Using a Mouse with USB Host

Created by Tim C

https://learn.adafruit.com/using-a-mouse-with-usb-host

Last updated on 2025-05-06 04:40:32 PM EDT

Table of Contents

Overview

This is a mouse. A nice, simple mouse. No bells or whistles. Just a mouse.

But that doesn't mean it's not the best simple mouse! We compared a few and liked this one quite a bit. It's optical for good resolution and precision, it has two buttons for use with any kind of OS, and it has a pretty sturdy USB connector. The track wheel has a good spin and click action. The right and left clicks also make a nice, satisfactory sound, which we always enjoy. It's small but not too small.

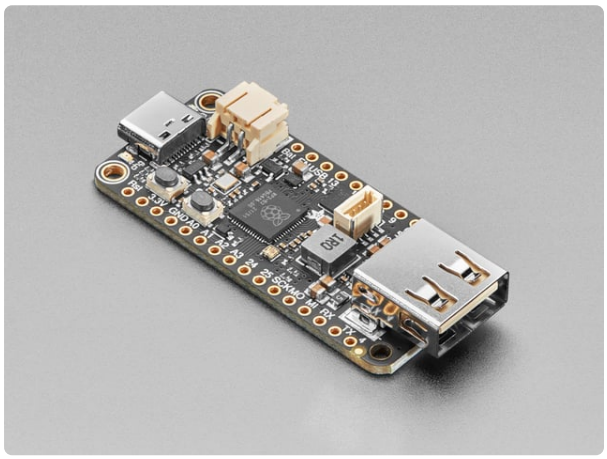
It comes in a nice Adafruit black as seen in the beautiful photos above. It's also big enough to fit your hand comfortably.

This will work well with Raspberry Pi and any other kind of computer, as it's a standard HID mouse!

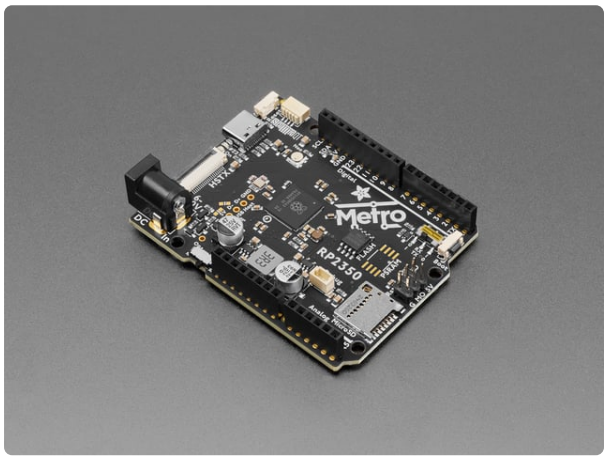
This guide will show how to use the mouse via USB Host on a microcontroller with CircuitPython and Arduino.

Parts

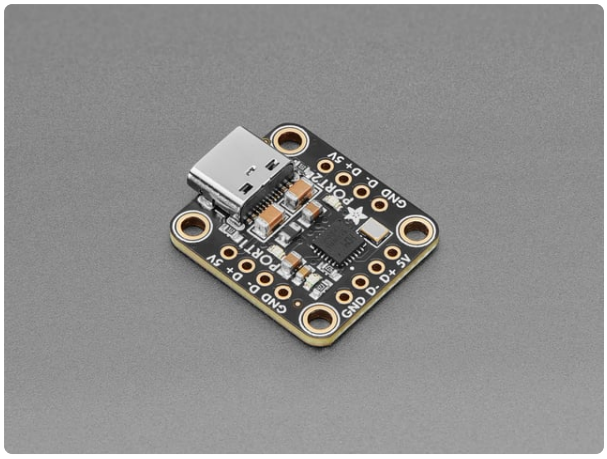
[USB Wired Mouse - Two Buttons plus Wheel]
This is a mouse. A nice, simple mouse. No bells or whistles. Just a mouse.But that doesn't mean it's not the best simple mouse! We compared...
https://www.adafruit.com/product/2025



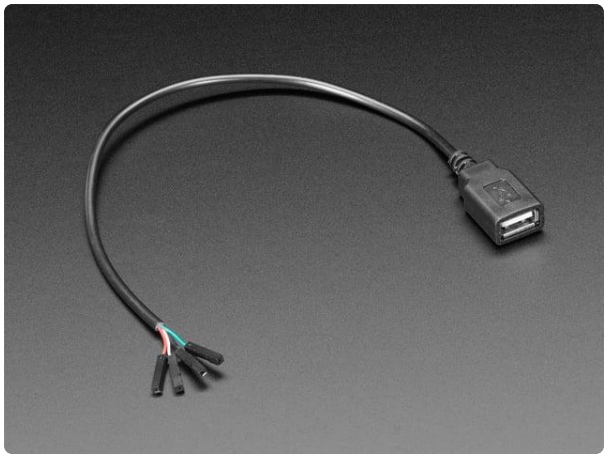
[Adafruit Feather RP2040 with USB Type A Host]
You're probably really used to microcontroller boards with USB, but what about a dev board with two? Two is more than one, so that makes it twice as good! And...
https://www.adafruit.com/product/5723



