AdafruitDisplay$_{B}uttonLibraryDocumentation$

Release 1.0

Limor Fried

Oct 25, 2021

Contents

UI Buttons for displayio

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.

1.1 Installing from PyPI

On supported GNU/Linux systems like the Raspberry Pi, you can install the driver locally from PyPI. To install for current user:

```
pip3 install adafruit-circuitpython-display-button
```

To install system-wide (this may be required in some cases):

```
sudo pip3 install adafruit-circuitpython-display-button
```

To install in a virtual environment in your current project:

```
mkdir project-name && cd project-name
python3 -m venv .env
source .env/bin/activate
pip3 install adafruit-circuitpython-display-button
```

CHAPTER 2

Usage Example

See examples in examples/ folder.

CHAPTER 3

Documentation

API documentation for this library can be found on Read the Docs.

CHAPTER 4

Contributing

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

CHAPTER 5

Building locally

5.1 Zip release files

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-display_button --
↪library_location .
```

5.2 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 6

Table of Contents

6.1 Simple test

Ensure your device works with this simple test.

Listing 1: examples/display_button_simpletest.py

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT
"""
Simple button example.
"""

import board
import displayio
import terminalio
import adafruit_touchscreen
from adafruit_button import Button

# use built in display (MagTag, PyPortal, PyGamer, PyBadge, CLUE, etc.)
# see guide for setting up external displays (TFT / OLED breakouts, RGB matrices, etc.)
# https://learn.adafruit.com/circuitpython-display-support-using-displayio/display-and-display-bus
display = board.DISPLAY

# --| Button Config |-----------------------------------------------
BUTTON_X = 110
BUTTON_Y = 95
BUTTON_WIDTH = 100
BUTTON_HEIGHT = 50
BUTTON_STYLE = Button.ROUNDRECT
BUTTON_FILL_COLOR = 0x00FFFF
BUTTON_OUTLINE_COLOR = 0xFF00FF
```

(continues on next page)

(continued from previous page)

```
BUTTON_LABEL = "HELLO WORLD"
BUTTON_LABEL_COLOR = 0x000000
# --| Button Config |-----------------------------------------------

# Setup touchscreen (PyPortal)
ts = adafruit_touchscreen.Touchscreen(
    board.TOUCH_XL,
    board.TOUCH_XR,
    board.TOUCH_YD,
    board.TOUCH_YU,
    calibration=((5200, 59000), (5800, 57000)),
    size=(display.width, display.height),
)

# Make the display context
splash = displayio.Group()
display.show(splash)

# Make the button
button = Button(
    x=BUTTON_X,
    y=BUTTON_Y,
    width=BUTTON_WIDTH,
    height=BUTTON_HEIGHT,
    style=BUTTON_STYLE,
    fill_color=BUTTON_FILL_COLOR,
    outline_color=BUTTON_OUTLINE_COLOR,
    label=BUTTON_LABEL,
    label_font=terminalio.FONT,
    label_color=BUTTON_LABEL_COLOR,
)

# Add button to the display context
splash.append(button)

# Loop and look for touches
while True:
    p = ts.touch_point
    if p:
        if button.contains(p):
            button.selected = True
        else:
            button.selected = False  # if touch is dragged outside of button
    else:
        button.selected = False  # if touch is released
```

6.2 Button Color Properties

Demonstrate the different color possibilities present in the library

Listing 2: examples/display_button_color_properties.py

```
# SPDX-FileCopyrightText: 2021 Tim Cocks for Adafruit Industries
# SPDX-License-Identifier: MIT
```

(continues on next page)

