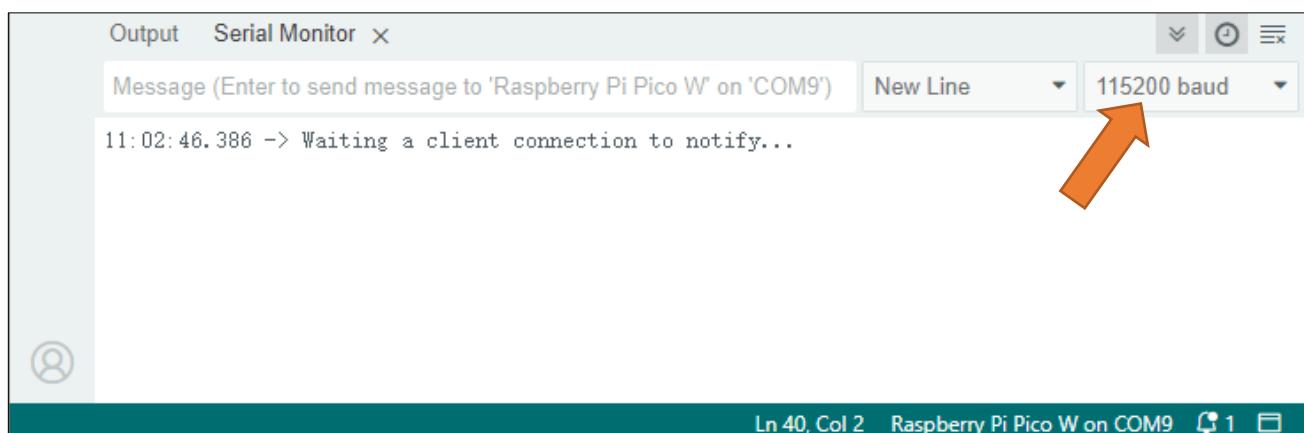
Step 1. Upload sketch 33.1 to Pico W.

Step 2. Open Serial Monitor.

```

File Edit Sketch Tools Help
Raspberry Pi Pico W
Sketch 33.1 BLE.ino
41 void setup(void) {
42     Serial.begin(115200);
43     while(!Serial);
44
45     setupBLE("PicoW_BLE");
46 }
47
48 void loop(void) {
49     BTstack.loop();
50
51     if (Serial.available() > 0) { // Check if there is serial
52         String input = Serial.readStringUntil('\n');
53         input.trim(); // Remove front and back bla
54
55         size_t input_length = input.length(); // Get String length
56     }
57 }
```

Step 3. Set the baud rate to 115200.

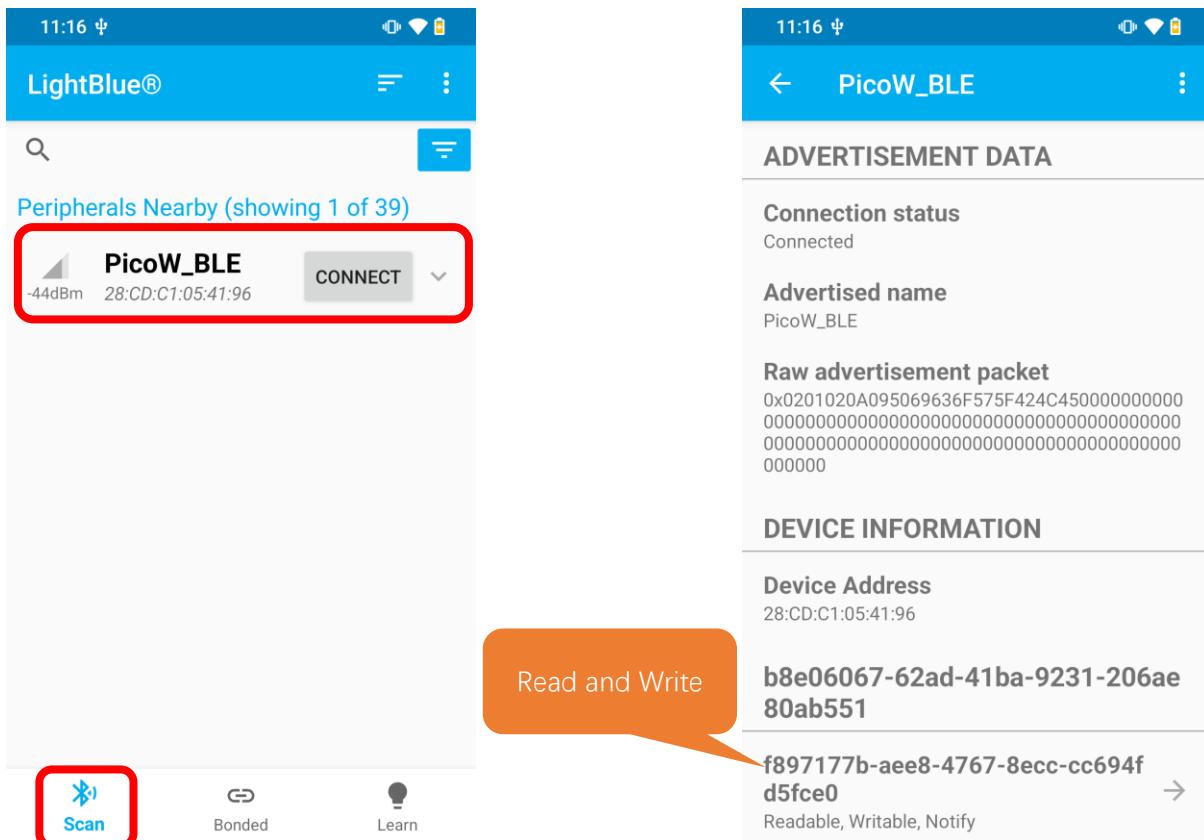


Turn ON Bluetooth on your phone and open LightBlue App.

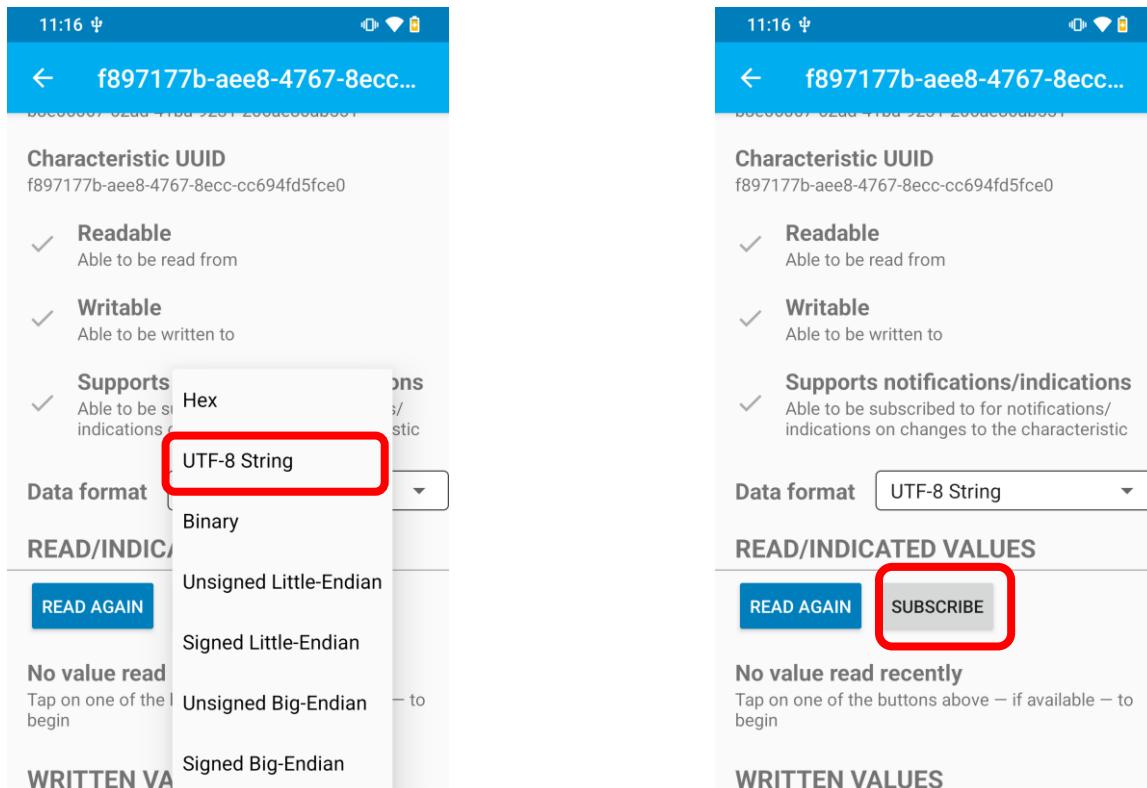


Any concerns? ✉ support@freenove.com

At the device scan page, scroll down to refresh the devices nearby. Select PicoW\_BLE to connect.



Click "Read and Write". Click on "Read and Write". Select the corresponding data format in the box on the right side of the data format, such as HEX hexadecimal, utf-string string, Binary binary, etc. Then click SUBSCRIBE.



Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

Return to the serial monitor on Arduino IDE, you can input any message and hit the Enter key to send.



Then you can see LightBlue on your phone receive the data.

The screenshot shows the LightBlue app interface. At the top, it displays the time "11:17" and signal strength. Below is a list of characteristic properties:

- ✓ **Readable**: Able to be read from
- ✓ **Writable**: Able to be written to
- ✓ **Supports notifications/indications**: Able to be subscribed to for notifications/indications on changes to the characteristic

A red box highlights the "Data format" dropdown set to "UTF-8 String".

