Connecting to the Internet with Raspberry Pi Pico W

Getting Raspberry Pi Pico W online with C/C++ or MicroPython

Colophon

Copyright © 2022 Raspberry Pi Ltd

The documentation of the RP2040 microcontroller is licensed under a Creative Commons Attribution-NoDerivatives 4.0 International (CC BY-ND).

build-date: 2022-06-28
build-version: 2612581-dirty

About the SDK

Throughout the text "the SDK" refers to our Raspberry Pi Pico SDK. More details about the SDK can be found in the **Raspberry Pi Pico C/C++ SDK** book. Source code included in the documentation is Copyright © 2020 Raspberry Pi (Trading) Ltd. and licensed under the 3-Clause BSD license.

Legal disclaimer notice

Table of Contents

Chapter 1. About Raspberry Pi Pico W

Raspberry Pi Pico W is a microcontroller board based on the Raspberry Pi RP2040 microcontroller chip.

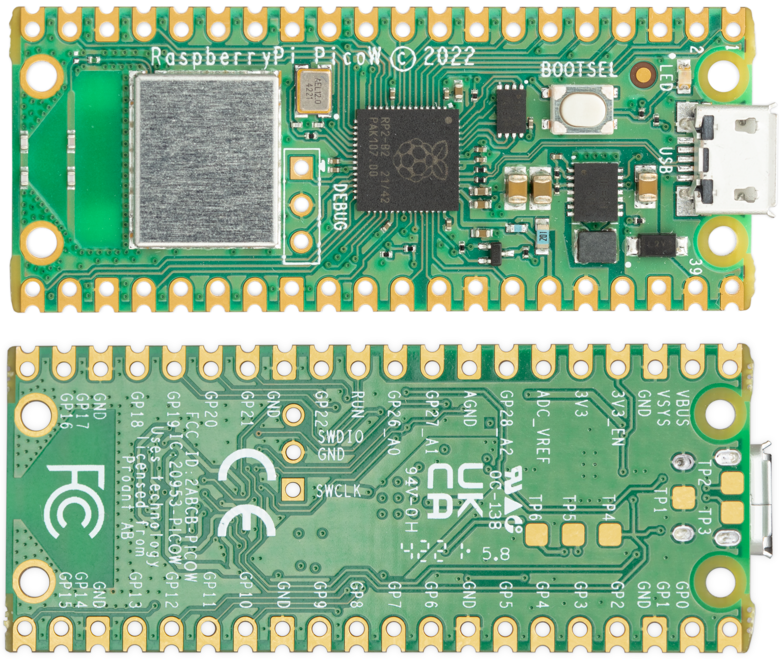


Figure 1. The Raspberry Pi Pico W Rev3 board.

Raspberry Pi Pico W has been designed to be a low cost yet flexible development platform for RP2040, with the addition of a 2.4GHz wireless interface and the following key features:

- RP2040 microcontroller with 2MB of external Flash
- On board 2.4GHz wireless (802.11n) interface
- Micro-USB B port for power and data (and for reprogramming the Flash)
- 40 pin 21mmx51mm 'DIP' style 1mm thick PCB with 0.1 inch through-hole pins also with edge castellations
  - Exposes 26 multi-function 3.3V General Purpose I/O (GPIO)
  - 23 GPIO are digital-only, with 3 more which also support ADC
  - Can be surface mounted as a module

Apart from the addition of wireless networking, Raspberry Pi Pico W is very similar to Raspberry Pi Pico and, like all RP2040-based boards, shares the same development environment. If you have not previously used an RP2040-based board you can get started by reading Getting started with Raspberry Pi Pico if you're intending to use our C SDK, or Raspberry Pi Pico Python SDK if you're thinking about using MicroPython.

NOTE

Full details of the Raspberry Pi Pico W can be found in the Raspberry Pi Pico W Datasheet.

Chapter 2. Getting on the internet with the C SDK

Wireless support for Raspberry Pi Pico W has been added to the C/C++ SDK.

NOTE

If you have not previously used an RP2040-based board you can get started by reading Getting started with Raspberry Pi Pico, while further details about the SDK along with API level documentation can be found in the Raspberry Pi Pico C/C++ SDK book.

2.1. Installing the SDK and examples

For full instructions on how to get started with the SDK see the Getting started with Raspberry Pi Pico book.

```
$ git clone https://github.com/raspberrypi/pico-sdk
$ cd pico-sdk
$ git submodule update --init
$ cd ..
$ git clone https://github.com/raspberrypi/pico-examples
```

WARNING

If you have not initialised the `tinyusb` submodule in your `pico-sdk` checkout, then USB CDC serial, and other USB functions and example code, will not work as the SDK will contain no USB functionality. Similarly, if you have not initialised the `cyw43-driver` and `lwip` submodules in your checkout, then network-related functionality will not be enabled.

