Adafruit HID Library Documentation

Release 1.0

Scott Shawcroft

Jun 21, 2019

Contents

This driver simulates USB HID devices. Currently keyboard and mouse are implemented.

CHAPTER 1

Dependencies

This driver depends on:

- Adafruit CircuitPython

Please ensure all dependencies are available on the CircuitPython filesystem. This is easily achieved by downloading the Adafruit library and driver bundle.

CHAPTER 2

Usage Example

The `Keyboard` class sends keypress reports for a USB keyboard device to the host.

The `Keycode` class defines USB HID keycodes to send using `Keyboard`.

```
from adafruit_hid.keyboard import Keyboard
from adafruit_hid.keycode import Keycode

# Set up a keyboard device.
kbd = Keyboard()

# Type lowercase 'a'. Presses the 'a' key and releases it.
kbd.send(Keycode.A)

# Type capital 'A'.
kbd.send(Keycode.SHIFT, Keycode.A)

# Type control-x.
kbd.send(Keycode.CONTROL, Keycode.X)

# You can also control press and release actions separately.
kbd.press(Keycode.CONTROL, Keycode.X)
kbd.release_all()

# Press and hold the shifted '1' key to get '!' (exclamation mark).
kbd.press(Keycode.SHIFT, Keycode.ONE)
# Release the ONE key and send another report.
kbd.release(Keycode.ONE)
# Press shifted '2' to get '@'.
kbd.press(Keycode.TWO)
# Release all keys.
kbd.release_all()
```

The `KeyboardLayoutUS` sends ASCII characters using keypresses. It assumes the host is set to accept keypresses from a US keyboard.

If the host is expecting a non-US keyboard, the character to key mapping provided by `KeyboardLayoutUS` will

not always be correct. Different keypresses will be needed in some cases. For instance, to type an `'A'` on a French keyboard (AZERTY instead of QWERTY), `Keycode.Q` should be pressed.

Currently this package provides only `KeyboardLayoutUS`. More `KeyboardLayout` classes could be added to handle non-US keyboards and the different input methods provided by various operating systems.

```
from adafruit_hid.keyboard import Keyboard
from adafruit_hid.keyboard_layout_us import KeyboardLayoutUS

kbd = Keyboard()
layout = KeyboardLayoutUS(kbd)

# Type 'abc' followed by Enter (a newline).
layout.write('abc\n')

# Get the keycodes needed to type a '$'.
# The method will return (Keycode.SHIFT, Keycode.FOUR).
keycodes = layout.keycodes('$')
```

The `Mouse` class simulates a three-button mouse with a scroll wheel.

```
from adafruit_hid.mouse import Mouse

m = Mouse()

# Click the left mouse button.
m.click(Mouse.LEFT_BUTTON)

# Move the mouse diagonally to the upper left.
m.move(-100, -100, 0)

# Roll the mouse wheel away from the user one unit.
# Amount scrolled depends on the host.
m.move(0, 0, -1)

# Keyword arguments may also be used. Omitted arguments default to 0.
m.move(x=-100, y=-100)
m.move(wheel=-1)

# Move the mouse while holding down the left button. (click-drag).
m.press(Mouse.LEFT_BUTTON)
m.move(x=50, y=20)
m.release_all()        # or m.release(Mouse.LEFT_BUTTON)
```

The `ConsumerControl` class emulates consumer control devices such as remote controls, or the multimedia keys on certain keyboards.

New in CircuitPython 3.0.

```
from adafruit_hid.consumer_control import ConsumerControl
from adafruit_hid.consumer_control_code import ConsumerControlCode

cc = ConsumerControl()

# Raise volume.
cc.send(ConsumerControlCode.VOLUME_INCREMENT)

# Pause or resume playback.
cc.send(ConsumerControlCode.PLAY_PAUSE)
```

The `Gamepad` class emulates a two-joystick gamepad with 16 buttons.

New in CircuitPython 3.0.

```
from adafruit_hid.gamepad import Gamepad

gp = Gamepad()

# Click gamepad buttons.
gp.click_buttons(1, 7)

# Move joysticks.
gp.move_joysticks(x=2, y=0, z=-20)
```

CHAPTER 3

Contributing

Contributions are welcome! Please read our Code of Conduct before contributing to help this project stay welcoming.