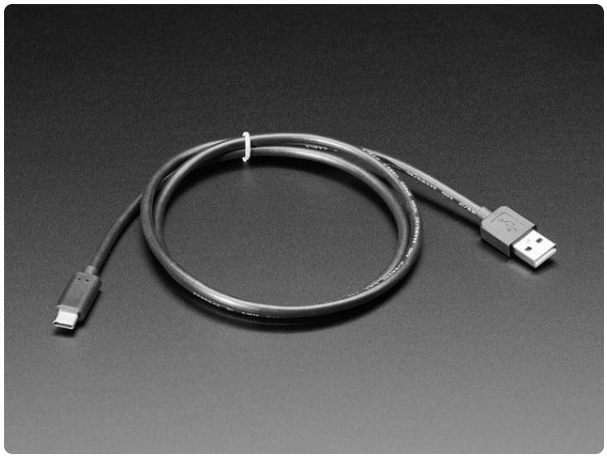
[Adafruit Metro RP2350]
Choo! Choo! This is the RP2350 Metro Line, making all station stops at "Dual Cortex M33 mountain", "528K RAM round-about" and "16 Megabytes of Flash...
https://www.adafruit.com/product/6003



[Adafruit CH334F Mini 2-Port USB Hub Breakout]
Sometimes, you've got something with a USB host, like an embedded Linux board, and you want to connect more than one thing. Or maybe you want to turn something like a keyboard into...
https://www.adafruit.com/product/5999



[USB Type A Jack Breakout Cable with Premium Female Jumpers]
If you'd like to connect a USB-host-capable chip to your USB peripheral, this cable will make the task very simple. There is no converter chip in this...
https://www.adafruit.com/product/4449



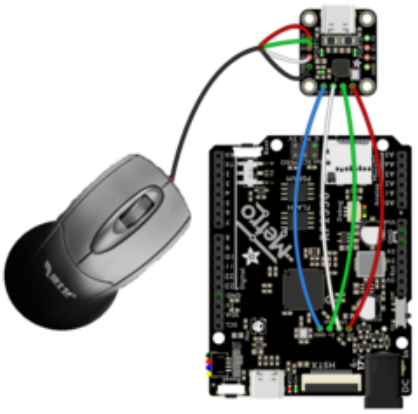
[USB Type A to Type C Cable - approx 1 meter / 3 ft long]
As technology changes and adapts, so does Adafruit. This USB Type A to Type C cable will help you with the transition to USB C, even if you're still...
https://www.adafruit.com/product/4474

CircuitPython

To read data from a mouse in CircuitPython, you must use a device that supports USB Host, such as the Metro RP2350 (http://adafru.it/6003) or Feather RP2040 USB Host (http://adafru.it/5723).

Feather RP2040 USB Host Wiring

Connecting to the Feather RP2040 USB Host requires plugging in a USB C cable to the Feather's Host port and connecting the other end to a USB Hub breakout such as the CH334F (http://adafru.it/5999).



Metro RP2350 USB Host Wiring

Connecting to the Metro RP2350 USB Host port requires soldering pins to the broken-out USB Host connections as shown on this guide page (https://adafru.it/1ahU). Make the following connections between the Metro USB Host pins and the CH334F host connection opposite the USB C connector.

GND to **GND** with **Black** or **Blue**
D+ to **D+** with **Green**
D- to **D-** with **White**
5V to **5V** with **Red**

Note that the data pins are swapped on the Metro compared to the CH334F breakout. On the Metro **D-** is next to **5V**, whereas on the CH334F **D-** is next to **GND**.

Be sure to connect **D-** on the breakout to **D-** on the Metro, and **D+** on the breakout to **D+** on the Metro.

USB Host is under active development. As of CircuitPython version 10.0.0-alpha.2, the wired USB Mouse only works on CircuitPython when connected through a USB hub such as the CH334F.

Demo Code: Read Data

This example uses the USB Host API to read data from the mouse and prints out the relevant values to the serial console. The CircuitPython USB Host API is made to mimic PyUSB from CPython.

To read data from the mouse, the code must first scan the connected devices and find one with a boot mouse endpoint. Once a mouse device and endpoint are found, the code will try to read data from them in the main loop. If no data is sent, it will timeout and simply keep trying again until data is present. Once it reads data from the mouse, it will print any buttons that are pressed along with the delta X and delta Y values that represent how far the mouse has been moved and in which directions.

The list `BUTTONS` is created with values `"left"`, `"right"`, and `"middle"`. The order and indexes of these values align with the mouse protocol, which will use bits in the same positions to denote whether each button is being pressed or not.

```
# SPDX-FileCopyrightText: 2025 Tim Cocks for Adafruit Industries
# SPDX-License-Identifier: MIT
"""
This example is made for a basic mouse with
two buttons and a wheel that can be pressed.

It assumes there is a single mouse connected to USB Host.
"""
import array
import usb.core
import adafruit_usb_host_descriptors

# button names
# This is ordered by bit position.
BUTTONS = ["left", "right", "middle"]

# scan for connected USB device and loop over any found
for device in usb.core.find(find_all=True):
    # print device info
    print(f"{device.idVendor:04x}:{device.idProduct:04x}")
    print(device.manufacturer, device.product)
    print(device.serial_number)

    # try to find mouse endpoint on the current device.
    mouse_interface_index, mouse_endpoint_address = (
        adafruit_usb_host_descriptors.find_boot_mouse_endpoint(device)
    )
    if mouse_interface_index is not None and mouse_endpoint_address is not None:
        mouse = device
        print(
            f"mouse interface: {mouse_interface_index} "
            + f"endpoint_address: {hex(mouse_endpoint_address)}"
        )

        # detach the kernel driver if needed
        if mouse.is_kernel_driver_active(0):
            mouse.detach_kernel_driver(0)

        # set configuration on the mouse so we can use it
        mouse.set_configuration()

        break
```