2.2. Building an SDK example

Building the SDK examples, and other wireless code, requires you to specify your network SSID and password, like this:

```
$ cd pico-examples
$ mkdir build
$ cd build
$ export PICO_SDK_PATH=../../pico-sdk
$ cmake -DPICO_BOARD=pico_w -DWIFI_SSID="Your Network" -DWIFI_PASSWORD="Your Password" ..
Using PICO_SDK_PATH from environment ('../../pico-sdk')
PICO_SDK_PATH is /home/pi/pico/pico-sdk
    .
    .
    .
-- Build files have been written to: /home/pi/pico/pico-examples/build
$
```

To then build a basic example for Raspberry Pi Pico W that will scan for nearby wireless networks, you can do:

```
$ cd pico_w/wifi_scan
$ make
PICO_SDK_PATH is /home/pi/pico-sdk
PICO platform is rp2040.
Build type is Release
PICO target board is pico_w.
    .
    .
    .
[100%] Built target picow_scan_test_background
$
```

Along with other targets, we have now built:

- `picow_scan_test_background.elf`, which is used by the debugger
- `picow_scan_test_background.uf2`, which can be dragged onto the RP2040 USB mass storage device

This binary will scan for wireless networks using the Raspberry Pi Pico W's wireless chip.

The fastest method to load software onto a RP2040-based board for the first time is by mounting it as a USB mass storage device. Doing this allows you to drag a file onto the board to program the flash. Go ahead and connect the Raspberry Pi Pico W to your Raspberry Pi using a micro-USB cable, making sure that you hold down the BOOTSEL button as you do so, to force it into USB mass storage mode.

If you are running the Raspberry Pi Desktop the Raspberry Pi Pico should automatically mount as a USB mass storage device. From here, you can drag-and-drop blink.uf2 onto the mass storage device. RP2040 will reboot, unmounting itself as a mass storage device, and start to run the flashed code.

By default the code will report its results via serial UART.

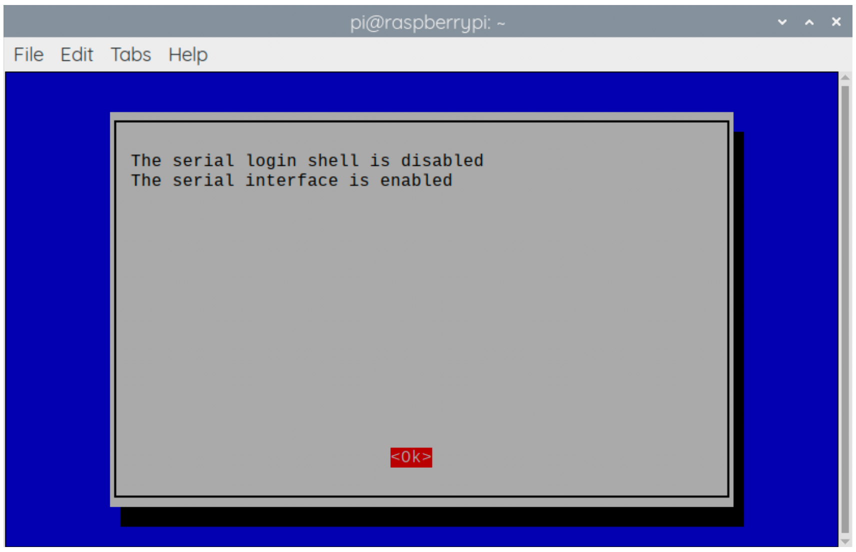
IMPORTANT

The default UART pins are configured on a per-board basis using board configuration files. The default Raspberry Pi Pico W UART TX pin (out from Pico W) is pin GP0, and the UART RX pin (in to Pico W) is pin GP1.

To see the text, you will need to enable UART serial communications on the Raspberry Pi host. To do so, run `raspi-config`:

```
$ sudo raspi-config
```

and go to `Interfacing Options` → `Serial` and select "No" when asked "Would you like a login shell to be accessible over serial?", then "Yes" when asked "Would you like the serial port hardware to be enabled?" You should see something like Figure 2.



```
The serial login shell is disabled
The serial interface is enabled
```

Figure 2. Enabling a serial UART using `raspi-config` on the Raspberry Pi.

Leaving `raspi-config` you should choose "Yes" and reboot your Raspberry Pi to enable the serial port.

You should then wire the Raspberry Pi and the Raspberry Pi Pico together with the following mapping:

Raspberry Pi	Raspberry Pi Pico
GND (Pin 14)	GND (Pin 3)
GPIO15 (UART_RX0, Pin 10)	GP0 (UART0_TX, Pin 1)
GPIO14 (UART_TX0, Pin 8)	GP1 (UART0_RX, Pin 2)

See Figure 3.