CHAPTER 4

Building locally

To build this library locally you'll need to install the circuitpython-build-tools package.

```
python3 -m venv .env
source .env/bin/activate
pip install circuitpython-build-tools
```

Once installed, make sure you are in the virtual environment:

```
source .env/bin/activate
```

Then run the build:

```
circuitpython-build-bundles --filename_prefix adafruit-circuitpython-hid --library_
↪location .
```

4.1 Sphinx documentation

Sphinx is used to build the documentation based on rST files and comments in the code. First, install dependencies (feel free to reuse the virtual environment from above):

```
python3 -m venv .env
source .env/bin/activate
pip install Sphinx sphinx-rtd-theme
```

Now, once you have the virtual environment activated:

```
cd docs
sphinx-build -E -W -b html . _build/html
```

This will output the documentation to `docs/_build/html`. Open the index.html in your browser to view them. It will also (due to -W) error out on any warning like Travis will. This is a good way to locally verify it will pass.

CHAPTER 5

Table of Contents

5.1 Simple test

Ensure your device works with this simple test.

Listing 1: examples/hid_keyboard_shortcuts.py

```
import time
import board
import digitalio
from adafruit_hid.keyboard import Keyboard
from adafruit_hid.keycode import Keycode

kbd = Keyboard()

# define buttons. these can be any physical switches/buttons, but the values
# here work out-of-the-box with a CircuitPlayground Express' A and B buttons.
swap = digitalio.DigitalInOut(board.D4)
swap.direction = digitalio.Direction.INPUT
swap.pull = digitalio.Pull.DOWN

search = digitalio.DigitalInOut(board.D5)
search.direction = digitalio.Direction.INPUT
search.pull = digitalio.Pull.DOWN

while True:
    # press ALT+TAB to swap windows
    if swap.value:
        kbd.send(Keycode.ALT, Keycode.TAB)

    # press CTRL+K, which in a web browser will open the search dialog
    elif search.value:
        kbd.send(Keycode.CONTROL, Keycode.K)

```

(continues on next page)

(continued from previous page)

```
    time.sleep(0.1)
```

Listing 2: examples/hid_simpletest.py

```
import time
import board
import digitalio
from adafruit_hid.mouse import Mouse

mouse = Mouse()

# define buttons. these can be any physical switches/buttons, but the values
# here work out-of-the-box with a CircuitPlayground Express' A and B buttons.
up = digitalio.DigitalInOut(board.D4)
up.direction = digitalio.Direction.INPUT
up.pull = digitalio.Pull.DOWN

down = digitalio.DigitalInOut(board.D5)
down.direction = digitalio.Direction.INPUT
down.pull = digitalio.Pull.DOWN

while True:
    # scroll up one unit (varies with host/OS)
    if up.value:
        mouse.move(wheel=1)

    # scroll down one unit (varies with host/OS)
    elif down.value:
        mouse.move(wheel=-1)

    time.sleep(0.1)
```

Listing 3: examples/hid_simple_gamepad.py

```
import board
import digitalio
import analogio

from adafruit_hid.gamepad import Gamepad

gp = Gamepad()

# Create some buttons. The physical buttons are connected
# to ground on one side and these and these pins on the other.
button_pins = (board.D2, board.D3, board.D4, board.D5)

# Map the buttons to button numbers on the Gamepad.
# gamepad_buttons[i] will send that button number when buttons[i]
# is pushed.
gamepad_buttons = (1, 2, 8, 15)

buttons = [digitalio.DigitalInOut(pin) for pin in button_pins]
for button in buttons:
    button.direction = digitalio.Direction.INPUT
    button.pull = digitalio.Pull.UP

```

(continues on next page)

(continued from previous page)

```
# Connect an analog two-axis joystick to A4 and A5.
ax = analogio.AnalogIn(board.A4)
ay = analogio.AnalogIn(board.A5)

# Equivalent of Arduino's map() function.
def range_map(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) // (in_max - in_min) + out_min

while True:
    # Buttons are grounded when pressed (.value = False).
    for i, button in enumerate(buttons):
        gamepad_button_num = gamepad_buttons[i]
        if button.value:
            gp.release_buttons(gamepad_button_num)
            print(" release", gamepad_button_num, end='')
        else:
            gp.press_buttons(gamepad_button_num)
            print(" press", gamepad_button_num, end='')

    # Convert range[0, 65535] to -127 to 127
    gp.move_joysticks(x=range_map(ax.value, 0, 65535, -127, 127),
                      y=range_map(ay.value, 0, 65535, -127, 127))
    print(" x", ax.value, "y", ay.value)
```