**READ/INDICATED VALUES**

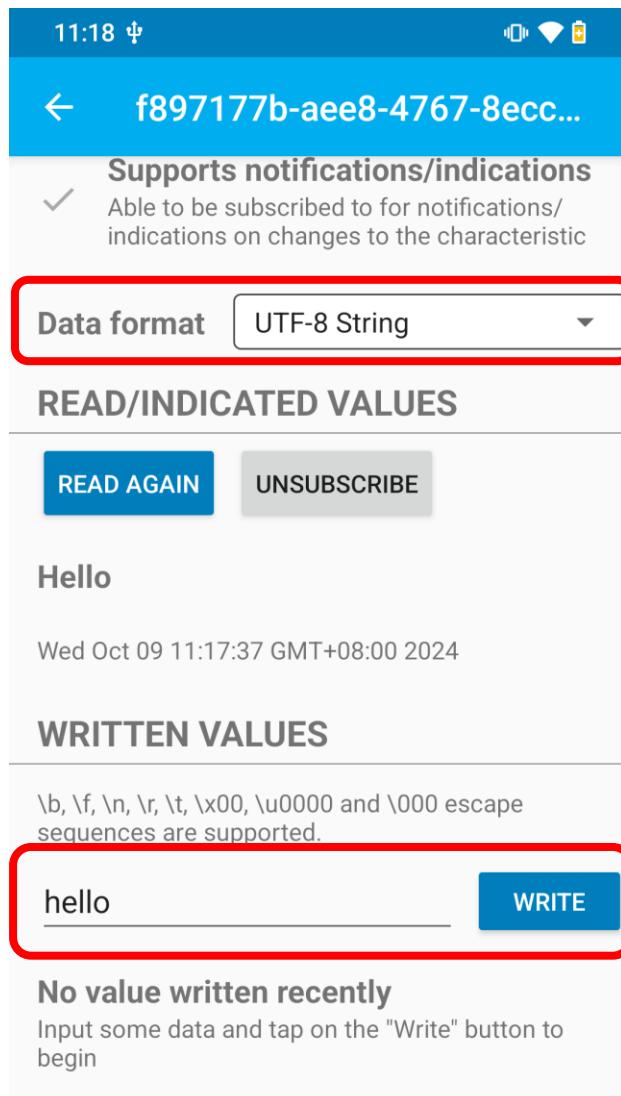
**READ AGAIN**   **UNSUBSCRIBE**

A red box highlights the value "Hello" and the timestamp "Wed Oct 09 11:17:37 GMT+08:00 2024".

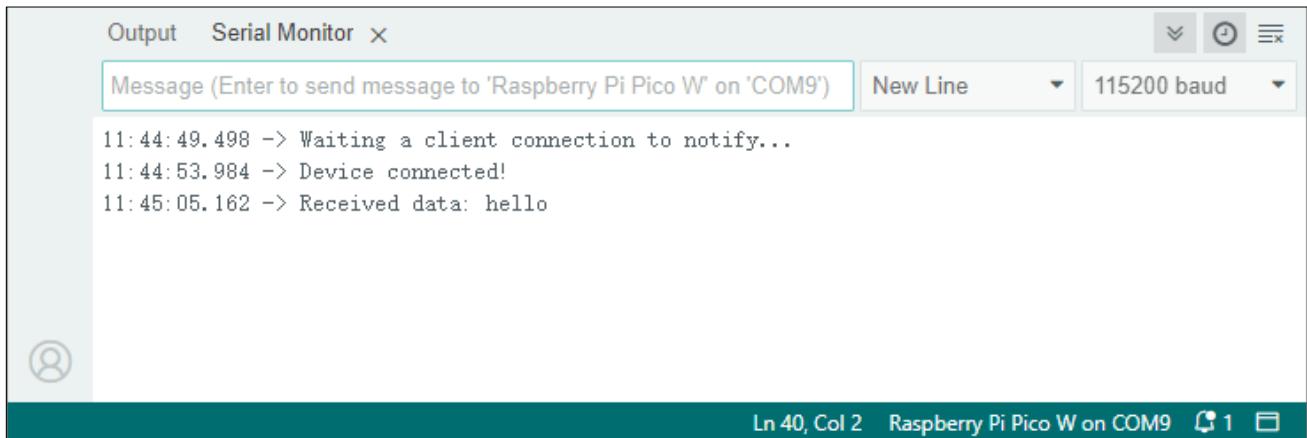
**WRITTEN VALUES**

\b, \f, \n, \r, \t, \x00, \u0000 and \000 escape sequences are supported.

Similarly, you can send data from your phone to computer. Input any content and click WRITE to send the data.



You can see the data received.



Now the data can be transferred between your phone and pico W.

The following is the program code:

```
1 #include <BTstackLib.h>
2 #include <SPI.h>
3 #include <btstack.h>
4
5 #define MAX_LENGTH 256
6 static char characteristic_data[MAX_LENGTH] = "Pico W Bluetooth";
7
8 // Track subscription status
9 bool isSubscribed = false;
10
11 // The handle of the connection
12 hci_con_handle_t connection_handle = HCI_CON_HANDLE_INVALID;
13
14 // characteristic handle
15 uint16_t characteristic_handle = 0;
16
17 // Assuming the handle of CCCD is a characteristic handle+1
18 #define CCCD_HANDLE (characteristic_handle + 1)
19
20 void setupBLE(const char *BLEName) {
21     //Set callback function
22     BTstack.setBLEDeviceConnectedCallback(deviceConnectedCallback);
23     BTstack.setBLEDeviceDisconnectedCallback(deviceDisconnectedCallback);
24     BTstack.setGATTCharacteristicRead(gattReadCallback);
25     BTstack.setGATTCharacteristicWrite(gattWriteCallback);
26
27     //Set GATT database
28     BTstack.addGATTService(new UUID("B8E06067-62AD-41BA-9231-206AE80AB551"));
29     characteristic_handle = BTstack.addGATTCharacteristicDynamic(
30         new UUID("f897177b-ae8-4767-8ecc-cc694fd5fce0"),
31         ATT_PROPERTY_READ | ATT_PROPERTY_WRITE | ATT_PROPERTY_NOTIFY,
32         0
33     );
34
35     //Activate Bluetooth and broadcast
36     BTstack.setup(BLEName);
37     BTstack.startAdvertising();
38
39     Serial.println("Waiting a client connection to notify... ");
40 }
41
42 void setup() {
43     Serial.begin(115200);
```

```
44     while(!Serial);
45
46     setupBLE("PicoW_BLE");
47 }
48
49 void loop() {
50     BTstack.loop();
51
52     if (Serial.available() > 0) {           // Check if there is serial data
53         String input = Serial.readStringUntil('\n');
54         input.trim();                      // Remove front and back blank spaces
55
56         size_t input_length = input.length(); // Get String length
57
58         // Copy input to characteristic_data and add line breaks
59         memcpy(characteristic_data, input.c_str(), input_length);
60         characteristic_data[input_length] = '\n'; // add linefeeds
61         characteristic_data[input_length + 1] = '\0'; // End of string
62
63         Serial.print("input data: ");
64         Serial.print(characteristic_data);
65
66         sendNotificationToSubscribers();        //Send notifications to subscribed devices
67     }
68     //Delay by 5ms to prevent data loss due to receiving too quickly
69     delay(5);
70 }
71
72 void deviceConnectedCallback(BLEStatus status, BLEDevice *device) {
73     if(status == BLE_STATUS_OK) {
74         Serial.println("Device connected!");
75         connection_handle = device->getHandle(); // Get connection handle
76     }
77 }
78
79 void deviceDisconnectedCallback(BLEDevice * device) {
80     (void) device;
81     Serial.println("Disconnected.");
82     connection_handle = HCI_CON_HANDLE_INVALID;
83     isSubscribed = false;                  // Reset subscription status
84 }
85
86 // Callback function for reading data
87 uint16_t gattReadCallback(uint16_t value_handle, uint8_t * buffer, uint16_t buffer_size) {
```

```
88     (void) value_handle;
89     if (buffer && buffer_size > 0) {
90         Serial.print("Read data: ");
91         Serial.println(characteristic_data);
92
93         size_t data_length = strlen(characteristic_data); // Get characteristic_data length
94         if (data_length > buffer_size) // Limit length
95             data_length = buffer_size;
96         }
97
98         memcpy(buffer, characteristic_data, data_length); // Copy String
99         return data_length;
100    }
101    return 0;
102 }
103
104 // Callback function for writing data
105 int gattWriteCallback(uint16_t value_handle, uint8_t *buffer, uint16_t size) {
106     if (value_handle == CCCD_HANDLE) { // Processing CCCD writing
107         if (size >= 2) {
108             uint16_t cccd_value = buffer[0] | (buffer[1] << 8);
109             isSubscribed = true; //Subscription status is true
110         }
111     } else { // Writing of processing feature data
112         size_t copy_size = (size < (MAX_LENGTH - 1)) ? size : (MAX_LENGTH - 1);
113         memcpy(characteristic_data, buffer, copy_size); // Copy String
114         characteristic_data[copy_size] = '\0'; // Ensure that the string ends
115
116         Serial.print("Received data: ");
117         Serial.println(characteristic_data);
118     }
119
120     return 0;
121 }
122
123 void sendNotificationToSubscribers() {
124     if (isSubscribed && connection_handle != HCI_CON_HANDLE_INVALID) {
125         // Send notifications
126         att_server_notify(connection_handle, characteristic_handle,
127                           (uint8_t*)characteristic_data, strlen(characteristic_data));
128     }
129 }
130 }
```



Define the specified UUID number for BLE vendor.

```

28  BTstack.addGATTService(new UUID("B8E06067-62AD-41BA-9231-206AE80AB551"));
29  characteristic_handle = BTstack.addGATTCharacteristicDynamic(
30      new UUID("f897177b-aee8-4767-8ecc-cc694fd5fce0"),
31      ATT_PROPERTY_READ | ATT_PROPERTY_WRITE | ATT_PROPERTY_NOTIFY,
32      0
33  );

```

Write a Callback function for the BLE server to manage BLE connections.

```

72  void deviceConnectedCallback(BLEStatus status, BLEDevice *device) {
73      if(status == BLE_STATUS_OK) {
74          Serial.println("Device connected!");
75          connection_handle = device->getHandle();           // Get connection handle
76      }
77  }

```

Write a Callback function for the BLE server to handle BLE disconnections.

```

79  void deviceDisconnectedCallback(BLEDevice * device) {
80      (void) device;
81      Serial.println("Disconnected.");
82      connection_handle = HCI_CON_HANDLE_INVALID;
83      isSubscribed = false;                                // Reset subscription status
84  }