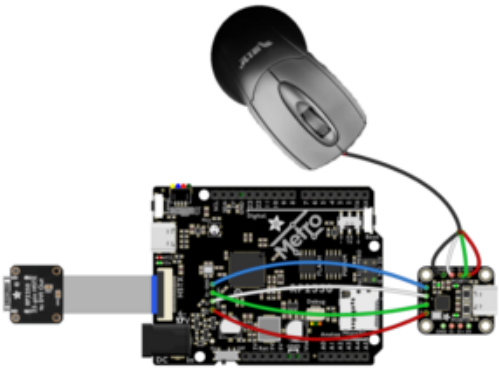
```
# buffer to hold mouse data
buf = array.array("b", [0] * 8)

# main loop
while True:
    try:
        # attempt to read data from the mouse
        # 20ms timeout, so we don't block long if there
        # is no data
        count = mouse.read(0x81, buf, timeout=20)
    except usb.core.USBTimeoutError:
        # skip the rest of the loop if there is no data
        continue

    # string with delta x, y values to print
    out_str = f"{buf[1]},{buf[2]}"

    # loop over the button names
    for i, button in enumerate(BUTTONS):
        # check if each button is pressed using bitwise AND shifted
        # to the appropriate index for this button
        if buf[0] & (1 << i) != 0:
            # append the button name to the string to show if
            # it is being clicked.
            out_str += f" {button}"

    print(out_str)
```

Here is a sample output using the Adafruit mouse. If you have a different mouse, the identification information will likely be different, but the functionality should be the same.

```
code.py output:
1a2c:0044
None None
None
1a2c:0044
SIGMACHIP Usb Mouse
None
mouse interface: 0 endpoint_address: 0x81
1,0
9,4
27,0
0,0 left right
1,1 left right
1,1 left right
-1,-1
0,-1
-1,1
-1,1
```

CircuitPython Displayio Cursor

To use a mouse to draw and move a visible cursor on a display in CircuitPython requires a device that supports both USB Host and displayio. One such device is the Metro RP2350 with its HSTX connection for DVI display output.



Metro RP2350 Wiring

Connecting to the Metro RP2350 USB Host port requires soldering pins to the broken out USB Host connections as shown in this guide page (https://adafru.it/1ahU). Make the following connections between the Metro USB Host pins and CH334F host connection opposite the USB C connector.

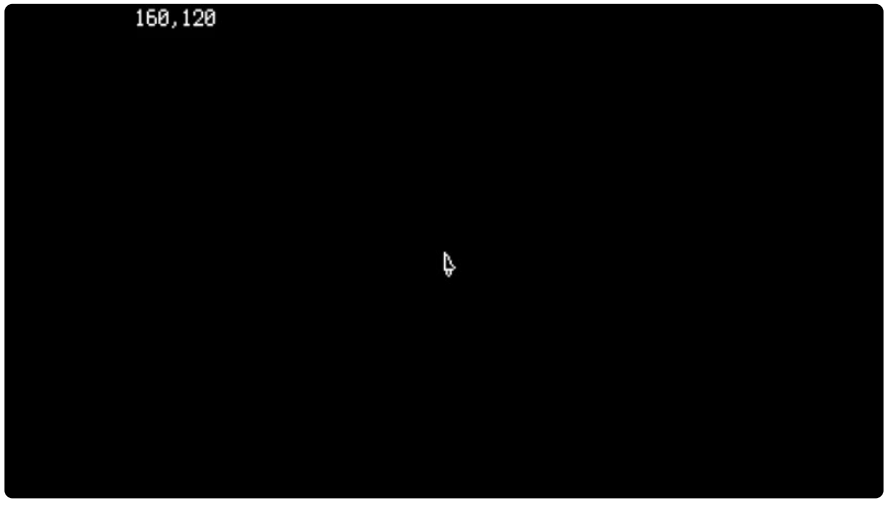
GND to **GND** with **Black** or **Blue**
D+ to **D+** with **Green**
D- to **D-** with **White**
5V to **5V** with **Red**

Note that the data pins are swapped on the Metro compared to the CH334F breakout. On the Metro **D-** is next to **5V**, whereas on the CH334F **D-** is next to **GND**.

Be sure to connect **D-** on the breakout to **D-** on the Metro, and **D+** on the breakout to **D+** on the Metro.

USB Host is under active development. As of CircuitPython version 10.0.0-alpha.2, the wired USB Mouse only works on CircuitPython when connected through a USB hub such as the CH334F.