Listing 4: examples/hid_joywing_gamepad.py

```
# Use Joy FeatherWing to drive Gamepad.

import time

import board
import busio
from micropython import const
import adafruit_seesaw
from adafruit_hid.gamepad import Gamepad

def range_map(value, in_min, in_max, out_min, out_max):
    return (value - in_min) * (out_max - out_min) // (in_max - in_min) + out_min

BUTTON_RIGHT = const(6)
BUTTON_DOWN = const(7)
BUTTON_LEFT = const(9)
BUTTON_UP = const(10)
BUTTON_SEL = const(14)
button_mask = const((1 << BUTTON_RIGHT) |
                    (1 << BUTTON_DOWN) |
                    (1 << BUTTON_LEFT) |
                    (1 << BUTTON_UP) |
                    (1 << BUTTON_SEL))

i2c = busio.I2C(board.SCL, board.SDA)

ss = adafruit_seesaw.Seesaw(i2c)

ss.pin_mode_bulk(button_mask, ss.INPUT_PULLUP)

```

(continues on next page)

(continued from previous page)

```
last_game_x = 0
last_game_y = 0

g = Gamepad()

while True:
    x = ss.analog_read(2)
    y = ss.analog_read(3)

    game_x = range_map(x, 0, 1023, -127, 127)
    game_y = range_map(y, 0, 1023, -127, 127)
    if last_game_x != game_x or last_game_y != game_y:
        last_game_x = game_x
        last_game_y = game_y
        print(game_x, game_y)
        g.move_joysticks(x=game_x, y=game_y)

    buttons = (BUTTON_RIGHT, BUTTON_DOWN, BUTTON_LEFT, BUTTON_UP, BUTTON_SEL)
    button_state = [False] * len(buttons)
    for i, button in enumerate(buttons):
        buttons = ss.digital_read_bulk(button_mask)
        if not (buttons & (1 << button) and not button_state[i]):
            g.press_buttons(i+1)
            print("Press", i+1)
            button_state[i] = True
        elif button_state[i]:
            g.release_buttons(i+1)
            print("Release", i+1)
            button_state[i] = False

    time.sleep(.01)
```

5.2 adafruit_hid.keyboard.Keyboard

- Author(s): Scott Shawcroft, Dan Halbert

class adafruit_hid.keyboard.**Keyboard**

Send HID keyboard reports.

press(**keycodes*)

Send a report indicating that the given keys have been pressed.

Parameters **keycodes** – Press these keycodes all at once.

Raises **ValueError** – if more than six regular keys are pressed.

Keycodes may be modifiers or regular keys. No more than six regular keys may be pressed simultaneously.

Examples:

```
from adafruit_hid.keycode import Keycode

# Press ctrl-x.
kbd.press(Keycode.LEFT_CONTROL, Keycode.X)

# Or, more conveniently, use the CONTROL alias for LEFT_CONTROL:
```

(continues on next page)

(continued from previous page)

```
kbd.press(Keycode.CONTROL, Keycode.X)

# Press a, b, c keys all at once.
kbd.press(Keycode.A, Keycode.B, Keycode.C)
```

release(**keycodes*)
Send a USB HID report indicating that the given keys have been released.

Parameters **keycodes** – Release these keycodes all at once.

If a keycode to be released was not pressed, it is ignored.

Example:

```
# release SHIFT key
kbd.release(Keycode.SHIFT)
```

release_all()
Release all pressed keys.

send(**keycodes*)
Press the given keycodes and then release all pressed keys.

Parameters **keycodes** – keycodes to send together

5.3 adafruit_hid.keycode.Keycode

- Author(s): Scott Shawcroft, Dan Halbert

class adafruit_hid.keycode.**Keycode**
USB HID Keycode constants.

This list is modeled after the names for USB keycodes defined in https://www.usb.org/sites/default/files/documents/hut1_12v2.pdf#page=53. This list does not include every single code, but does include all the keys on a regular PC or Mac keyboard.