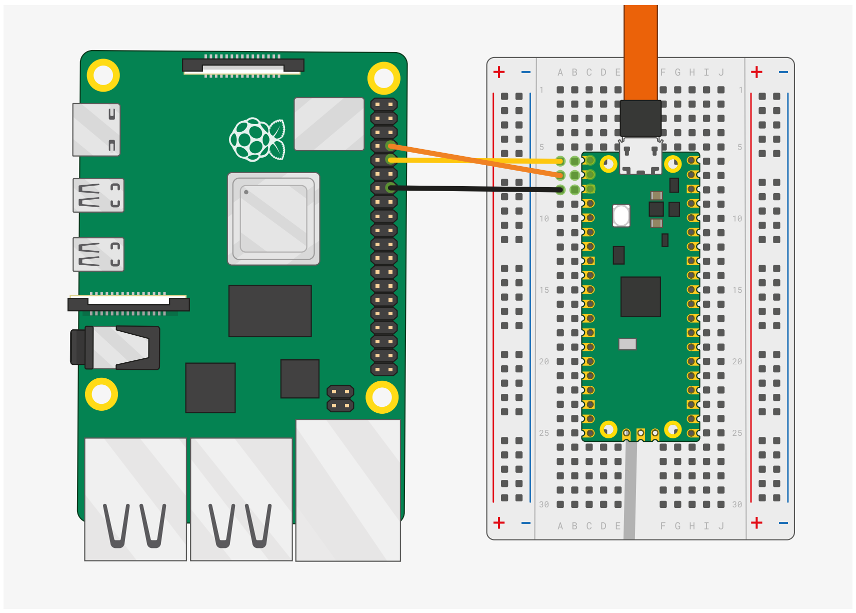


Figure 3. A Raspberry Pi 4 and the Raspberry Pi Pico with UART0 connected together.

Once the two boards are wired together you should install `minicom` if you have not already done so:

```
$ sudo apt install minicom
```

and open the serial port:

```
$ minicom -b 115200 -o -D /dev/serial0
```

You should see the results of our wireless scanning being printed to the console, see Figure 4.

TIP

To exit minicom, use `CTRL-A` followed by `X`.

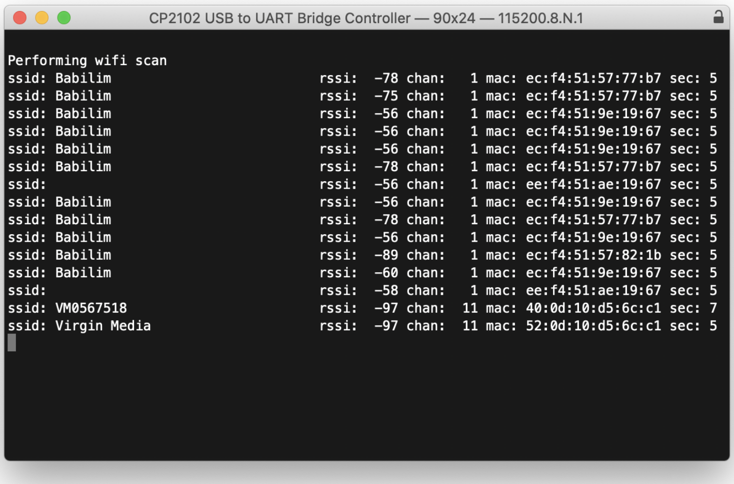


Figure 4. Results of our wireless scanning in the console

```
Performing wifi scan
ssid: Babilim                         rssi:  -78 chan:   1 mac: ec:f4:51:57:77:b7 sec: 5
ssid: Babilim                         rssi:  -75 chan:   1 mac: ec:f4:51:57:77:b7 sec: 5
ssid: Babilim                         rssi:  -56 chan:   1 mac: ec:f4:51:9e:19:67 sec: 5
ssid: Babilim                         rssi:  -56 chan:   1 mac: ec:f4:51:9e:19:67 sec: 5
ssid: Babilim                         rssi:  -56 chan:   1 mac: ec:f4:51:9e:19:67 sec: 5
ssid: Babilim                         rssi:  -78 chan:   1 mac: ec:f4:51:57:77:b7 sec: 5
ssid:                                 rssi:  -56 chan:   1 mac: ee:f4:51:ae:19:67 sec: 5
ssid: Babilim                         rssi:  -56 chan:   1 mac: ec:f4:51:9e:19:67 sec: 5
ssid: Babilim                         rssi:  -78 chan:   1 mac: ec:f4:51:57:77:b7 sec: 5
ssid: Babilim                         rssi:  -56 chan:   1 mac: ec:f4:51:9e:19:67 sec: 5
ssid: Babilim                         rssi:  -89 chan:   1 mac: ec:f4:51:57:82:1b sec: 5
ssid: Babilim                         rssi:  -60 chan:   1 mac: ec:f4:51:9e:19:67 sec: 5
ssid:                                 rssi:  -58 chan:   1 mac: ee:f4:51:ae:19:67 sec: 5
ssid: VM0567518                       rssi:  -97 chan:  11 mac: 40:0d:10:d5:6c:c1 sec: 7
ssid: Virgin Media                    rssi:  -97 chan:  11 mac: 52:0d:10:d5:6c:c1 sec: 5
```

2.3. Creating your own project

TK TK TK

2.4. Which hardware am I running on?

There is no way direct method in the C SDK that can be called to allow software to discover whether it is running on a Raspberry Pi Pico or a Pico W. However, it is possible to indirectly discover the type of underlying hardware. If the board is powered via USB or V_{sys}, so `3v3_EN` is not pulled low externally, with GPIO25 low, ADC3 will be around 0V for Raspberry Pi Pico W and approximately 1/3 of V_{sys} for Raspberry Pi Pico.

TK TK TK

Chapter 3. Getting on the internet with MicroPython

Wireless support for Raspberry Pi Pico W has been added to MicroPython. A pre-built binary which can be downloaded from the MicroPython section of the documentation website should serve most use cases and comes with `micropython-lib` pre-integrated into the binary.