Demo Code: Displayio Mouse Cursor



```
# SPDX-FileCopyrightText: 2025 Tim Cocks for Adafruit Industries
# SPDX-License-Identifier: MIT
"""
This example is made for a basic boot mouse with
two buttons and a wheel that can be pressed.

It assumes there is a single mouse connected to USB Host,
and no other devices connected.
"""
import array
from displayio import Group, OnDiskBitmap, TileGrid
import supervisor
import terminalio
import usb.core
from adafruit_display_text.bitmap_label import Label
import adafruit_usb_host_descriptors

display = supervisor.runtime.display

# group to hold visual elements
main_group = Group()

# make the group visible on the display
display.root_group = main_group

# load the mouse cursor bitmap
mouse_bmp = OnDiskBitmap("mouse_cursor.bmp")

# make the background pink pixels transparent
mouse_bmp.pixel_shader.make_transparent(0)

# create a TileGrid for the mouse, using its bitmap and pixel_shader
mouse_tg = TileGrid(mouse_bmp, pixel_shader=mouse_bmp.pixel_shader)

# move it to the center of the display
mouse_tg.x = display.width // 2
mouse_tg.y = display.height // 2

# text label to show the x, y coordinates on the screen
output_lbl = Label(
    terminalio.FONT, text=f"{mouse_tg.x},{mouse_tg.y}", color=0xFFFFFF, scale=1
)

# move it to the upper left corner
output_lbl.anchor_point = (0, 0)
output_lbl.anchored_position = (1, 1)

# add it to the main group
main_group.append(output_lbl)

# add the mouse tile grid to the main group
main_group.append(mouse_tg)

# button names
# This is ordered by bit position.
BUTTONS = ["left", "right", "middle"]

# scan for connected USB device and loop over any found
for device in usb.core.find(find_all=True):
    # print device info
    print(f"{device.idVendor:04x}:{device.idProduct:04x}")
    print(device.manufacturer, device.product)
    print(device.serial_number)

    # try to find mouse endpoint on the current device.
    mouse_interface_index, mouse_endpoint_address = (
        adafruit_usb_host_descriptors.find_boot_mouse_endpoint(device)
    )
```

```
    if mouse_interface_index is not None and mouse_endpoint_address is not None:
        mouse = device
        print(
            f"mouse interface: {mouse_interface_index} "
            + f"endpoint_address: {hex(mouse_endpoint_address)}"
        )

        # detach the kernel driver if needed
        if mouse.is_kernel_driver_active(0):
            mouse.detach_kernel_driver(0)

        # set configuration on the mouse so we can use it
        mouse.set_configuration()

        break

# buffer to hold mouse data
buf = array.array("b", [0] * 8)

# main loop
while True:
    try:
        # attempt to read data from the mouse
        # 20ms timeout, so we don't block long if there
        # is no data
        count = mouse.read(0x81, buf, timeout=20)
    except usb.core.USBTimeoutError:
        # skip the rest of the loop if there is no data
        continue

    # update the mouse tilegrid x and y coordinates
    # based on the delta values read from the mouse
    mouse_tg.x = max(0, min(display.width - 1, mouse_tg.x + buf[1]))
    mouse_tg.y = max(0, min(display.height - 1, mouse_tg.y + buf[2]))

    # string with updated coordinates for the text label
    out_str = f"{mouse_tg.x},{mouse_tg.y}"

    # loop over the button names
    for i, button in enumerate(BUTTONS):
        # check if each button is pressed using bitwise AND shifted
        # to the appropriate index for this button
        if buf[0] & (1 << i) != 0:
            # append the button name to the string to show if
            # it is being clicked.
            out_str += f" {button}"

    # update the text label with the new coordinates
    # and buttons being pressed
    output_lbl.text = out_str
```

Code Explanation

The example code contains comments for the line or section of code that details its purpose. Read the comments along with a summary below to understand how the demo works.

This demo scans for and reads data from the USB mouse in the same way as the basic demo on the previous page. The code uses the default built-in display with `supervisor.runtime.display`. For the Metro RP2350, that is the HSTX / DVI connected display.

Visual Elements Setup

The code creates a displayio `main_group` to put all visual elements into and sets it as the `root_group` on the display so it is shown on the screen. Then it creates an `OnDiskBitmap` to load the **mouse_cursor.bmp** file.

This files contains a pink color in the palette index 0 which is treated as transparency by calling `mouse_bmp.pixel_shader.make_transparent(0)`. A TileGrid is created and stored in the variable `mouse_tg`. Later, when reading mouse data, the program will use the data to move `mouse_tg` around the screen. The mouse cursor is put into the center of the screen to start with.

A `Label` named `output_lbl` is created and added to the `main_group` after being placed in the top left corner of the screen. This will be used to show the current mouse coordinates and any buttons that are pressed.

The `mouse_tg` is added to `main_group` last so that it will be visually in front of everything else.

Main Loop

Inside the main loop, `mouse.read(0x81, buf, timeout=20)` is called to read data from the mouse. `0x81` is the default endpoint address for basic HID mice, `buf` is the 8-byte buffer array that will get filled with the data that is read. `timeout` is how many milliseconds to wait before raising a `USBTimeoutError` if there is no data to read. This code uses a low value of `20` milliseconds to illustrate how the main loop can do other things if the timeout is kept low. Higher timeout values result in the `read()` call blocking other code execution for longer times.

If there is no data, the `USBTimeoutError` is raised, and the code skips to the next iteration with `continue`.

If there is data, the code reads the delta x and y values from buffer indexes `1` and `2`. These delta values represent how far the mouse has moved in each direction. It will have negative values for up/left, and positive values for right/down.

The delta values are used to move the `mouse_tg` to a new location. `min()` and `max()` are used to clamp the mouse cursor to the bounds of the display. The mouse itself is not aware of these bounds, the code enforces staying on the display after reading raw data from the mouse.

The `out_str` is updated with the current x and y coordinates of the `mouse_tg`.