Remember that keycodes are the names for key *positions* on a US keyboard, and may not correspond to the character that you mean to send if you want to emulate non-US keyboard. For instance, on a French keyboard (AZERTY instead of QWERTY), the keycode for 'q' is used to indicate an 'a'. Likewise, 'y' represents 'z' on a German keyboard. This is historical: the idea was that the keycaps could be changed without changing the keycodes sent, so that different firmware was not needed for different variations of a keyboard.

A = 4
`a` and `A`

ALT = 226
Alias for LEFT_ALT; Alt is also known as Option (Mac)

APPLICATION = 101
Application: also known as the Menu key (Windows)

B = 5
`b` and `B`

BACKSLASH = 49
\ and |

BACKSPACE = 42
Delete backward (Backspace)

C = 6
c and C

CAPS_LOCK = 57
Caps Lock

COMMA = 54
, and <

COMMAND = 227
Labeled as Command on Mac keyboards, with a clover glyph

CONTROL = 224
Alias for LEFT_CONTROL

D = 7
d and D

DELETE = 76
Delete forward

DOWN_ARROW = 81
Move the cursor down

E = 8
e and E

EIGHT = 37
8 and *

END = 77
End (often moves to end of line)

ENTER = 40
Enter (Return)

EQUALS = 46
=` and ``+

ESCAPE = 41
Escape

F = 9
f and F

F1 = 58
Function key F1

F10 = 67
Function key F10

F11 = 68
Function key F11

F12 = 69
Function key F12

F13 = 104
Function key F13 (Mac)

F14 = 105
Function key F14 (Mac)

F15 = 106
Function key F15 (Mac)

F16 = 107
Function key F16 (Mac)

F17 = 108
Function key F17 (Mac)

F18 = 109
Function key F18 (Mac)

F19 = 110
Function key F19 (Mac)

F2 = 59
Function key F2

F3 = 60
Function key F3

F4 = 61
Function key F4

F5 = 62
Function key F5

F6 = 63
Function key F6

F7 = 64
Function key F7

F8 = 65
Function key F8

F9 = 66
Function key F9

FIVE = 34
`5` and `%`

FORWARD_SLASH = 56
`/` and `?`

FOUR = 33
`4` and `$`

G = 10
`g` and `G`

GRAVE_ACCENT = 53
`` ` `` and ~

GUI = 227
Alias for LEFT_GUI; GUI is also known as the Windows key, Command (Mac), or Meta

H = 11
`h` and `H`

HOME = 74
Home (often moves to beginning of line)

I = 12
i and I

INSERT = 73
Insert

J = 13
j and J

K = 14
k and K

KEYPAD_ASTERISK = 85
Keypad *

KEYPAD_BACKSLASH = 100
Keypad \ and | (Non-US)

KEYPAD_EIGHT = 96
Keypad 8 and Up Arrow

KEYPAD_ENTER = 88
Keypad Enter

KEYPAD_EQUALS = 103
Keypad = (Mac)

KEYPAD_FIVE = 93
Keypad 5

KEYPAD_FORWARD_SLASH = 84
Keypad /

KEYPAD_FOUR = 92
Keypad 4 and Left Arrow

KEYPAD_MINUS = 86
Keyapd −

KEYPAD_NINE = 97
Keypad 9 and PgUp

KEYPAD_NUMLOCK = 83
Num Lock (Clear on Mac)

KEYPAD_ONE = 89
Keypad 1 and End

KEYPAD_PERIOD = 99
Keypad . and Del

KEYPAD_PLUS = 87
Keypad +

KEYPAD_SEVEN = 95
Keypad 7 and Home

KEYPAD_SIX = 94
Keypad 6 and Right Arrow

KEYPAD_THREE = 91
Keypad 3 and PgDn

KEYPAD_TWO = 90
Keypad 2 and Down Arrow

KEYPAD_ZERO = 98
Keypad 0 and Ins

L = 15
l and L

LEFT_ALT = 226
Alt modifier left of the spacebar

LEFT_ARROW = 80
Move the cursor left

LEFT_BRACKET = 47
[and {

LEFT_CONTROL = 224
Control modifier left of the spacebar

LEFT_GUI = 227
GUI modifier left of the spacebar

LEFT_SHIFT = 225
Shift modifier left of the spacebar

M = 16
m and M

MINUS = 45
-` and ``_

N = 17
n and N

NINE = 38
9 and (

O = 18
o and O

ONE = 30
1 and !