NOTE

If you have not previously used an RP2040-based board you can get started by reading Raspberry Pi Pico Python SDK book.

NOTE

You can build your own MicroPython firmware from source (see if Appendix A from more details) if you'd like to customise its low-level aspects.

3.1. Installing MicroPython on Raspberry Pi Pico W

Raspberry Pi Pico W has a BOOTSEL mode for programming firmware over the USB port. Holding the BOOTSEL button when powering up your board will put it into a special mode where it appears as a USB mass storage device. First make sure your Raspberry Pi Pico W is not plugged into any source of power: disconnect the micro USB cable if plugged in, and disconnect any other wires that might be providing power to the board, e.g. through the VSYS or VBUS pin. Now hold down the BOOTSEL button, and plug in the micro USB cable (which hopefully has its other end plugged into your computer).

A drive called RPI-RP2 should pop up. Go ahead and drag the MicroPython `firmware.uf2` file onto this drive. This programs the MicroPython firmware onto the flash memory on your Raspberry Pi Pico W.

It should take a few seconds to program the UF2 file into the flash. The board will automatically reboot when finished, causing the RPI-RP2 drive to disappear, and boot into MicroPython.

When MicroPython boots for the first time, it will sit and wait for you to connect and tell it what to do. You can load a `.py` file from your computer onto the board, but a more immediate way to interact with it is through what is called the *read-evaluate-print loop*, or REPL.

There are two ways to connect to this REPL, so you can communicate with the MicroPython firmware on your board: over USB, and over the UART serial port on Raspberry Pi Pico GPIOs.

NOTE

The MicroPython port for RP2040 does not expose REPL over a UART port by default, please see Raspberry Pi Pico Python SDK for more details of how to configure MicroPython to allow you to connec to the REPL over UART.

3.2. Connecting from a Raspberry Pi over USB

The MicroPython firmware is equipped with a virtual USB serial port which is accessed through the micro USB connector on {boardname_picow_}. Your computer should notice this serial port and list it as a character device, most likely `/dev/ttyACM0`.

TIP

> You can run `ls /dev/tty*` to list your serial ports. There may be quite a few, but MicroPython's USB serial will start with `/dev/ttyACM`. If in doubt, unplug the micro USB connector and see which one disappears. If you don't see anything, you can try rebooting your Raspberry Pi.

You can install `minicom` to access the serial port:

```
$ sudo apt install minicom
```

and then open it as such:

```
$ minicom -o -D /dev/ttyACM0
```

Where the `-D /dev/ttyACM0` is pointing `minicom` at MicroPython's USB serial port, and the `-o` flag essentially means "just do it". There's no need to worry about baud rate, since this is a virtual serial port.

Press the enter key a few times in the terminal where you opened `minicom`. You should see this:

```
>>>
```

This is a *prompt*. MicroPython wants you to type something in, and tell it what to do.

If you press `CTRL-D` on your keyboard whilst the `minicom` terminal is focused, you should see a message similar to this:

```
MPY: soft reboot
MicroPython v1.18-524-g22474d25d on 2022-05-25; Raspberry Pi Pico W with RP2040
Type "help()" for more information.
>>>
```

This key combination tells MicroPython to reboot. You can do this at any time. When it reboots, MicroPython will print out a message saying exactly what firmware version it is running, and when it was built. Your version number will be different from the one shown here.

NOTE

If you are working on an Apple Mac, so long as you're using a recent version of macOS like Catalina, drivers should already be loaded. Otherwise, see the manufacturers' website for FTDI Chip Drivers. Then you should use a Terminal program to connect to Serial-over-USB (USB CDC). The serial port will show up as `/dev/cu.usbmodem` with a number appended to the end.

If you don't already have a Terminal program installed you can install `minicom` using Homebrew:

```
$ brew install minicom
```

and connect to the board as below.

```
$ minicom -b 115200 -o -D /dev/tty.usbmodem0000000000001
```

Other Terminal applications like CoolTerm or Serial can also be used.

3.2.1. Using an Integrated Development Environment

The MicroPython port to Raspberry Pi Pico and other RP2040-based boards works with commonly used development environments. Thonny is the recommended editor. Thonny packages are available for Linux, MS Windows, and macOS. After installation, using the Thonny development environment is the same across all three platforms. The latest release of Thonny can be downloaded from thonny.org.

For full details on how to use the Thonny editor, see the section on using a development environment in the Raspberry Pi Pico Python SDK book.

3.2.2. Remote access via serial port

It's suggested you use the `mpremote` tool to access the device via the serial port.

```
$ pip install mpremote
$ mpremote connect list
/dev/cu.Bluetooth-Incoming-Port None 0000:0000 None None
/dev/cu.usbmodem22201 e660583883807e27 2e8a:0005 MicroPython Board in FS mode
$ mpremote connect port:/dev/cu.usbmodem22201
Connected to MicroPython at /dev/cu.usbmodem22201
Use Ctrl-] to exit this shell
>>>
```

With this you can run a script from your local machine directly on Raspberry Pi Pico W.

```
$ pip3 install mpremote
$ mpremote run hello_world.py
```

NOTE

For more information on `mpremote` see the documentation.