```

Write a Callback function to read data. When BLE needs to read data, print the read data on the phone.

```

86  // Callback function for reading data
87  uint16_t gattReadCallback(uint16_t value_handle, uint8_t * buffer, uint16_t buffer_size) {
88      (void) value_handle;
89      if (buffer && buffer_size > 0) {
90          Serial.print("Read data: ");
91          Serial.println(characteristic_data);
92
93          size_t data_length = strlen(characteristic_data); // Get characteristic_data length
94          if (data_length > buffer_size) {                  // Limit length
95              data_length = buffer_size;
96          }
97
98          memcpy(buffer, characteristic_data, data_length); // Copy String
99          return data_length;
100     }
101
102 }

```

Create a Callback function for writing data. When data is written to BLE, check if it is the CCCD characteristic handle or a write characteristic handle. If it is the CCCD, it indicates a subscription state; otherwise, print the written data to the serial console.

```

104 // Callback function for writing data
105 int gattWriteCallback(uint16_t value_handle, uint8_t *buffer, uint16_t size) {

```

**Any concerns? ✉ support@freenove.com**

```

106 if (value_handle == CCCD_HANDLE) {           // Processing CCCD writing
107     if (size >= 2) {
108         uint16_t cccd_value = buffer[0] | (buffer[1] << 8);
109         isSubscribed = true;                      //Subscription status is true
110     }
111 } else { // Writing of processing feature data
112     size_t copy_size = (size < (MAX_LENGTH - 1)) ? size : (MAX_LENGTH - 1);
113     memcpy(characteristic_data, buffer, copy_size); // Copy String
114     characteristic_data[copy_size] = '\0';          // Ensure that the string ends
115
116     Serial.print("Received data: ");
117     Serial.println(characteristic_data);
118 }
119
120 return 0;
121 }
```

Write a function to send subscription notifications. When data is received through the serial port, send a notification to let BLE receive the data.

```

123 void sendNotificationToSubscribers() {
124     if (isSubscribed && connection_handle != HCI_CON_HANDLE_INVALID) {
125         // Send notifications
126         att_server_notify(connection_handle, characteristic_handle,
127                            (uint8_t*)characteristic_data, strlen(characteristic_data));
128     }
129 }
```

Process serial port data: Upon receiving data from the serial port, remove any leading and trailing whitespace, append a newline character to the end of the string, and then use the notification function to transmit the data to the mobile device via BLE.

```

52     if (Serial.available() > 0) {           // Check if there is serial data
53         String input = Serial.readStringUntil('\n');
54         input.trim();                      // Remove front and back blank spaces
55
56         size_t input_length = input.length(); // Get String length
57
58         // Copy input to characteristic_data and add line breaks
59         memcpy(characteristic_data, input.c_str(), input_length);
60         characteristic_data[input_length] = '\n'; // add linefeeds
61         characteristic_data[input_length + 1] = '\0'; // End of string
62
63         Serial.print("input data: ");
64         Serial.print(characteristic_data);
65
66         sendNotificationToSubscribers();        //Send notifications to subscribed devices
67     }
```



The design for creating the BLE server is:

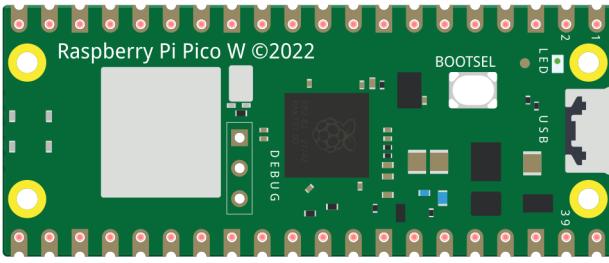
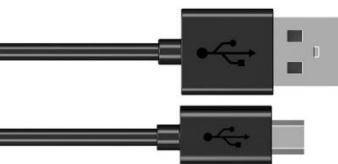
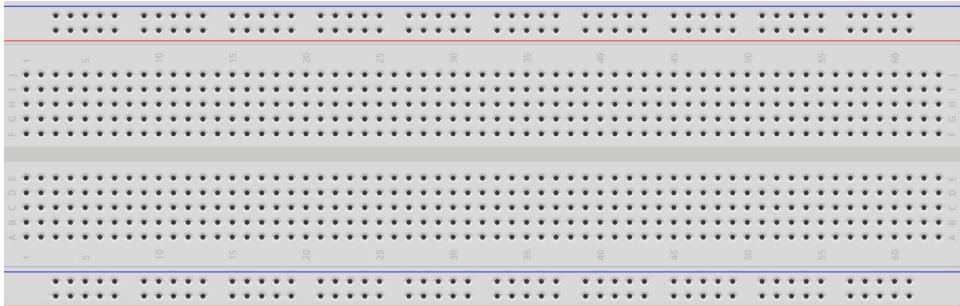
1. Set up a connection Callback function.
2. Set up a Callback function for disconnection.
3. Set up GATT read characteristic.
4. Set up GATT write characteristic.
5. Add GATT service.
6. Add GATT dynamic characteristic (read, write, notify).
7. Initialize BLE.
8. Start advertising.

```
20 void setupBLE(const char *BLEName) {  
21     //Set callback function  
22     BTstack.setBLEDeviceConnectedCallback(deviceConnectedCallback);  
23     BTstack.setBLEDeviceDisconnectedCallback(deviceDisconnectedCallback);  
24     BTstack.setGATTCharacteristicRead(gattReadCallback);  
25     BTstack.setGATTCharacteristicWrite(gattWriteCallback);  
26  
27     //Set GATT database  
28     BTstack.addGATTService(new UUID("B8E06067-62AD-41BA-9231-206AE80AB551"));  
29     characteristic_handle = BTstack.addGATTCharacteristicDynamic(  
30         new UUID("f897177b-aee8-4767-8ecc-cc694fd5fce0"),  
31         ATT_PROPERTY_READ | ATT_PROPERTY_WRITE | ATT_PROPERTY_NOTIFY,  
32         0  
33     );  
34  
35     //Activate Bluetooth and broadcast  
36     BTstack.setup(BLEName);  
37     BTstack.startAdvertising();  
38  
39     Serial.println("Waiting a client connection to notify...");  
40 }
```

## Project 15.3 Bluetooth Control LED

In this project, we will control an LED via Pico W's Bluetooth function.

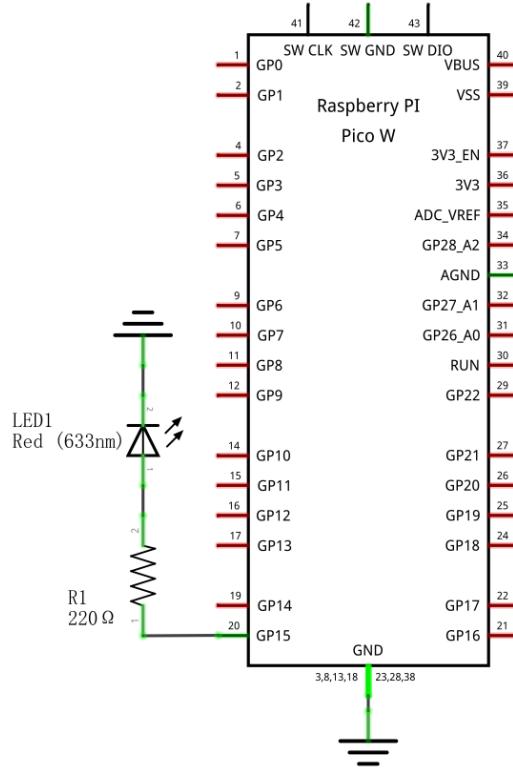
### Component List

Raspberry Pi Pico W x1	Micro USB Wire x1
	
Breadboard x1	
	  

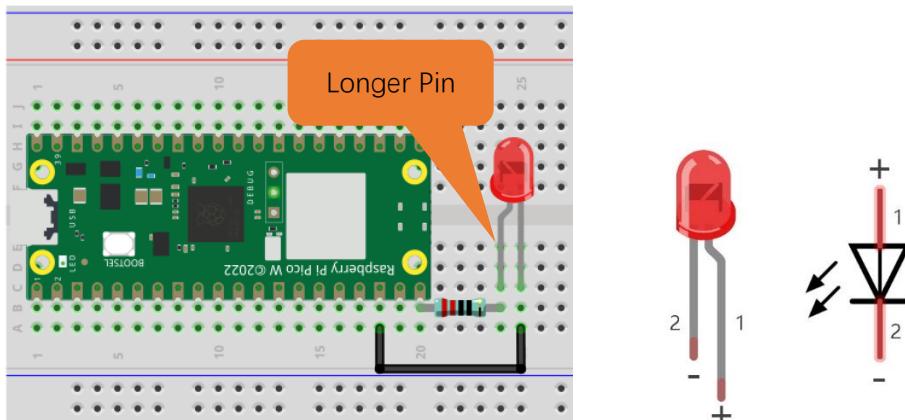
## Circuit

Connect Pico W to your phone with the USB cable.