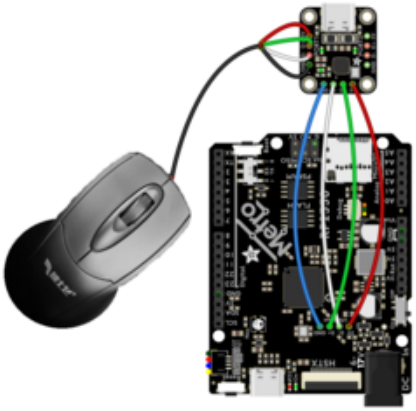
Next, the code checks for button presses by looping over the `BUTTONS` list and checking the bits in the respective positions within the byte at an index `0` of the buffer. Any buttons that are pressed have their name added to the `out_str`.

Finally `out_str` is set as the text on the `output_lbl` to show the current values on the screen.

Arduino

To read data from a mouse in Arduino, you must use a device which supports USB Host such as the Metro RP2350 or Feather RP2040 USB Host. Right now the Metro RP2350 only works when the wired mouse is connected through a USB hub such as the CH334F.

Feather RP2040 USB Host Wiring
Simply plug the mouse into the USB Host port on the Feather RP2040 USB Host



Metro RP2350 Wiring

Connecting to the Metro RP2350 USB Host port requires soldering pins to the broken out USB Host connections as shown in this guide page (https://adafru.it/1ahU). Make the following connections between the Metro USB Host pins and CH334F host connection opposite the USB C connector.

GND to **GND** with **Black** or **Blue**
D+ to **D+** with **Green**
D- to **D-** with **White**
5V to **5V** with **Red**

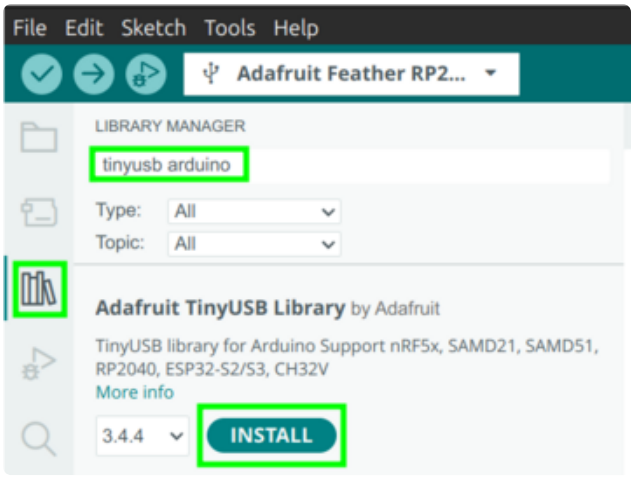
Note that the data pins are swapped on the Metro compared to the CH334F breakout. On the Metro **D-** is next to **5V**, whereas on the CH334F **D-** is next to **GND**.

Be sure to connect **D-** on the breakout to **D-** on the Metro, and **D+** on the breakout to **D+** on the Metro.

USB Host is under active development. As of Pico-PIO-USB version 0.7.1 and TinyUSB version 3.4.4, the wired USB Mouse only works on the Metro RP2350 with Arduino when connected through a USB hub such as the CH334F.

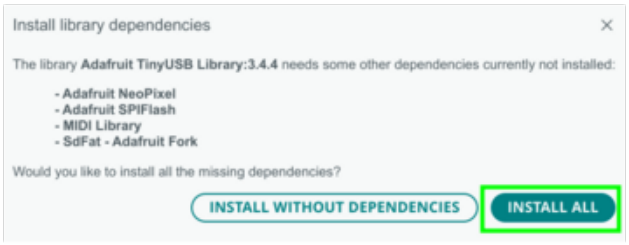
Install the Libraries

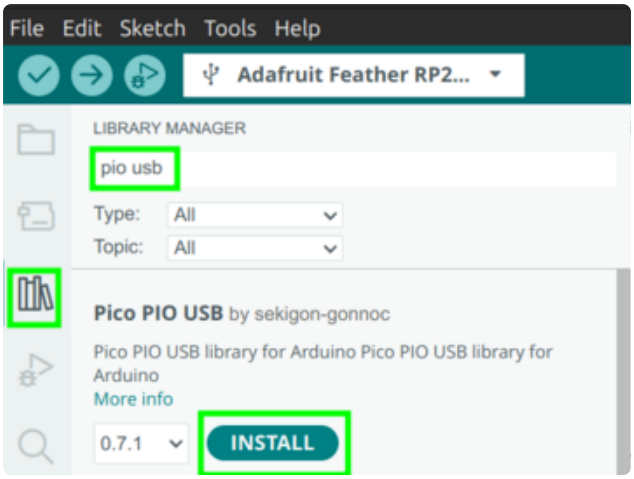
You can install the libraries for this project using the Library Manager in the Arduino IDE.

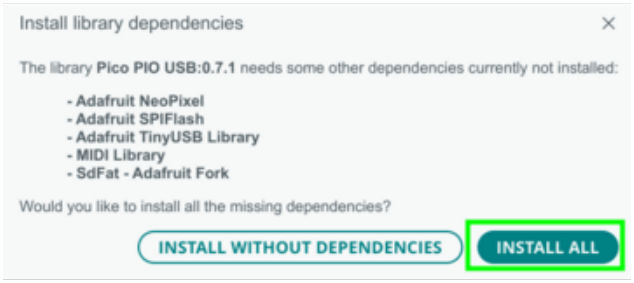


Click the Library Manager icon, search for Adafruit TinyUSB Arduino, and select the Adafruit TinyUSB Library.

If asked about dependencies click "Install All".