3.3. The on-board LED

Unlike the original Raspberry Pi Pico, the on-board LED on Pico W is connected not to a pin on RP2040, but instead to a GPIO pin on wireless chip. MicroPython has been modified accordingly. This means that you can now do:

```
>>> import machine
>>> led = machine.Pin("LED", machine.Pin.OUT)
>>> led.off()
>>> led.on()
```

or even:

```
>>> led.toggle()
```

to change the current state. However, if you now look at the `led` object:

```
>>> led
Pin(WL_GPIO0, mode=OUT)
>>>
```

this also means that you can do the following.

```
>>> led = machine.Pin("LED", machine.Pin.OUT, value=1)
```

This will configure the `led` object, associate it with the on-board LED and turn the LED on.

NOTE

Full details of the Raspberry Pi Pico W can be found in the Raspberry Pi Pico W Datasheet. `WL_GPIO1` is connected to the `PS/SYNC` pin on the RT6154A to allow selection of different operation modes, while `WL_GPIO2` can be used to monitor USB `VBUS`.

3.4. Installing modules

You can use the `upip` tool to install modules that are not present in the default MicroPython installation.

```
>>> import upip
>>> upip.install("micropython-pystone_lowmem")
>>> import pystone_lowmem
>>> pystone_lowmem.main()
Pystone(1.2) time for 500 passes = 402ms
```

```
This machine benchmarks at 1243 pystones/second
>>>
```

3.5. Connecting to a wireless network

We're using the `network` library to talk to the wireless hardware:

```python
import network
import time

wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect('Wireless Network', 'The Password')

while not wlan.isconnected() and wlan.status() >= 0:
    print("Waiting to connect:")
    time.sleep(1)

print(wlan.ifconfig())
```

although more correctly, you should wait for the connection to succeed or fail in your code, and handle any connection errors that might occur.

```python
import time
import network

ssid = 'Wireless Network'
password = 'The Password'

wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect(ssid, password)

# Wait for connect or fail
max_wait = 10
while max_wait > 0:
    if wlan.status() < 0 or wlan.status() >= 3:
        break
    max_wait -= 1
    print('waiting for connection...')
    time.sleep(1)

# Handle connection error
if wlan.status() != 3:
    raise RuntimeError('wifi connection failed')
else:
    print('connected')
    status = wlan.ifconfig()
    print( 'ip = ' + status[0] )
```

You can also disconnect and then connect to a different wireless network.

```
# Connect to another wifi
wlan.disconnect();
wlan.connect('Other Network', 'The Other Password')
```

For more information on `network.WLAN` library see the library documentation.

3.5.1. Connection status codes

The values returned by the `wlan.status()` call are defined in the CYW43 wireless driver, and are passed directly through to user code.

```
// Return value of cyw43_wifi_link_status
#define CYW43_LINK_DOWN         (0)
#define CYW43_LINK_JOIN         (1)
#define CYW43_LINK_NOIP         (2)
#define CYW43_LINK_UP           (3)
#define CYW43_LINK_FAIL         (-1)
#define CYW43_LINK_NONET        (-2)
#define CYW43_LINK_BADAUTH      (-3)
```

3.5.2. Setting the country

By default the country setting for wireless network is unset. This means that the driver will use a default world-wide safe setting, which may mean some channels are unavailable.

```
>>> import rp2
>>> rp2.country()
'\x00\x00'
>>>
```

This can cause problems on some wireless networks. If you find that your Raspberry Pi Pico W does not connect to your wireless network you may want to try setting the country code, e.g.

```
>>> rp2.country('GB')
```

3.5.3. Powersaving mode

By default the wireless chip will active powersaving mode when it is idle, which might lead it to being less responsive. If you are running a server or need more responsiveness, you can change this by toggling the power mode.

```
import network

wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.config(pm = 0xa11140)
```

3.6. The MAC address

The MAC is in the wireless chip OTP.

```
import network
import ubinascii

wlan = network.WLAN(network.STA_IF)
wlan.active(True)
mac = ubinascii.hexlify(network.WLAN().config('mac'),':').decode()
print(mac)

# Other things you can query
print(wlan.config('channel'))
print(wlan.config('essid'))
print(wlan.config('txpower'))
```

NOTE

We have to set the wireless active (which loads the firmware) before we can get the MAC address.

3.7. Making HTTP requests

You can take a low-level or high-level approach to HTTP requests.

3.7.1. HTTP with sockets

```
# Connect to wifi
import network

wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect('Wireless Network', 'The Password')

# Should be connected and have an IP address
wlan.status() # 3 == success
wlan.ifconfig()

# Get IP address for google
import socket
ai = socket.getaddrinfo("google.com", 80)
addr = ai[0][-1]

# Create a socket and make a HTTP request
s = socket.socket()
s.connect(addr)
s.send(b"GET / HTTP/1.0\r\n\r\n")

# Print the response
print(s.recv(512))
```

3.7.2. HTTP with urequests

It is much simpler to use the `urequests` library to make an HTTP connection.

```
# Connect wifi
import network
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect('Wireless Network', 'The Password')

# Make GET request
import urequests
r = urequests.get("http://www.google.com")
print(r.content)
r.close()
```

Support has been added for redirects.

```
import urequests
r = urequests.get("http://www.raspberrypi.com")
print(r.status_code) # redirects to https
r.close()
```

NOTE

HTTPS works, but you should be aware that SSL verification is currently disabled.