Schematic diagram



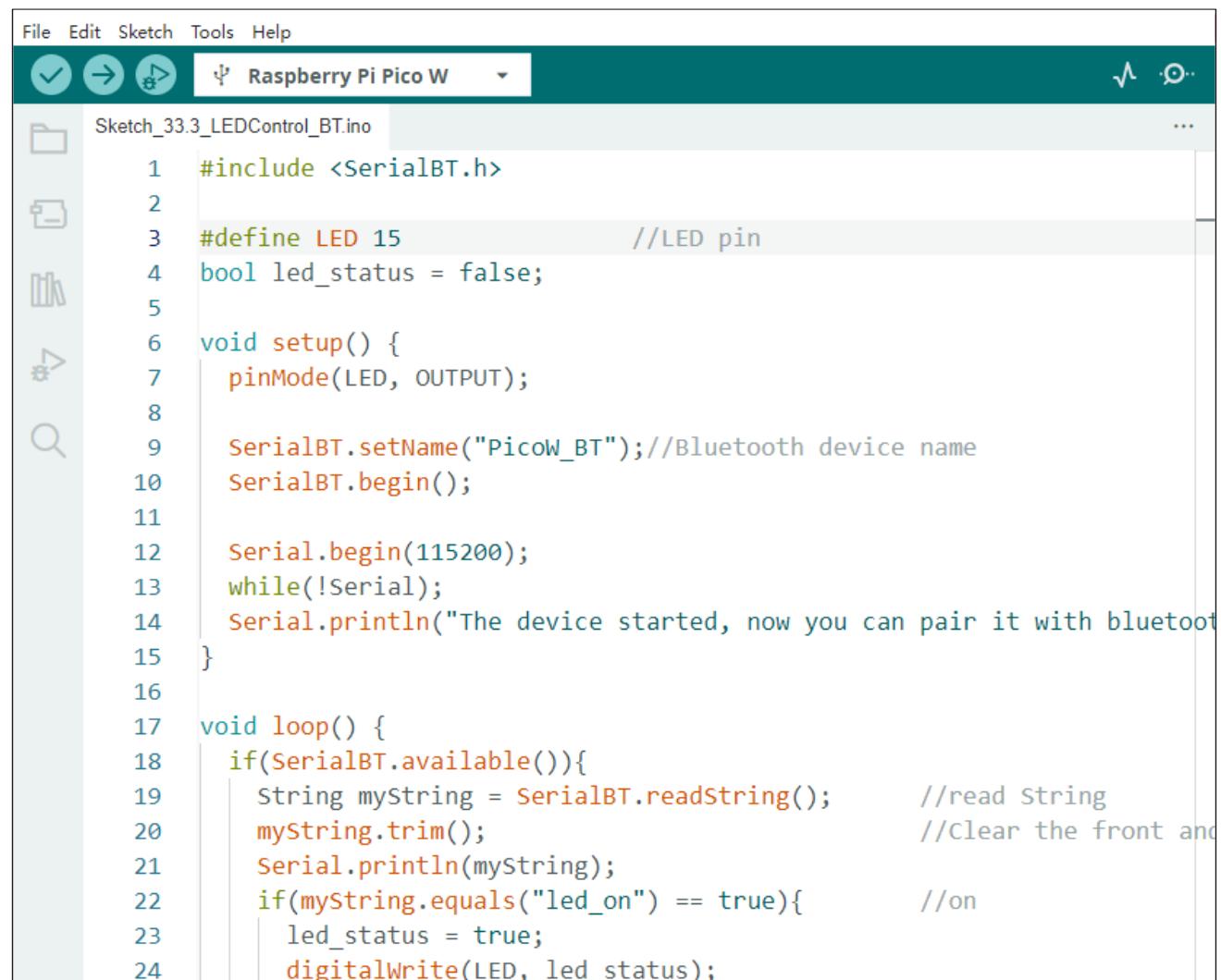
Hardware connection. If you need any support, please feel free to contact us via: [support@freenove.com](mailto:support@freenove.com)



Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

## Sketch

### Sketch\_15.3\_LEDControl\_BT



The screenshot shows the Arduino IDE interface with the sketch file `Sketch_15.3_LEDControl_BT.ino` open. The code is written in C++ and uses the `SerialBT.h` library to control a LED connected to pin 15. The sketch sets up the serial connection and begins listening for commands. It reads strings from the serial port, trims them, and then checks if they are "led\_on", "led\_off", or "led\_toggle". If "led\_on" is found, it sets the LED status to true and writes it to the pin. If "led\_off" is found, it sets the LED status to false and writes it to the pin. If "led\_toggle" is found, it toggles the LED status and writes it to the pin.

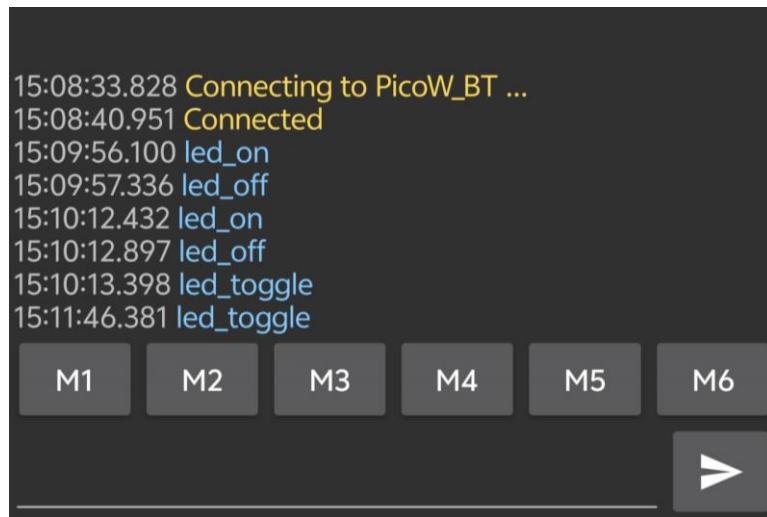
```
#include <SerialBT.h>
#define LED 15 //LED pin
bool led_status = false;

void setup() {
    pinMode(LED, OUTPUT);
    SerialBT.setName("PicoW_BT");//Bluetooth device name
    SerialBT.begin();
    Serial.begin(115200);
    while(!Serial);
    Serial.println("The device started, now you can pair it with bluetooth");
}

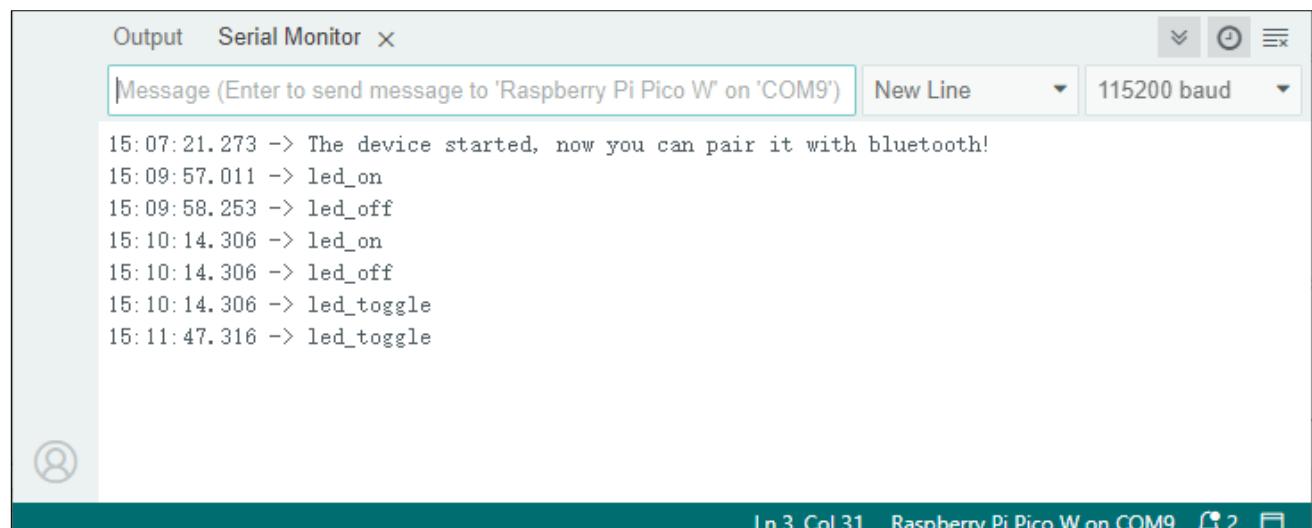
void loop() {
    if(SerialBT.available()){
        String myString = SerialBT.readString(); //read String
        myString.trim(); //Clear the front and back spaces
        Serial.println(myString);
        if(myString.equals("led_on") == true){ //on
            led_status = true;
            digitalWrite(LED, led_status);
        }
        if(myString.equals("led_off") == true){ //off
            led_status = false;
            digitalWrite(LED, led_status);
        }
        if(myString.equals("led_toggle") == true){ //toggle
            led_status = !led_status;
            digitalWrite(LED, led_status);
        }
    }
}
```

Upload the sketch to Pico W.

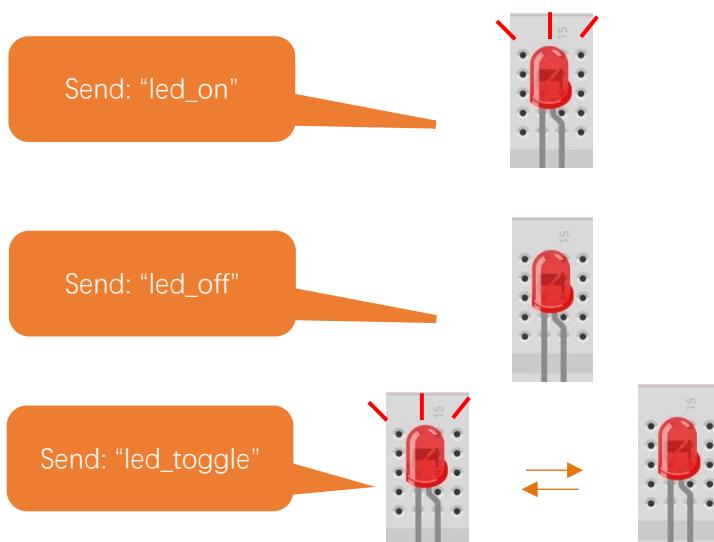
The operation on the phone app is similar to that in Project 15.1. You just need to change the sending messages to "led\_on", "led\_off" and "led\_toggle" to control the status of the LED.



Displays on the serial monitor.



LED status.



Note: If the messages you send are not "led\_on", "led\_off", or "led toggle", the status of the LED won't change. For example, when the LED is already ON, it will remain ON unless the message "led\_off", or "led toggle" is received.