Then install the Pico PIO USB library. Click the **Library Manager** icon menu item again, search for **PIO USB**, and select the **Pico PIO USB** library by sekigon-gonnoc.

Code Prep

The code consists of a main **.ino** program file and two header files. The header files store configuration for USB host on the RP2040/RP2350 and HID device reports for the mouse. You'll need all three of these files to properly compile and run the project.

usbhost_mouse_simpletest.ino

```
// SPDX-FileCopyrightText: 2025 Tim Cocks for Adafruit Industries
//
// SPDX-License-Identifier: MIT

/*********************************************************************
 Adafruit invests time and resources providing this open source code,
 please support Adafruit and open-source hardware by purchasing
 products from Adafruit!

 MIT license, check LICENSE for more information
 Copyright (c) 2019 Ha Thach for Adafruit Industries
 All text above, and the splash screen below must be included in
 any redistribution
*********************************************************************/

/* This example demonstrates use of usb host with a standard HID boot mouse.
 * - Host depends on MCU:
 *   - rp2040: bit-banging 2 GPIOs with Pico-PIO-USB library (roothub port1)
 *
 * Requirements:
 * - For rp2040:
 *   - Pico-PIO-USB library
```

```
 *   - 2 consecutive GPIOs: D+ is defined by PIN_USB_HOST_DP, D- = D+ +1
 *   - Provide VBus (5v) and GND for peripheral
 */

// USBHost is defined in usbh_helper.h
#include "usbh_helper.h"
#include "tusb.h"
#include "Adafruit_TinyUSB.h"
#include "hid_mouse_reports.h"


bool printed_blank = false;

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

  // configure pio-usb: defined in usbh_helper.h
  rp2040_configure_pio_usb();

  // run host stack on controller (rhport) 1
  // Note: For rp2040 pico-pio-usb, calling USBHost.begin() on core1 will have most
of the
  // host bit-banging processing works done in core1 to free up core0 for other
works
  USBHost.begin(1);
  delay(3000);
  Serial.print("USB D+ Pin:");
  Serial.println(PIN_USB_HOST_DP);
  Serial.print("USB 5V Pin:");
  Serial.println(PIN_5V_EN);
}

void loop() {
  USBHost.task();
  Serial.flush();

}

//--------------------------------------------------------------------+
// HID Host Callback Functions
//--------------------------------------------------------------------+

void tuh_hid_mount_cb(uint8_t dev_addr, uint8_t instance, uint8_t const*
desc_report, uint16_t desc_len)
{
  Serial.printf("HID device mounted (address %d, instance %d)\n", dev_addr,
instance);

  // Start receiving HID reports
  if (!tuh_hid_receive_report(dev_addr, instance))
  {
    Serial.printf("Error: cannot request to receive report\n");
  }
}

void tuh_hid_umount_cb(uint8_t dev_addr, uint8_t instance)
{
  Serial.printf("HID device unmounted (address %d, instance %d)\n", dev_addr,
instance);
}

void tuh_hid_report_received_cb(uint8_t dev_addr, uint8_t instance, uint8_t const*
report, uint16_t len) {

      if (len > 0){

        //debug print report data
        // Serial.print("Report data: ");
        // for (int i = 0; i < len; i++) {
```

```
        //   if (i == 0 || i == 3){
        //     Serial.print(report[i], HEX);
        //     Serial.print(" ");
        //   }else { // i==1 or i==2
        //     Serial.print((int8_t)report[i]);
        //     Serial.print(" ");
        //   }
        // }
        // Serial.println();

        Serial.print("X: ");
        Serial.print((int8_t)report[1]);
        Serial.print(" ");
        Serial.print("Y: ");
        Serial.print((int8_t)report[2]);
        Serial.print(" ");

        if (report[BYTE_BUTTONS] != BUTTON_NEUTRAL){
          if ((report[BYTE_BUTTONS] & BUTTON_LEFT) == BUTTON_LEFT){
            Serial.print("Left ");
          }
          if ((report[BYTE_BUTTONS] & BUTTON_RIGHT) == BUTTON_RIGHT){
            Serial.print("Right ");
          }
          if ((report[BYTE_BUTTONS] & BUTTON_MIDDLE) == BUTTON_MIDDLE){
            Serial.print("Middle ");
          }
        }
        Serial.println();
      }

  // Continue to receive the next report
  if (!tuh_hid_receive_report(dev_addr, instance)) {
    Serial.println("Error: cannot request to receive report");
  }
}
```