The `urequests` libary comes with limited JSON support.

```
>>> r = urequests.get("http://date.jsontest.com")
>>> r.json()
{'milliseconds_since_epoch': 1652188199441, 'date': '05-10-2022', 'time': '01:09:59 PM'}
>>>>
```

For more information on `urequests` see the library documentation.

IMPORTANT

You must close the returned response object after making a request using the `urequests` library using `response.close()`. If you do not, the object will not be garbage collected, and if the request is being made inside a loop this will quickly lead to a crash.

3.7.3. Ensuring robust connections

This partial example illustrates a more robust approach to connecting to a network using `urequests`.

```
import time
import network
import urequests as requests

ssid = 'A Network'
```

```
password = 'The Password'

wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect(ssid, password)

# Wait for connect or fail
max_wait = 10
while max_wait > 0:
    if wlan.status() < 0 or wlan.status() >= 3:
        break
    max_wait -= 1
    print('waiting for connection...')
    time.sleep(1)

# Handle connection error
if wlan.status() != 3:
    raise RuntimeError('wifi connection failed')
else:
    print('connected')
    status = wlan.ifconfig()
    print( 'ip = ' + status[0] )

while True:

    # Do things here, perhaps measure something using a sensor?

    # ...and then define the headers and payloads
    headers = ...
    payload = ...

    # Then send it in a try/except block
    try:
        print("sending...")
        response = requests.post("A REMOTE END POINT", headers=headers, data=payload)
        print("sent (" + str(response.status_code) + "), status = " + str(wlan.status()) )
        response.close()
    except:
        print("could not connect (status =" + str(wlan.status()) + ")")
        if wlan.status() < 0 or wlan.status() >= 3:
            print("trying to reconnect...")
            wlan.disconnect()
            wlan.connect(ssid, password)
            if wlan.status() == 3:
                print('connected')
            else:
                print('failed')

    time.sleep(5)
```

Here we handle the possibility that we lose connection to our wireless network and then will seek to reconnect.

3.8. Building HTTP servers

You can take build synchronous or asynchronous web servers.

3.8.1. A simple server for static pages

You can use the `socket` library to build a simple web server.

```python
import network
import socket
import time

from machine import Pin

led = Pin(15, Pin.OUT)

ssid = 'A Network'
password = 'A Password'

wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect(ssid, password)

html = """<!DOCTYPE html>
<html>
    <head> <title>Pico W</title> </head>
    <body> <h1>Pico W</h1>
        <p>Hello World</p>
    </body>
</html>
"""

# Wait for connect or fail
max_wait = 10
while max_wait > 0:
    if wlan.status() < 0 or wlan.status() >= 3:
        break
    max_wait -= 1
    print('waiting for connection...')
    time.sleep(1)

# Handle connection error
if wlan.status() != 3:
    raise RuntimeError('wifi connection failed')
else:
    print('connected')
    status = wlan.ifconfig()
    print( 'ip = ' + status[0] )

# Open socket
addr = socket.getaddrinfo('0.0.0.0', 80)[0][-1]

s = socket.socket()
s.bind(addr)
s.listen(1)

print('listening on', addr)

# Listen for connections
while True:
    try:
        cl, addr = s.accept()
        print('client connected from', addr)
        cl_file = cl.makefile('rwb', 0)
        while True:
            line = cl_file.readline()
            if not line or line == b'\r\n':
                break
        response = html
        cl.send('HTTP/1.0 200 OK\r\nContent-type: text/html\r\n\r\n')
        cl.send(response)
        cl.close()

    except OSError as e:
        cl.close()
        print('connection closed')
```

NOTE

This example is synchronous, for more robust request handling you should implement the server to handle requests asynchronously.

3.8.2. Controlling an LED via a web server

Going further, we can implement a RESTful web server that will allow us to control an LED.