Any concerns? [✉ support@freenove.com](mailto:support@freenove.com)

The following is the program code:

```

1 #include <SerialBT.h>
2
3 #define LED 15           //LED pin
4 bool led_status = false;
5
6 void setup() {
7     pinMode(LED, OUTPUT);
8
9     SerialBT.setName("PicoW_BT"); //Bluetooth device name
10    SerialBT.begin();
11
12    Serial.begin(115200);
13    while(!Serial);
14    Serial.println("The device started, now you can pair it with bluetooth!");
15 }
16
17 void loop() {
18     if(SerialBT.available()) {
19         String myString = SerialBT.readString(); //read String
20         myString.trim(); //Clear the front and back blank spaces
21         Serial.println(myString);
22         if(myString.equals("led_on") == true){ //on
23             led_status = true;
24             digitalWrite(LED, led_status);
25         }
26         if(myString.equals("led_off") == true){ //off
27             led_status = false;
28             digitalWrite(LED, led_status);
29         }
30         if(myString.equals("led_toggle") == true){ //toggle
31             led_status = !led_status;
32             digitalWrite(LED, led_status);
33         }
34     }
35 }
```

Define the pin of LED and set its default status as false.

```

3 #define LED 15           //LED pin
4 bool led_status = false;
```

Set the LED pin to output mode.

```

3 pinMode(LED, OUTPUT);
```

Process the received Bluetooth data, determine the status of the LED light. If it is "led\_on", the LED light is on; if it is "led\_off", the LED light is off; if it is "led\_toggle", the LED status is opposite to the current status.

```

18 if(SerialBT.available()) {
```

```

19   String myString = SerialBT.readString();      //read String
20   myString.trim();                            //Clear the front and back blank spaces
21   Serial.println(myString);
22   if(myString.equals("led_on") == true){        //on
23       led_status = true;
24       digitalWrite(LED, led_status);
25   }
26   if(myString.equals("led_off") == true){        //off
27       led_status = false;
28       digitalWrite(LED, led_status);
29   }
30   if(myString.equals("led_toggle") == true){    //toggle
31       led_status = !led_status;
32       digitalWrite(LED, led_status);
33   }
34 }
```

## Reference

The ‘trim()’ function is used to remove whitespace from both sides of a string, converting "led\_on\n" to "led\_on" for easier subsequent judgment.

### **void String: :trim(void)**

removes whitespace from both sides of a string.

The ‘equals()’ function is used to compare if two strings are identical, providing a simple and fast way to compare strings.

### **unsigned char String: :equals(const String &s2)**

s2: The other string to be compared.

Return value: If s2 is not equal to the current string, the return value is 0.

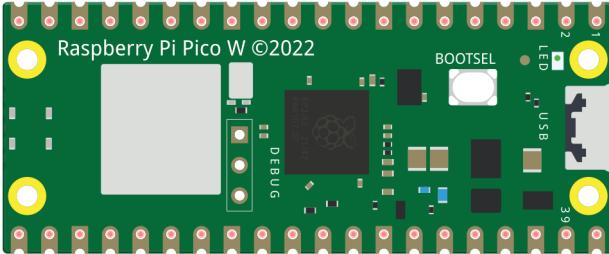
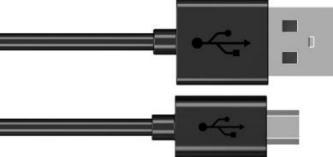
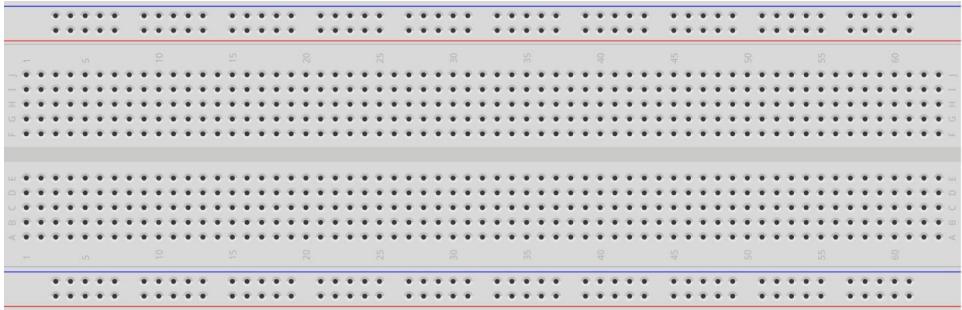
If s2 is equal to the current string, the return value is 1.

Note: Both data being compared must be of string type.

## Project 15.4 Bluetooth Low Energy Control LED

This project is basically the same as the previous one. You do not need to change the circuit wiring. Just upload the corresponding sketch.

### Component List

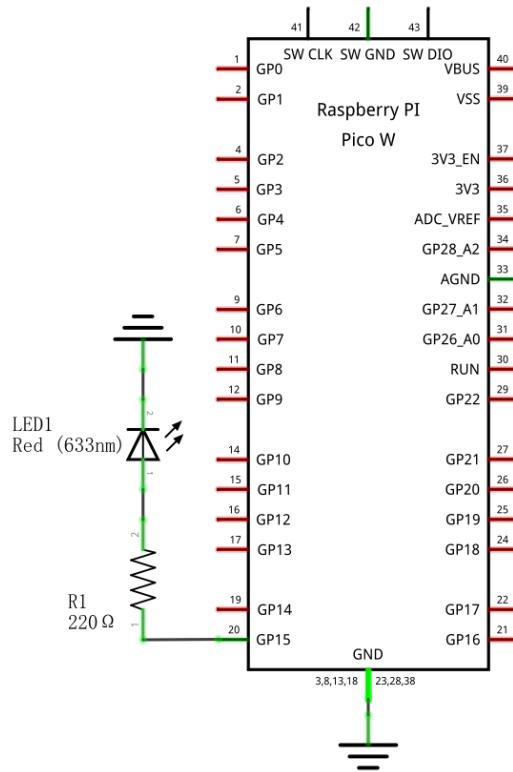
Raspberry Pi Pico W x1	Micro USB Wire x1	
		
Breadboard x1		
LED x1	Resistor 220Ω x1	Jumper



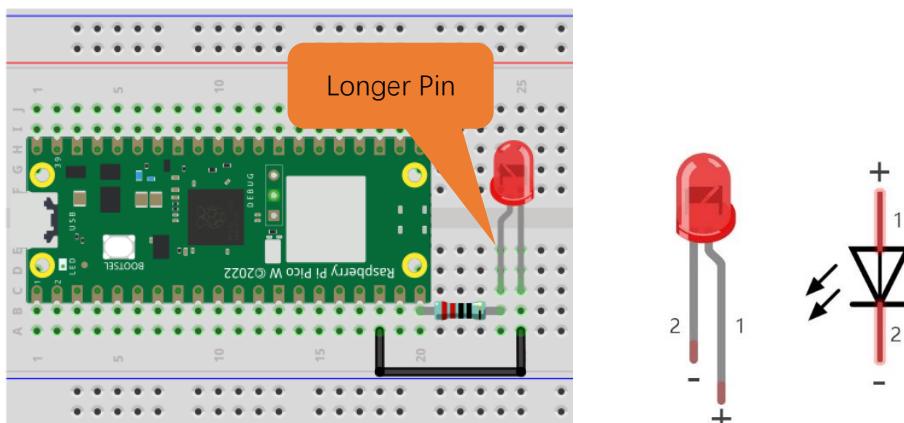
## Circuit

Connect the Pico W to your computer with a USB cable.

Schematic diagram



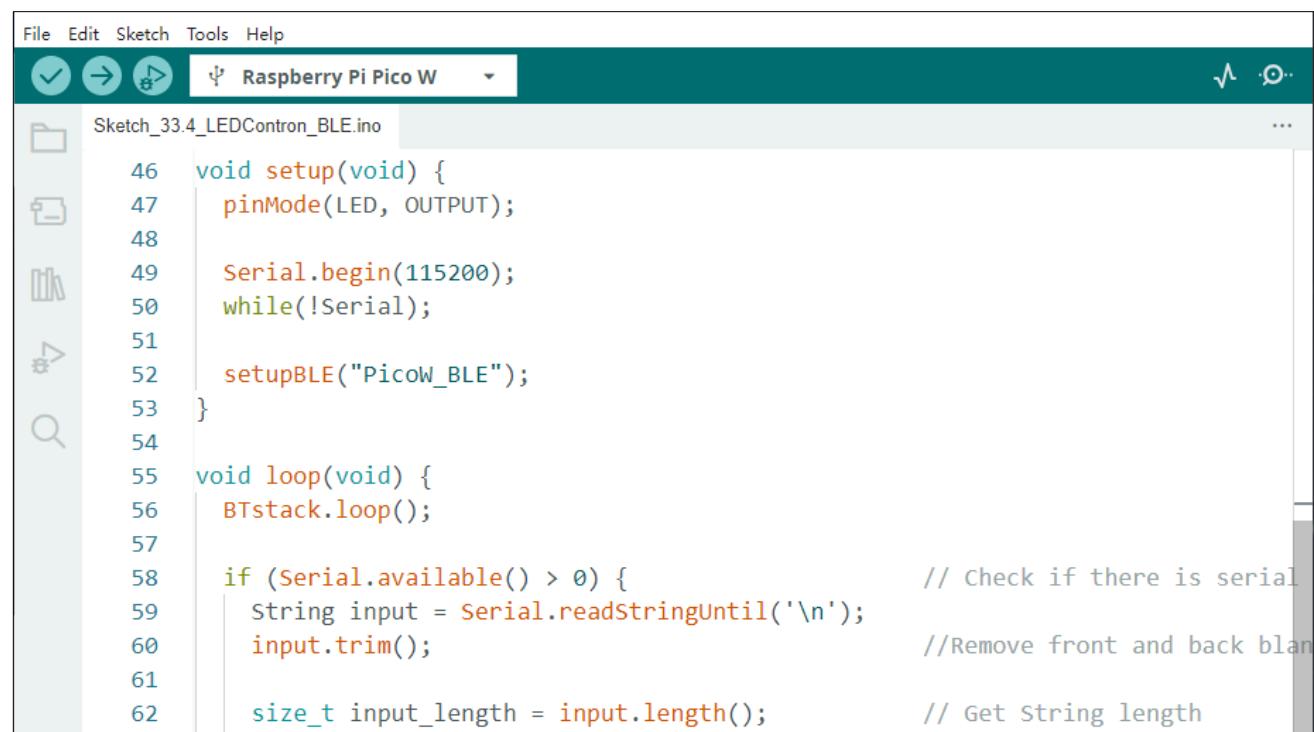
Hardware connection. If you need any support, please feel free to contact us via: [support@freenove.com](mailto:support@freenove.com)



Note: To help users have a better experience when doing the projects, we have made some modifications to Pico's simulation diagram. Please note that there are certain differences between the simulation diagram and the actual board to avoid misunderstanding.