usbh_helper.h

```
// SPDX-FileCopyrightText: 2024 Ha Thach for Adafruit Industries
//
// SPDX-License-Identifier: MIT

/*********************************************************************
 Adafruit invests time and resources providing this open source code,
 please support Adafruit and open-source hardware by purchasing
 products from Adafruit!

 MIT license, check LICENSE for more information
 Copyright (c) 2019 Ha Thach for Adafruit Industries
 All text above, and the splash screen below must be included in
 any redistribution
*********************************************************************/

#ifndef USBH_HELPER_H
#define USBH_HELPER_H

#ifdef ARDUINO_ARCH_RP2040
  // pio-usb is required for rp2040 host
  #include "pio_usb.h"

  // Pin D+ for host, D- = D+ + 1
  #ifndef PIN_USB_HOST_DP
  #define PIN_USB_HOST_DP  16
  #endif

  // Pin for enabling Host VBUS. comment out if not used
```

```
  #ifndef PIN_5V_EN
  #define PIN_5V_EN        18
  #endif

  #ifndef PIN_5V_EN_STATE
  #define PIN_5V_EN_STATE  1
  #endif
#endif // ARDUINO_ARCH_RP2040

#ifdef ARDUINO_ARCH_RP2350

  // pio-usb is required for rp2040 host
  #include "pio_usb.h"

  // Pin D+ for host, D- = D+ + 1
  #ifndef PIN_USB_HOST_DP
  #define PIN_USB_HOST_DP  32
  #endif

  // Pin for enabling Host VBUS. comment out if not used
  #ifndef PIN_5V_EN
  #define PIN_5V_EN        29
  #endif

  #ifndef PIN_5V_EN_STATE
  #define PIN_5V_EN_STATE  1
  #endif
#endif // ARDUINO_ARCH_RP2350

#include "Adafruit_TinyUSB.h"

#if defined(CFG_TUH_MAX3421) && CFG_TUH_MAX3421
  // USB Host using MAX3421E: SPI, CS, INT
  #include "SPI.h"

  #if defined(ARDUINO_METRO_ESP32S2)
    Adafruit_USBH_Host USBHost(&SPI, 15, 14);
  #elif defined(ARDUINO_ADAFRUIT_FEATHER_ESP32_V2)
    Adafruit_USBH_Host USBHost(&SPI, 33, 15);
  #else
    // Default CS and INT are pin 10, 9
    Adafruit_USBH_Host USBHost(&SPI, 10, 9);
  #endif
#else
  // Native USB Host such as rp2040
  Adafruit_USBH_Host USBHost;
#endif

//--------------------------------------------------------------------+
// Helper Functions
//--------------------------------------------------------------------+

#ifdef ARDUINO_ARCH_RP2040
static void rp2040_configure_pio_usb(void) {
  //while ( !Serial ) delay(10);   // wait for native usb
  Serial.println("Core1 setup to run TinyUSB host with pio-usb");

#ifdef PIN_5V_EN
  pinMode(PIN_5V_EN, OUTPUT);
  digitalWrite(PIN_5V_EN, PIN_5V_EN_STATE);
#endif

  pio_usb_configuration_t pio_cfg = PIO_USB_DEFAULT_CONFIG;
  pio_cfg.pin_dp = PIN_USB_HOST_DP;

#if defined(ARDUINO_RASPBERRY_PI_PICO_W)
  // For pico-w, PIO is also used to communicate with cyw43
  // Therefore we need to alternate the pio-usb configuration
  // details https://github.com/sekigon-gonnoc/Pico-PIO-USB/issues/46
```

```
  pio_cfg.sm_tx      = 3;
  pio_cfg.sm_rx      = 2;
  pio_cfg.sm_eop     = 3;
  pio_cfg.pio_rx_num = 0;
  pio_cfg.pio_tx_num = 1;
  pio_cfg.tx_ch      = 9;
#endif

  USBHost.configure_pio_usb(1, &pio_cfg);
}
#endif

#endif
```

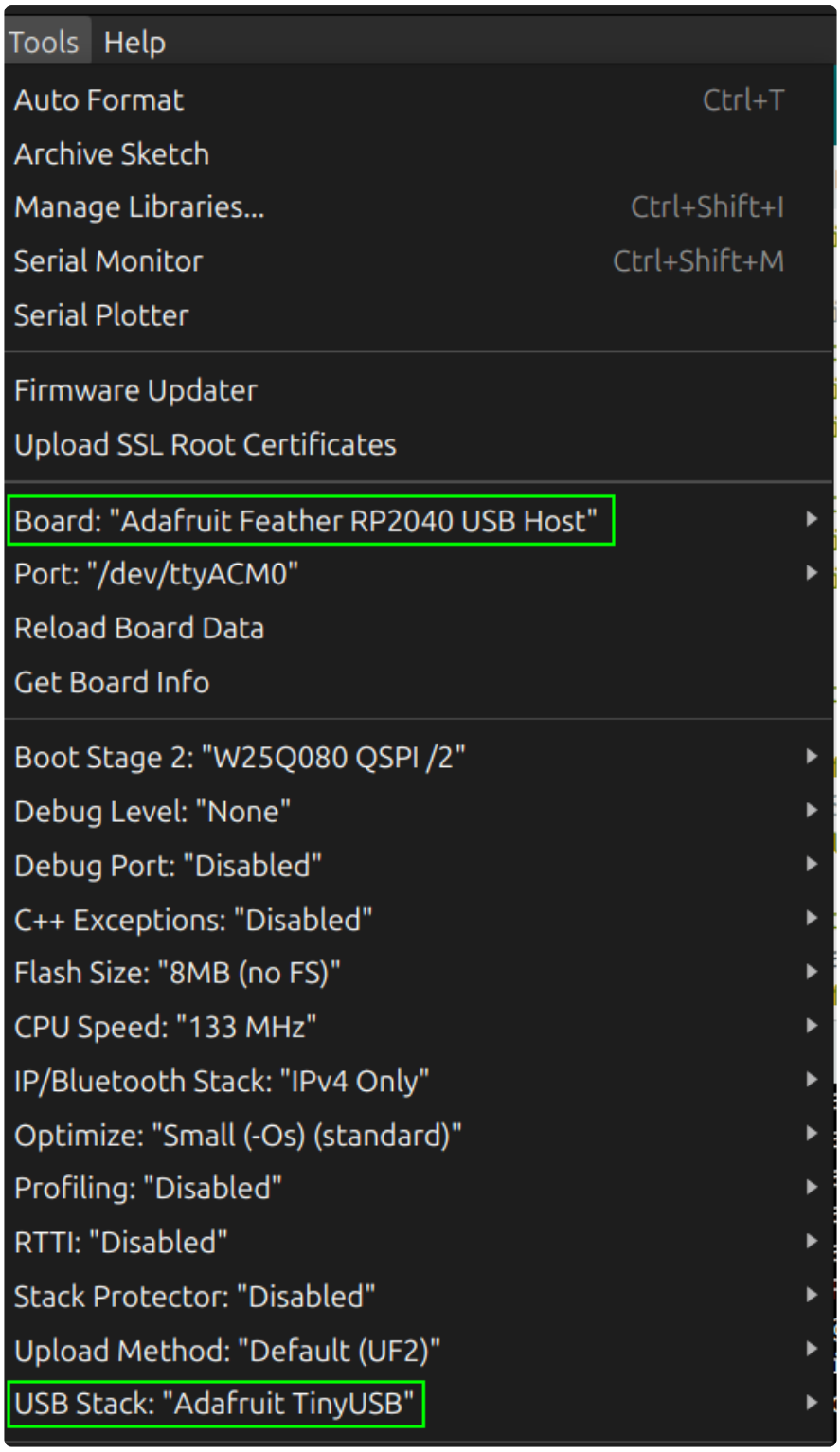
hid_mouse_reports.h

```
// SPDX-FileCopyrightText: 2025 Tim Cocks for Adafruit Industries
//
// SPDX-License-Identifier: MIT

// HID reports for standard boot mouse

// Byte indices for the gamepad report
#define BYTE_BUTTONS    0   // Left, right, middle click buttons
#define BYTE_DELTA_X    1   // Mouse movement horizontal
#define BYTE_DELTA_Y    2   // Mouse movement vertical

#define BUTTON_NEUTRAL      0x00
#define BUTTON_LEFT         0x01
#define BUTTON_RIGHT        0x02
#define BUTTON_MIDDLE       0x04
```