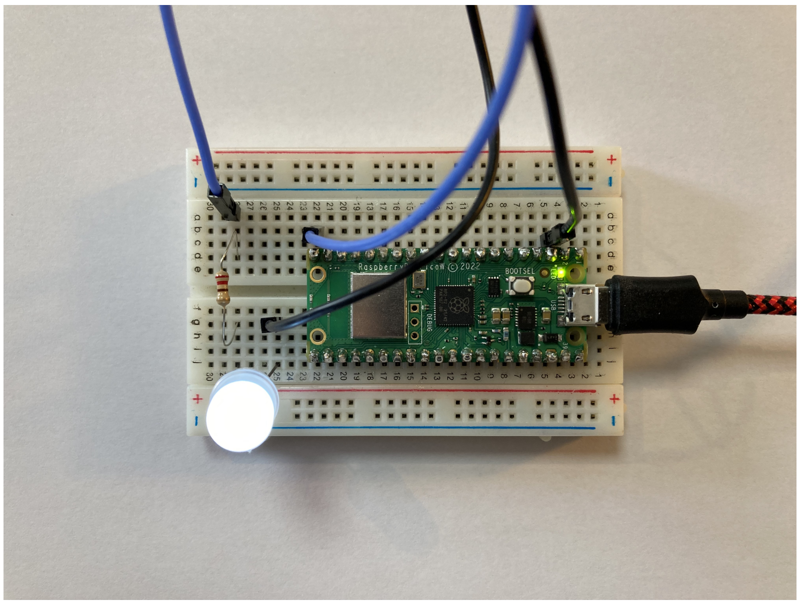
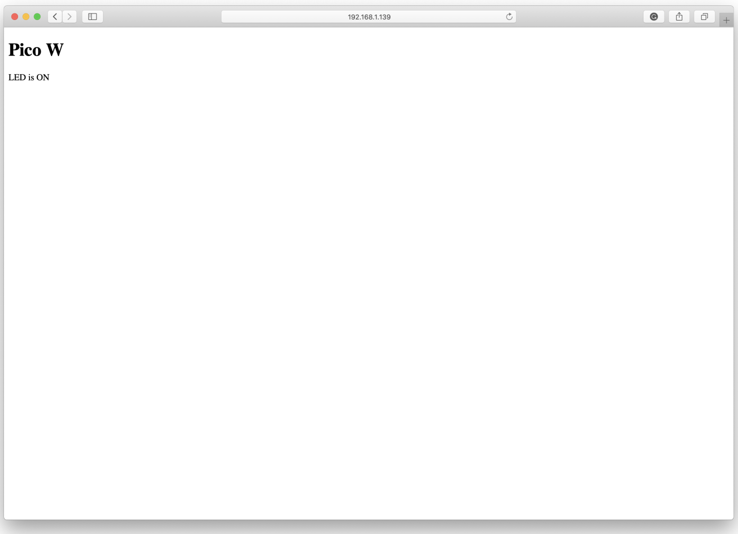


Figure 5. The Raspberry Pi Pico W with an LED on GP15.

Connecting an LED to GP15 we can turn the LED on and off by using HTTP GET. We can do this by going to `http://192.168.1.X/light/on` to turn the LED on, and `http://192.168.1.X/light/off` to turn the LED off, in our web browser where `192.168.1.X` is the IP address of our Pico W, which will be printed in the console after it connects to the network.

```
import network
import socket
import time

from machine import Pin

led = Pin(15, Pin.OUT)

ssid = 'A Newtwork'
password = 'A Password'

wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect(ssid, password)

html = """<!DOCTYPE html>
<html>
    <head> <title>Pico W</title> </head>
    <body> <h1>Pico W</h1>
        <p>%s</p>
    </body>
</html>
"""

# Wait for connect or fail
max_wait = 10
while max_wait > 0:
    if wlan.status() < 0 or wlan.status() >= 3:
        break
    max_wait -= 1
    print('waiting for connection...')
    time.sleep(1)

# Handle connection error
if wlan.status() != 3:
    raise RuntimeError('wifi connection failed')
else:
    print('connected')
    status = wlan.ifconfig()
    print( 'ip = ' + status[0] )

# Open socket
addr = socket.getaddrinfo('0.0.0.0', 80)[0][-1]

s = socket.socket()
s.bind(addr)
s.listen(1)

print('listening on', addr)

# Listen for connections
while True:
    try:
        cl, addr = s.accept()
        print('client connected from', addr)
        request = cl.recv(1024)
        print(request)

        request = str(request)
        led_on = request.find('/light/on')
        led_off = request.find('/light/off')
        print( 'led on = ' + str(led_on))
        print( 'led off = ' + str(led_off))

        if led_on == 6:
            print("led on")
            led.value(1)
            stateis = "LED is ON"

        if led_off == 6:
            print("led off")
            led.value(0)
            stateis = "LED is OFF"

        response = html % stateis

        cl.send('HTTP/1.0 200 OK\r\nContent-type: text/html\r\n\r\n')
        cl.send(response)
        cl.close()

    except OSError as e:
        cl.close()
        print('connection closed')
```

Running the code, we can see the response in our browser.


192.168.1.139

Pico W

LED is ON

Figure 6. What we see in our web browser when connecting to our Pico W web server

3.8.3. An asynchronous web server

We can use the `uasyncio` module to implement the same server, but in this case it will handle HTTP requests asynchronously rather than blocking.

```
import network
import socket
import time

from machine import Pin
import uasyncio as asyncio

led = Pin(15, Pin.OUT)
onboard = Pin("LED", Pin.OUT, value=0)

ssid = 'A Newtwork'
password = 'A Password'

html = """<!DOCTYPE html>
<html>
    <head> <title>Pico W</title> </head>
    <body> <h1>Pico W</h1>
        <p>%s</p>
    </body>
</html>
"""

wlan = network.WLAN(network.STA_IF)