## Sketch

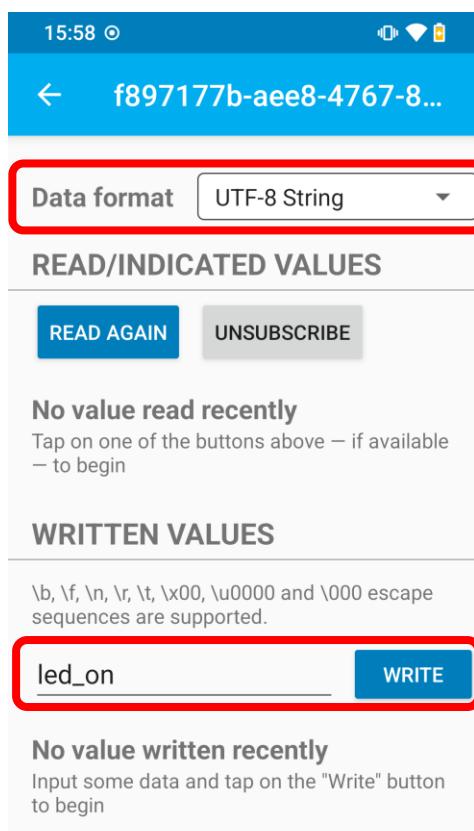
### Sketch\_15.4\_LEDControl\_BLE



```

File Edit Sketch Tools Help
Sketch_15.4_LEDControl_BLE.ino
Sketch_33.4_LEDControl_BLE.ino ...
46 void setup(void) {
47     pinMode(LED, OUTPUT);
48
49     Serial.begin(115200);
50     while(!Serial);
51
52     setupBLE("PicoW_BLE");
53 }
54
55 void loop(void) {
56     BTstack.loop();
57
58     if (Serial.available() > 0) { // Check if there is serial
59         String input = Serial.readStringUntil('\n');
60         input.trim(); //Remove front and back blank
61
62         size_t input_length = input.length(); // Get String length
    
```

Upload the sketch to Pico W. As previously mentioned, you can input “led\_on”, “led\_off”, or “led\_toggle” to change the status of the LED.





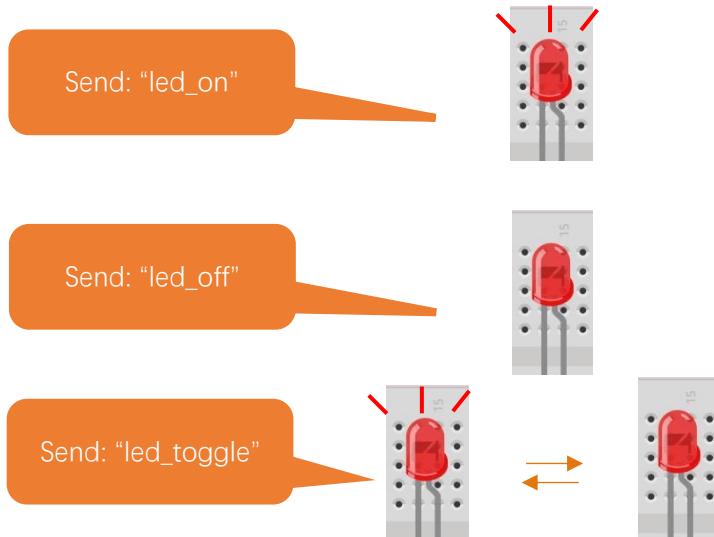
Displays on the serial monitor.

The screenshot shows the 'Serial Monitor' window with the title 'Output' and 'Serial Monitor'. The message area displays the following log:

```
Message (Enter to send message to 'Raspberry Pi Pico W' on 'COM9')  
15:59:27.800 -> Waiting a client connection to notify...  
15:59:28.952 -> Device connected!  
15:59:49.320 -> Received data: led_on  
15:59:54.347 -> Received data: led_off  
15:59:58.369 -> Received data: led_on  
16:00:01.319 -> Received data: led_off  
16:00:06.918 -> Received data: led_toggle  
16:00:07.991 -> Received data: led_toggle
```

The status bar at the bottom shows 'Ln 47, Col 18' and 'Raspberry Pi Pico W on COM9'.

LED Status.



Note: If the messages you send are not "led\_on", "led\_off", or "led toggle", the status of the LED will not change.

For example, when the LED is already ON, it will remain ON unless the message "led\_off", or "led toggle" is received.

The following is the program code:

```
1 #include <BTstackLib.h>
2 #include <SPI.h>
3 #include <btstack.h>
4
5 #define LED 15           // LED pin
6 bool led_status = false;
7
8 #define MAX_LENGTH 256
9 static char characteristic_data[MAX_LENGTH] = "Pico W Bluetooth";
10
11 // Track subscription status
12 bool isSubscribed = false;
13
14 // The handle of the connection
15 hci_con_handle_t connection_handle = HCI_CON_HANDLE_INVALID;
16
17 // characteristic handle
18 uint16_t characteristic_handle = 0;
19
20 // Assuming the handle of CCCD is a characteristic handle+1
21 #define CCCD_HANDLE (characteristic_handle + 1)
22
23 void setupBLE(const char *BLEName) {
24     //Set callback function
25     BTstack.setBLEDeviceConnectedCallback(deviceConnectedCallback);
26     BTstack.setBLEDeviceDisconnectedCallback(deviceDisconnectedCallback);
27     BTstack.setGATTCharacteristicRead(gattReadCallback);
28     BTstack.setGATTCharacteristicWrite(gattWriteCallback);
29
30     //Set GATT database
31     BTstack.addGATTService(new UUID("B8E06067-62AD-41BA-9231-206AE80AB551"));
32     characteristic_handle = BTstack.addGATTCharacteristicDynamic(
33         new UUID("f897177b-aee8-4767-8ecc-cc694fd5fce0"),
34         ATT_PROPERTY_READ | ATT_PROPERTY_WRITE | ATT_PROPERTY_NOTIFY,
35         0
36     );
37
38     //Activate Bluetooth and broadcast
39     BTstack.setup(BLEName);
40     BTstack.startAdvertising();
41
42     Serial.println("Waiting a client connection to notify...");
43 }
```

```
44
45 void setup() {
46     pinMode(LED, OUTPUT);
47
48     Serial.begin(115200);
49     while(!Serial);
50
51     setupBLE("PicoW_BLE");
52 }
53
54 void loop() {
55     BTstack.loop();
56
57     if (Serial.available() > 0) { // Check if there is serial data
58         String input = Serial.readStringUntil('\n');
59         input.trim(); // Remove front and back blank spaces
60
61         size_t input_length = input.length(); // Get String length
62
63         // Copy input to characteristic_data and add line breaks
64         memcpy(characteristic_data, input.c_str(), input_length);
65         characteristic_data[input_length] = '\n'; // add linefeeds
66         characteristic_data[input_length + 1] = '\0'; // End of string
67
68         Serial.print("input data: ");
69         Serial.print(characteristic_data);
70
71         sendNotificationToSubscribers(); //Send notifications to subscribed devices
72     }
73     //Delay by 5ms to prevent data loss due to receiving too quickly
74     delay(5);
75 }
76
77 void deviceConnectedCallback(BLEStatus status, BLEDevice *device) {
78     if(status == BLE_STATUS_OK) {
79         Serial.println("Device connected!");
80         connection_handle = device->getHandle(); // Get connection handle
81     }
82 }
83
84 void deviceDisconnectedCallback(BLEDevice * device) {
85     (void) device;
86     Serial.println("Disconnected.");
87     connection_handle = HCI_CON_HANDLE_INVALID;
```

```
88     isSubscribed = false;           // Reset subscription status
89 }
90
91 // Callback function for reading data
92 uint16_t gattReadCallback(uint16_t value_handle, uint8_t * buffer, uint16_t buffer_size) {
93     (void) value_handle;
94     if (buffer && buffer_size > 0) {
95         Serial.print("Read data: ");
96         Serial.println(characteristic_data);
97
98         size_t data_length = strlen(characteristic_data); // Get characteristic_data length
99         if (data_length > buffer_size) {                  // Limit length
100             data_length = buffer_size;
101         }
102
103         memcpy(buffer, characteristic_data, data_length); // Copy String
104         return data_length;
105     }
106     return 0;
107 }

108
109 // Callback function for writing data
110 int gattWriteCallback(uint16_t value_handle, uint8_t *buffer, uint16_t size) {
111     if (value_handle == CCCD_HANDLE) {                // Processing CCCD writing
112         if (size >= 2) {
113             uint16_t cccd_value = buffer[0] | (buffer[1] << 8);
114             isSubscribed = true;                      // Subscription status is true
115         }
116     } else {                                         // Writing of processing feature data
117         size_t copy_size = (size < (MAX_LENGTH - 1)) ? size : (MAX_LENGTH - 1);
118         memcpy(characteristic_data, buffer, copy_size); // Copy String
119         characteristic_data[copy_size] = '\0';          // Ensure that the string ends
120
121         String myString = String(characteristic_data)    // Convert to String
122         myString.trim();                                // Remove front and back blank spaces
123         if(myString.equals("1led_on") == true){          // led on
124             led_status = true;
125             digitalWrite(LED, led_status);
126         }
127         if(myString.equals("1led_off") == true){          // led off
128             led_status = false;
129             digitalWrite(LED, led_status);
130         }
131         if(myString.equals("1led_toggle") == true){        // led toggle
```

```
132     led_status = !led_status;
133     digitalWrite(LED, led_status);
134 }
135
136 Serial.print("Received data: ");
137 Serial.println(characteristic_data);
138 }
139
140 return 0;
141 }
142
143 void sendNotificationToSubscribers() {
144 if (isSubscribed && connection_handle != HCI_CON_HANDLE_INVALID) {
145 // Send notifications
146 att_server_notify(connection_handle, characteristic_handle,
147                     (uint8_t*)characteristic_data, strlen(characteristic_data));
148 }
149 }
```

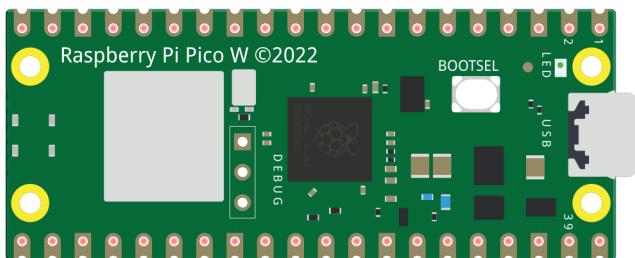
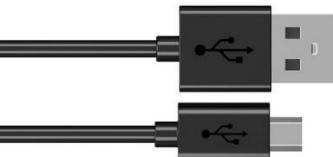
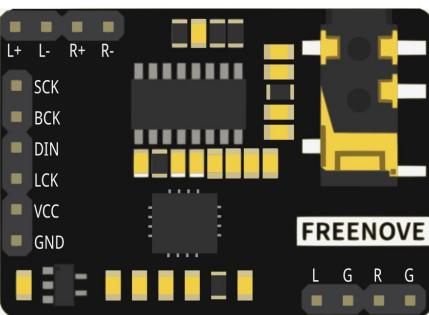
# Chapter 16 Bluetooth Audio (Only for Pico W)

Raspberry Pi Pico W integrates both Classic Bluetooth and Bluetooth Low Energy (BLE), enabling it to transmit not only simple data and commands but also files including text and audio. In this section, we will use the Bluetooth reception function to receive music from a smartphone and play it.