Upload and Test

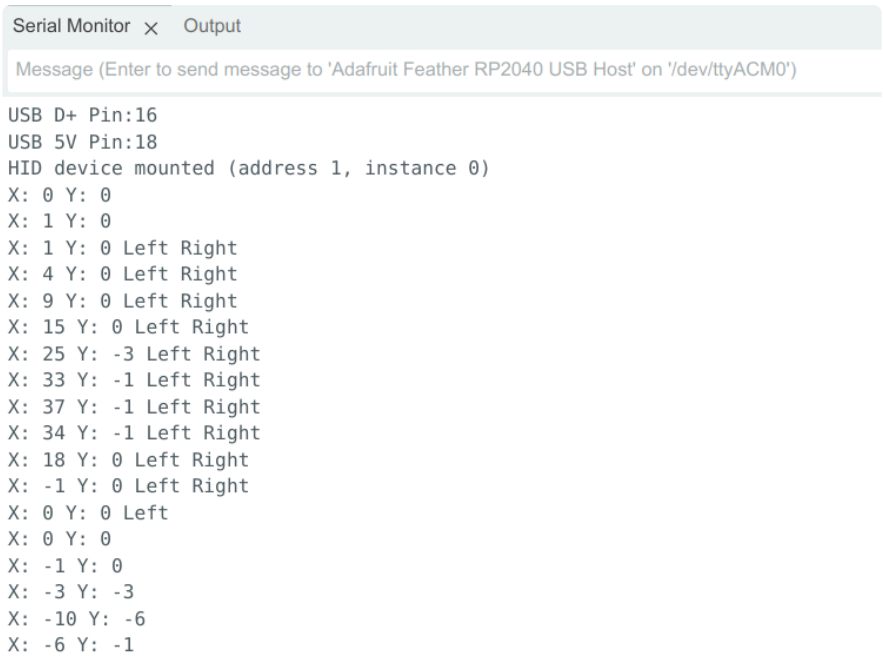
Before uploading, you'll need to update some settings in the Boards menu. Select the appropriate board that you are using, either **Adafruit Feather RP2040 USB Host**, or **Adafruit Metro RP2350**. Under **USB Stack** select **Adafruit TinyUSB**.

Then, upload the sketch to your board. You can use the Serial Monitor in the Arduino IDE for debugging any errors.



Serial Output

The code will connect to the USB mouse and read data coming from it. Delta X and Y values are printed to the serial output, when it detects that buttons are pressed it will print which ones are down into the serial output as well..


Serial Monitor
Output
Message (Enter to send message to 'Adafruit Feather RP2040 USB Host' on '/dev/ttyACM0')
USB D+ Pin:16
USB 5V Pin:18
HID device mounted (address 1, instance 0)
X: 0 Y: 0
X: 1 Y: 0
X: 1 Y: 0 Left Right
X: 4 Y: 0 Left Right
X: 9 Y: 0 Left Right
X: 15 Y: 0 Left Right
X: 25 Y: -3 Left Right
X: 33 Y: -1 Left Right
X: 37 Y: -1 Left Right
X: 34 Y: -1 Left Right
X: 18 Y: 0 Left Right
X: -1 Y: 0 Left Right
X: 0 Y: 0 Left
X: 0 Y: 0
X: -1 Y: 0
X: -3 Y: -3
X: -10 Y: -6
X: -6 Y: -1