```

```
def connect_to_wifi():
    wlan.active(True)
    wlan.config(pm = 0xa11140)  # Diable powersave mode
    wlan.connect(ssid, password)

    max_wait = 10
    while max_wait > 0:
        if wlan.status() < 0 or wlan.status() >= 3:
            break
        max_wait -= 1
        print('waiting for connection...')
        time.sleep(1)

    if wlan.status() != 3:
        raise RuntimeError('wifi connection failed')
    else:
        print('connected')
        status = wlan.ifconfig()
        print('ip = ' + status[0])

async def serve_client(reader, writer):
    print("Client connected")
    request_line = await reader.readline()
    print("Request:", request_line)
    # We are not interested in HTTP request headers, skip them
    while await reader.readline() != b"\r\n":
        pass

    request = str(request_line)
    led_on = request.find('/light/on')
    led_off = request.find('/light/off')
    print( 'led on = ' + str(led_on))
    print( 'led off = ' + str(led_off))

    stateis = ""
    if led_on == 6:
        print("led on")
        led.value(1)
        stateis = "LED is ON"

    if led_off == 6:
        print("led off")
        led.value(0)
        stateis = "LED is OFF"

    response = html % stateis
    writer.write('HTTP/1.0 200 OK\r\nContent-type: text/html\r\n\r\n')
    writer.write(response)

    await writer.drain()
    await writer.wait_closed()
    print("Client disconnected")

async def main():
    print('Connecting to WiFi...')
    connect_to_wifi()

    print('Setting up webserver...')
    asyncio.create_task(asyncio.start_server(serve_client, "0.0.0.0", 80))
    while True:
        onboard.on()
        print("heartbeat")
        await asyncio.sleep(0.25)
        onboard.off()
        await asyncio.sleep(5)

try:
    asyncio.run(main())
finally:
    asyncio.new_event_loop()
```

3.9. Running iperf

You can install iperf using the upip tool:

```
>>> import network
>>> wlan = network.WLAN(network.STA_IF)
>>> wlan.active(True)
>>> wlan.connect('Wireless Network', 'The Password')
>>> import upip
>>> upip.install("uiperf3")
```

and start an iperf3 client.

NOTE

The iperf server should be running on another machine.

```
>>> import uiperf3
>>> uiperf3.client('10.3.15.xx)

CLIENT MODE: TCP sending
Connecting to ('10.3.15.234', 5201)
Interval           Transfer     Bitrate
  0.00-1.00   sec  48.4 KBytes   397 Kbits/sec
  1.00-2.00   sec  48.4 KBytes   397 Kbits/sec
  2.00-3.00   sec  80.5 KBytes   659 Kbits/sec
  3.00-4.00   sec   100 KBytes   819 Kbits/sec
  4.00-5.00   sec   103 KBytes   845 Kbits/sec
  5.00-6.00   sec  22.7 KBytes   186 Kbits/sec
  6.00-7.00   sec  0.00 Bytes  0.00 bits/sec
  7.00-8.00   sec  0.00 Bytes  0.00 bits/sec
  8.00-9.00   sec  45.3 KBytes   371 Kbits/sec
  9.00-10.00  sec  89.1 KBytes   729 Kbits/sec
 10.00-10.01  sec  0.00 Bytes  0.00 bits/sec
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  0.00-10.01  sec   538 KBytes   440 Kbits/sec  sender
>>>
```

3.10. Which hardware am I running on?

There is no way direct method for software written in MircoPython to discover whether it is running on a Raspberry Pi Pico or a Pico W. However, it is possible to indirectly discover the type of underlying hardware. If the board is powered via USB or V_{sys}, so 3v3_EN is not pulled low externally, with GPIO25 low, ADC3 will be around 0V for Raspberry Pi Pico W and approximately 1/3 of V_{sys} for Raspberry Pi Pico.

```
import machine

adc3 = machine.ADC(3)
gpio25 = machine.Pin(25, machine.Pin.IN)

adc_value = adc3.read_u16()*3.3/65535

print("ADC3 = %.2f V "  % adc_value)
print("GPIO25 = " + str(gpio25.value()) )
```

Gives this for Raspberry Pi Pico W,

```
ADC3 = 0.02 V
GPIO25 = 0
```

and this for an original Raspberry Pi Pico board,

```
ADC3 = 0.41 V
GPIO25 = 0
```

Appendix A: Building MicroPython from source

Before you can proceed with building a MicroPython UF2 for Raspberry Pi Pico W from source, you should install the normal dependencies to build MicroPython. See Section 1.3 of the Raspberry Pi Pico Python SDK book for full details.

Afterwards you should clone the `micropython` and `micropython-lib` repositories.

```
$ mkdir picow
$ cd picow
$ git clone git@github.com:micropython/micropython.git
$ git clone git@github.com:micropython/micropython-lib.git
```

NOTE

Putting `micropython-lib` side-by-side with your MicroPython checkout will mean that it is automatically pulled into your MicroPython build, and libraries in `micropython-lib` will be "pre-added" to the list of modules available by default on your Pico W device.

Then build MicroPython:

```
$ cd micropython
$ make -C mpy-cross
$ cd ports/rp2
$ make BOARD=PICO_W submodules
$ make BOARD=PICO_W
```

If everything went well, there will be a new directory called `build-PICO_W` (that’s `ports/rp2/build-PICO_W` relative to the top-level `micropython` directory), which contains the new firmware binaries. Drag and drop the `firmware.uf2` onto the RPI-RP2 drive that pops up once your Raspberry Pi Pico W is in BOOTSEL mode.

Appendix B: Documentation release history

Table 1. Documentation Release History

Release	Date	Description
1.0	DD mmm 2022	Initial release

The latest release can be found at https://datasheets.raspberrypi.com/NOT_RELEASED.pdf.