OPTION = 226
Labeled as Option on some Mac keyboards

P = 19
p and P

PAGE_DOWN = 78
Go forward one page

PAGE_UP = 75
Go back one page

PAUSE = 72
Pause (Break)

PERIOD = 55
. and >

POUND = 50
and ~ (Non-US keyboard)

POWER = 102
Power (Mac)

PRINT_SCREEN = 70
Print Screen (SysRq)

Q = 20
q and Q

QUOTE = 52
' and "

R = 21
r and R

RETURN = 40
Alias for ENTER

RIGHT_ALT = 230
Alt modifier right of the spacebar

RIGHT_ARROW = 79
Move the cursor right

RIGHT_BRACKET = 48
] and }

RIGHT_CONTROL = 228
Control modifier right of the spacebar

RIGHT_GUI = 231
GUI modifier right of the spacebar

RIGHT_SHIFT = 229
Shift modifier right of the spacebar

S = 22
s and S

SCROLL_LOCK = 71
Scroll Lock

SEMICOLON = 51
; and :

SEVEN = 36
7 and &

SHIFT = 225
Alias for LEFT_SHIFT

SIX = 35
6 and ^

SPACE = 44
Alias for SPACEBAR

SPACEBAR = 44
Spacebar

T = 23
t and T

TAB = 43
Tab and Backtab

THREE = 32
3 and #

TWO = 31
2 and @

U = 24
u and U

UP_ARROW = 82
Move the cursor up

V = 25
v and V

W = 26
w and W

WINDOWS = 227
Labeled with a Windows logo on Windows keyboards

X = 27
x and X

Y = 28
y and Y

Z = 29
z and Z

ZERO = 39
0 and)

classmethod modifier_bit(*keycode*)
Return the modifer bit to be set in an HID keycode report if this is a modifier key; otherwise return 0.

5.4 adafruit_hid.keyboard_layout_us.KeyboardLayoutUS

- Author(s): Dan Halbert

class adafruit_hid.keyboard_layout_us.**KeyboardLayoutUS**(*keyboard*)
Map ASCII characters to appropriate keypresses on a standard US PC keyboard.

Non-ASCII characters and most control characters will raise an exception.

keycodes(*char*)
Return a tuple of keycodes needed to type the given character.

Parameters **char** (*str of length one.*) – A single ASCII character in a string.

Returns tuple of Keycode keycodes.

Raises **ValueError** – if char is not ASCII or there is no keycode for it.

Examples:

```
# Returns (Keycode.TAB,)
keycodes('  ')
# Returns (Keycode.A,)
keycode('a')
# Returns (Keycode.SHIFT, Keycode.A)
keycode('A')
# Raises ValueError because it's a accented e and is not ASCII
keycode('é')
```

write(*string*)

Type the string by pressing and releasing keys on my keyboard.

Parameters **string** – A string of ASCII characters.

Raises **ValueError** – if any of the characters are not ASCII or have no keycode (such as some control characters).

Example:

```
# Write abc followed by Enter to the keyboard
layout.write('abc\n')
```

5.5 adafruit_hid.mouse.Mouse

- Author(s): Dan Halbert

class adafruit_hid.mouse.**Mouse**

Send USB HID mouse reports.

LEFT_BUTTON = 1

Left mouse button.

MIDDLE_BUTTON = 4

Middle mouse button.

RIGHT_BUTTON = 2

Right mouse button.

click(*buttons*)

Press and release the given mouse buttons.

Parameters **buttons** – a bitwise-or'd combination of LEFT_BUTTON, MIDDLE_BUTTON, and RIGHT_BUTTON.

Examples:

```
# Click the left button.
m.click(Mouse.LEFT_BUTTON)

# Double-click the left button.
m.click(Mouse.LEFT_BUTTON)
m.click(Mouse.LEFT_BUTTON)
```

move(*x=0*, *y=0*, *wheel=0*)

Move the mouse and turn the wheel as directed.

Parameters

- **x** – Move the mouse along the x axis. Negative is to the left, positive is to the right.