## Project 15.1 Bluetooth Passthrough

Utilizing the Bluetooth audio reception capability of the Raspberry Pi Pico W, we can decode audio data from a smartphone using the A2DP protocol, allowing the Raspberry Pi Pico W to receive high-quality audio. In this project, we will employ an audio converter and amplifier to transform the audio data into stereo and output it.

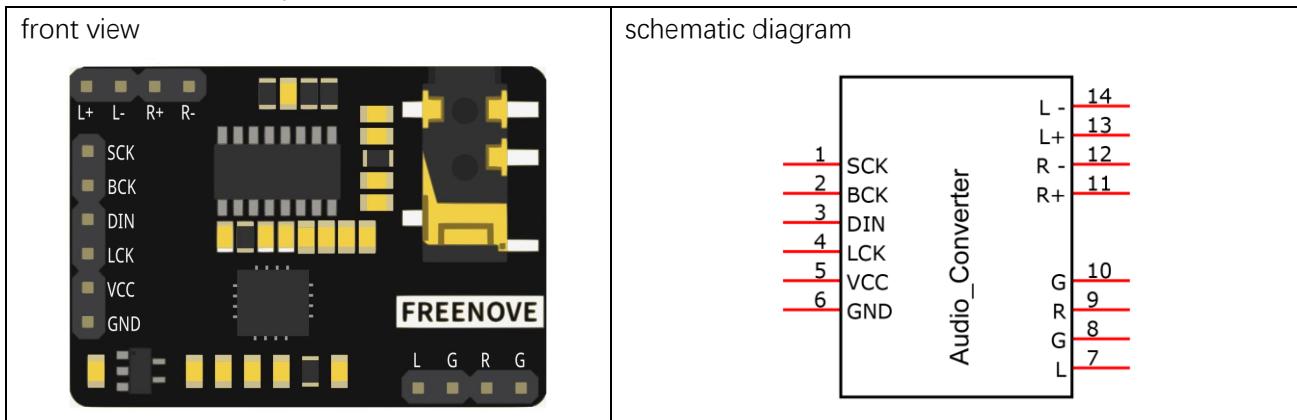
### Component List

Raspberry Pi Pico W x1	 A photograph of the Raspberry Pi Pico W development board. It is a green PCB with a central Broadcom chip, various capacitors, resistors, and connectors. Two yellow circular pads are visible on the left side. Labels include "Raspberry Pi Pico W ©2022", "DEBUG", "BOOTSEL", "LED", and "USB".	Micro USB Wire x1	 A photograph of a standard black Micro USB cable with two ends, each featuring a black plastic housing and a metal connector.
Audio Converter & Amplifier	 A photograph of the Freenove Audio Converter & Amplifier module. It is a black PCB with several surface-mount components, including a large black integrated circuit and smaller resistors and capacitors. Pin labels on the left include L+, L-, R+, R-, SCK, BCK, DIN, LCK, VCC, GND, and GND. On the right, pins are labeled L, G, R, G.	Speaker*1	 A photograph of a black speaker with a circular grille and a central black driver unit. Two wires are visible at the bottom.
Jumper wire F/M*4			 A photograph of a long, thin black jumper wire with four pins at each end, designed for connecting between the Pico W and the audio converter module.



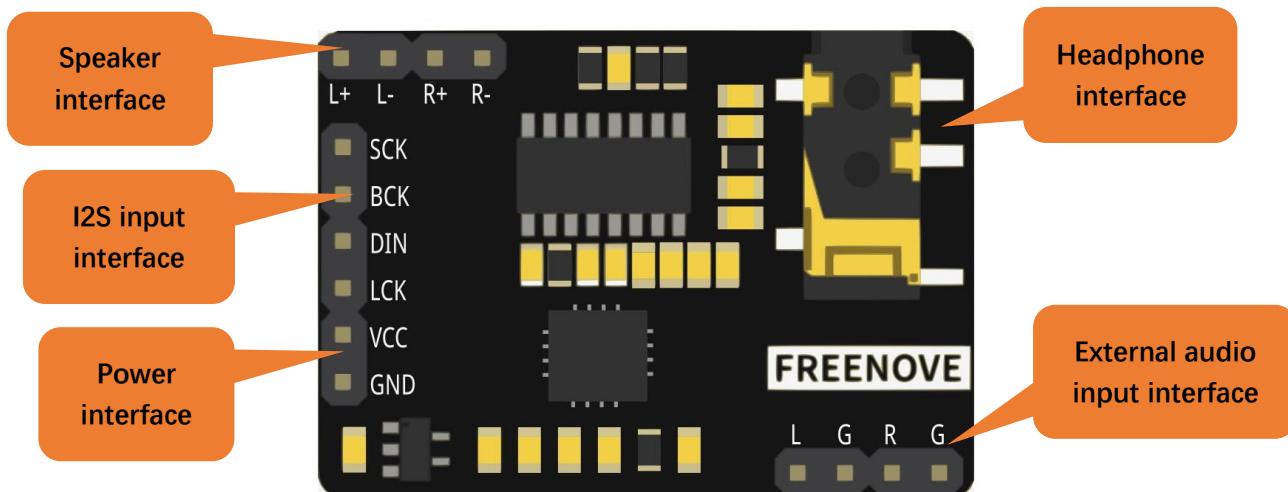
## Component knowledge

### Audio Converter & Amplifier module



### Interface description for Audio Converter & Amplifier module

Pin	Name	Introductions
1	SCK	System clock input
2	BCK	Audio data bit clock input
3	DIN	Audio data input
4	LCK	Audio data word clock input
5	VCC	Power input, 3.3V~5.0V
6	GND	Power Ground
7	L	External audio left channel input
8	G	Power Ground
9	R	External audio right channel input
10	G	Power Ground
11	R+	Positive pole of right channel horn
12	R-	Negative pole of right channel horn
13	L+	Positive pole of left channel horn
14	L-	Negative pole of left channel horn



**Speaker interface:** Connect left channel speaker and right channel speaker. Group L: L+ & L-; Group R: R+ & R-. The two interfaces of the speaker can be connected to the interfaces of group L or group R. However, when one interface is connected to group L, the other cannot be connected to group R. Doing so may cause the module to malfunction.

**Headphone interface:** the interface to connect the headphones.

**I2S input interface:** connect to the device with I2S. Used to transcode audio data into DAC audio signals.

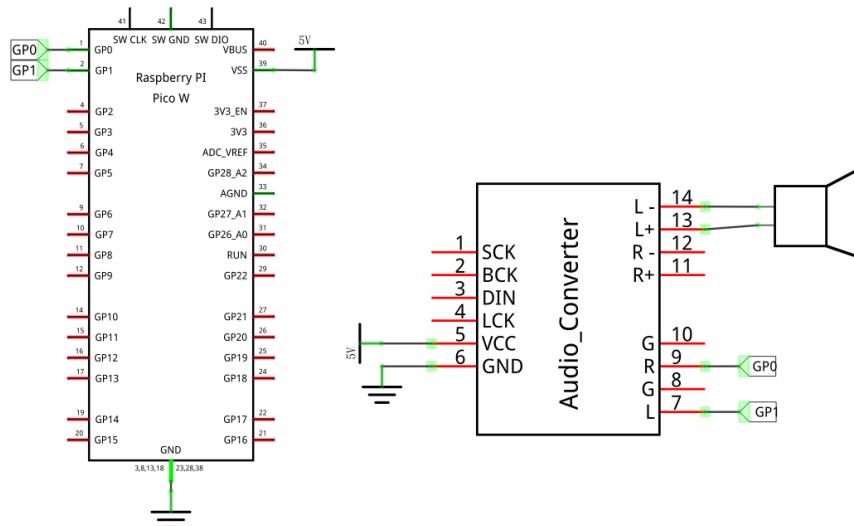
**External audio input interface:** connect to external audio equipment. Used to amplify externally input audio signals.

**Power interface:** connect to external power supply. External power supply selection range: 3.3V-5.0V.

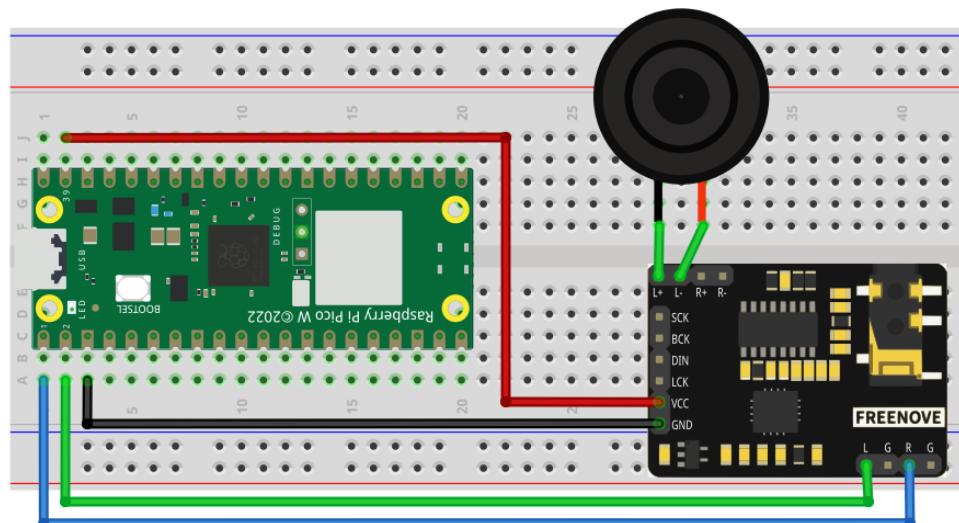
## Circuit

The connection between the control board and the audio module is shown in the figure below.