- **y** – Move the mouse along the y axis. Negative is upwards on the display, positive is downwards.
- **wheel** – Rotate the wheel this amount. Negative is toward the user, positive is away from the user. The scrolling effect depends on the host.

Examples:

```
# Move 100 to the left. Do not move up and down. Do not roll the scroll wheel.
m.move(-100, 0, 0)
# Same, with keyword arguments.
m.move(x=-100)

# Move diagonally to the upper right.
m.move(50, 20)
# Same.
m.move(x=50, y=-20)

# Roll the mouse wheel away from the user.
m.move(wheel=1)
```

press(*buttons*)
Press the given mouse buttons.

Parameters **buttons** – a bitwise-or'd combination of LEFT_BUTTON, MIDDLE_BUTTON, and RIGHT_BUTTON.

Examples:

```
# Press the left button.
m.press(Mouse.LEFT_BUTTON)

# Press the left and right buttons simultaneously.
m.press(Mouse.LEFT_BUTTON | Mouse.RIGHT_BUTTON)
```

release(*buttons*)
Release the given mouse buttons.

Parameters **buttons** – a bitwise-or'd combination of LEFT_BUTTON, MIDDLE_BUTTON, and RIGHT_BUTTON.

release_all()
Release all the mouse buttons.

5.6 adafruit_hid.consumer_control.ConsumerControl

- Author(s): Dan Halbert

class adafruit_hid.consumer_control.**ConsumerControl**
Send ConsumerControl code reports, used by multimedia keyboards, remote controls, etc.

New in CircuitPython 3.0.

send(*consumer_code*)
Send a report to do the specified consumer control action, and then stop the action (so it will not repeat).

Parameters **consumer_code** – a 16-bit consumer control code.

Examples:

```
from adafruit_hid.consumer_control_code import ConsumerControlCode

# Raise volume.
consumer_control.send(ConsumerControlCode.VOLUME_INCREMENT)

# Advance to next track (song).
consumer_control.send(ConsumerControlCode.SCAN_NEXT_TRACK)
```

5.7 adafruit_hid.consumer_control_code.ConsumerControlCode

- Author(s): Dan Halbert

class adafruit_hid.consumer_control_code.**ConsumerControlCode**

USB HID Consumer Control Device constants.

This list includes a few common consumer control codes from http://www.usb.org/developers/hidpage/Hut1_12v2.pdf#page=75.

New in CircuitPython 3.0.

EJECT = 184

Eject

FAST_FORWARD = 179

Fast Forward

MUTE = 226

Mute

PLAY_PAUSE = 205

Play/Pause toggle

RECORD = 178

Record

REWIND = 180

Rewind

SCAN_NEXT_TRACK = 181

Skip to next track

SCAN_PREVIOUS_TRACK = 182

Go back to previous track

STOP = 183

Stop

VOLUME_DECREMENT = 234

Decrease volume

VOLUME_INCREMENT = 233

Increase volume

5.8 adafruit_hid.gamepad.Gamepad

- Author(s): Dan Halbert

class adafruit_hid.gamepad.**Gamepad**

Emulate a generic gamepad controller with 16 buttons, numbered 1-16, and two joysticks, one controlling `x` and `y` values, and the other controlling `z` and `r_z` (z rotation or `Rz`) values.

The joystick values could be interpreted differently by the receiving program: those are just the names used here. The joystick values are in the range -127 to 127.

click_buttons(**buttons*)

Press and release the given buttons.

move_joysticks(*x=None*, *y=None*, *z=None*, *r_z=None*)

Set and send the given joystick values. The joysticks will remain set with the given values until changed

One joystick provides `x` and `y` values, and the other provides `z` and `r_z` (z rotation). Any values left as `None` will not be changed.

All values must be in the range -127 to 127 inclusive.

Examples:

```
# Change x and y values only.
gp.move_joysticks(x=100, y=-50)

# Reset all joystick values to center position.
gp.move_joysticks(0, 0, 0, 0)
```

press_buttons(**buttons*)

Press and hold the given buttons.

release_all_buttons()

Release all the buttons.

release_buttons(**buttons*)

Release the given buttons.

reset_all()

Release all buttons and set joysticks to zero.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index

P

Q

R

S

T

U

V

W

X

Y

Z