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: [support@freenove.com](mailto:support@freenove.com)





## Sketch

### Sketch\_15.4\_Bluetooth\_By\_PCM5102A

```

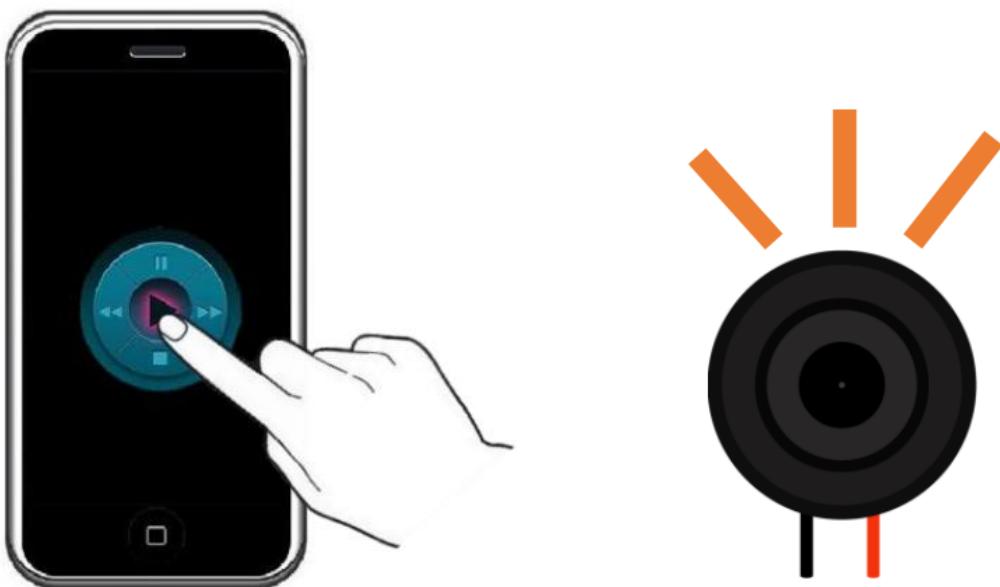
File Edit Sketch Tools Help
Sketch_15.4_Bluetooth_By_PCM5102A.ino
1 #include <BluetoothAudio.h>
2 #include <PWMAudio.h>
3
4 PWMAudio pwm;
5 A2DPSink a2dp;
6
7 // Define a playback state
8 volatile A2DPSink::PlaybackStatus status = A2DPSink::STOPPED;
9
10 // Volume callback function
11 void volumeCB(void *param, int pct) {
12     (void) param;
13     Serial.printf("Speaker volume changed to %d%\n", pct);
14 }
15
16 // Connect callback function
17 void connectCB(void *param, bool connected) {
18     (void) param;
19     if (connected) {
20         //Print the address of the connected device
21         Serial.printf("A2DP connection started to %s\n", bd_addr_to_str(a2dp.getSourceAddress()));
22     } else {
23         Serial.printf("A2DP connection stopped\n");
24     }
25 }
26
27 // Play status callback function
28 void playbackCB(void *param, A2DPSink::PlaybackStatus state) {
29     (void) param;
30     status = state;
31 }

```

When you see the messages as shown below, it indicates that the Pico W's Bluetooth is ready for connection and playing music.

Message	Time
17:38:41.169 -> Starting, connect to the PicoW and start playing music	17:38:41.169
17:38:41.169 -> Use BOOTSEL to pause/resume playback	17:38:41.169
17:38:41.918 -> NOW PLAYING:	17:38:41.918

Search “Pico W Boom” on your phone to connect. Upon successful connection, you can display audio via Pico W.



The following is the program code:

```
1 #include <BluetoothAudio.h>
2 #include <PWMAudio.h>
3
4 PWMAudio pwm;
5 A2DPSink a2dp;
6
7 // Define a playback state
8 volatile A2DPSink::PlaybackStatus status = A2DPSink::STOPPED;
9
10 // Volume callback function
11 void volumeCB(void *param, int pct) {
12     (void) param;
13     Serial.printf("Speaker volume changed to %d%%\n", pct);
14 }
15
16 // Connect callback function
17 void connectCB(void *param, bool connected) {
18     (void) param;
19     if (connected) {
20         //Print the address of the connected device
21         Serial.printf("A2DP connection started to %s\n", bd_addr_to_str(a2dp.getSourceAddress()));
22     } else {
23         Serial.printf("A2DP connection stopped\n");
24     }
25 }
```

```

27 // Play status callback function
28 void playbackCB(void *param, A2DPSink::PlaybackStatus state) {
29     (void) param;
30     status = state;
31 }
32
33 void setup() {
34     Serial.begin(115200);
35     while(!Serial);
36
37     Serial.printf("Starting, connect to the PicoW and start playing music\n");
38     Serial.printf("Use BOOTSEL to pause/resume playback\n");
39
40     a2dp.setName("PicoW Boom 00:00:00:00:00:00");
41     // Set the audio consumer to PWM audio output
42     a2dp.setConsumer(new BluetoothAudioConsumerPWM(pwm));
43     a2dp.onVolume(volumeCB);
44     a2dp.onConnect(connectCB);
45     a2dp.onPlaybackStatus(playbackCB);
46     a2dp.begin();
47 }
48
49 char *nowPlaying = nullptr; //Store the name of the currently playing track
50
51 void loop() {
52     if (BOOTSEL) {
53         if (status == A2DPSink::PAUSED) {           // if the playback status is paused
54             a2dp.play();                            // play
55             Serial.printf("Resuming\n");
56         } else if (status == A2DPSink::PLAYING) { // if the playback status is playing
57             a2dp.pause();                          // pause
58             Serial.printf("Pausing\n");
59         }
60         while (BOOTSEL);
61     }
62     if (!nowPlaying || strcmp(nowPlaying, a2dp.trackTitle())) {
63         free(nowPlaying);
64         nowPlaying = strdup(a2dp.trackTitle());
65         Serial.printf("NOW PLAYING: %s\n", nowPlaying);
66     }
67 }
```

Include two libraries: "BluetoothAudio.h" and "PWMAudio.h".

1	#include <BluetoothAudio.h>
2	#include <PWMAudio.h>

**Any concerns? ✉ support@freenove.com**

Before using the functions in these two libraries, it is necessary to construct objects of the PWMAudio class and the A2DPSink class.

```
4  PWMAudio pwm;
5  A2DPSink a2dp;
```

Define the playback status of Bluetooth audio, the default is stop status.

```
8  volatile A2DPSink::PlaybackStatus status = A2DPSink::STOPPED;
```

Write a volume callback function that is called when the volume changes and prints the volume percentage.

```
11 void volumeCB(void *param, int pct) {
12     (void) param;
13     Serial.printf("Speaker volume changed to %d%%\n", pct);
14 }
```

Write a connection callback function to print the Bluetooth device address when connected to Pico W Bluetooth.

```
17 void connectCB(void *param, bool connected) {
18     (void) param;
19     if (connected) {
20         //Print the address of the connected device
21         Serial.printf("A2DP connection started to %s\n", bd_addr_to_str(a2dp.getSourceAddress()));
22     } else {
23         Serial.printf("A2DP connection stopped\n");
24     }
25 }
```

Write a playback status Callback function to control the playback status of Bluetooth audio.

```
28 void playbackCB(void *param, A2DPSink::PlaybackStatus state) {
29     (void) param;
30     status = state;
31 }
```

The steps for creating A2DP Bluetooth audio are:

1. Set the Bluetooth name and MAC address.
2. Set the audio consumer to PWM audio output.
3. Enable callbacks for volume, connection, and playback status.
4. Initialize A2DP.

```
40 a2dp.setName("PicoW Boom 00:00:00:00:00:00");
41 // Set the audio consumer to PWM audio output
42 a2dp.setConsumer(new BluetoothAudioConsumerPWM(pwm));
43 a2dp.onVolume(volumeCB);
44 a2dp.onConnect(connectCB);
45 a2dp.onPlaybackStatus(playbackCB);
46 a2dp.begin();
```



## Reference

## Class A2DPSink

**SetName(const char \*name)**: Sets the Bluetooth module name and MAC address.

**SetConsumer(BluetoothAudioConsumer\_ \*c)**: Sets the audio consumer as an A2DP receiver.

**onVolume(void ( \*cb)(void \*, int), void \*cbData = nullptr)**: Enables the Bluetooth audio volume Callback.

**onConnect(void ( \*cb)(void \*, bool), void \*cbData = nullptr)**: Enables the Bluetooth audio connection Callback.

**onPlaybackStatus(void ( \*cb)(void \*, PlaybackStatus), void \*cbData = nullptr)**: Enables the Bluetooth audio playback status Callback.

**begin()**: Initializes A2DP.

**play()**: Changes the Bluetooth audio playback status to playing.

**pause()**: Changes the Bluetooth audio playback status to paused.

**trackTitle()**: Obtains the title of the current Bluetooth audio track.

## What's Next?

THANK YOU for participating in this learning experience!

We have reached the end of this Tutorial. If you find errors, omissions, or if you have suggestions and/or questions about the Tutorial or component contents of this Kit, please feel free to contact us:

We will make every effort to make changes and correct errors as soon as feasibly possible and publish a revised version.

If you want to learn more about Arduino, Raspberry Pi, Smart Cars, Robotics and other interesting products in science and technology, please continue to visit our website. We will continue to launch fun, cost-effective, innovative and exciting products.

<https://www.freenove.com/>

Thank you again for choosing Freenove products.