(continued from previous page)

```
3
4   """
5   Basic example that illustrates how to set the various color options on the button␣
    ↪using
6   properties after the button has been initialized.
7   """
8
9   import board
10  import displayio
11  import terminalio
12  import adafruit_touchscreen
13  from adafruit_button import Button
14
15  # use built in display (MagTag, PyPortal, PyGamer, PyBadge, CLUE, etc.)
16  # see guide for setting up external displays (TFT / OLED breakouts, RGB matrices, etc.
    ↪)
17  # https://learn.adafruit.com/circuitpython-display-support-using-displayio/display-
    ↪and-display-bus
18  display = board.DISPLAY
19
20  # --| Button Config |-----------------------------------------------
21  BUTTON_X = 110
22  BUTTON_Y = 95
23  BUTTON_WIDTH = 100
24  BUTTON_HEIGHT = 50
25  BUTTON_STYLE = Button.ROUNDRECT
26  BUTTON_FILL_COLOR = 0xAA0000
27  BUTTON_OUTLINE_COLOR = 0x0000FF
28  BUTTON_LABEL = "HELLO WORLD"
29  BUTTON_LABEL_COLOR = 0x000000
30  # --| Button Config |-----------------------------------------------
31
32  # Setup touchscreen (PyPortal)
33  ts = adafruit_touchscreen.Touchscreen(
34      board.TOUCH_XL,
35      board.TOUCH_XR,
36      board.TOUCH_YD,
37      board.TOUCH_YU,
38      calibration=((5200, 59000), (5800, 57000)),
39      size=(display.width, display.height),
40  )
41
42  # Make the display context
43  splash = displayio.Group()
44  display.show(splash)
45
46  # Make the button
47  button = Button(
48      x=BUTTON_X,
49      y=BUTTON_Y,
50      width=BUTTON_WIDTH,
51      height=BUTTON_HEIGHT,
52      style=BUTTON_STYLE,
53      fill_color=BUTTON_FILL_COLOR,
54      outline_color=BUTTON_OUTLINE_COLOR,
55      label="HELLO WORLD",
56      label_font=terminalio.FONT,
```

(continues on next page)

(continued from previous page)

```
    label_color=BUTTON_LABEL_COLOR,
)

button.fill_color = 0x00FF00
button.outline_color = 0xFF0000

button.selected_fill = (0, 0, 255)
button.selected_outline = (255, 0, 0)

button.label_color = 0xFF0000
button.selected_label = 0x00FF00

# Add button to the display context
splash.append(button)

# Loop and look for touches
while True:
    p = ts.touch_point
    if p:
        if button.contains(p):
            print(p)
            button.selected = True
    else:
        button.selected = False
```

6.3 Button Custom Font

Shows how to use different fonts with your button

Listing 3: examples/display_button_customfont.py

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT
"""
Button example with a custom font.
"""

import os
import board
import displayio
from adafruit_bitmap_font import bitmap_font
import adafruit_touchscreen
from adafruit_button import Button

# use built in display (MagTag, PyPortal, PyGamer, PyBadge, CLUE, etc.)
# see guide for setting up external displays (TFT / OLED breakouts, RGB matrices, etc.)
# https://learn.adafruit.com/circuitpython-display-support-using-displayio/display-and-display-bus
display = board.DISPLAY

# These pins are used as both analog and digital! XL, XR and YU must be analog
# and digital capable. YD just need to be digital
ts = adafruit_touchscreen.Touchscreen(
```

(continues on next page)

(continued from previous page)

```
    board.TOUCH_XL,
    board.TOUCH_XR,
    board.TOUCH_YD,
    board.TOUCH_YU,
    calibration=((5200, 59000), (5800, 57000)),
    size=(display.width, display.height),
)

# the current working directory (where this file is)
cwd = ("/" + __file__).rsplit("/", 1)[0]
fonts = [
    file
    for file in os.listdir(cwd + "/fonts/")
    if (file.endswith(".bdf") and not file.startswith("._"))
]
for i, filename in enumerate(fonts):
    fonts[i] = cwd + "/fonts/" + filename
print(fonts)
THE_FONT = "/fonts/Arial-12.bdf"
DISPLAY_STRING = "Button Text"

# Make the display context
splash = displayio.Group()
display.show(splash)
BUTTON_WIDTH = 80
BUTTON_HEIGHT = 40
BUTTON_MARGIN = 20

################################################################################
# Make a background color fill

color_bitmap = displayio.Bitmap(display.width, display.height, 1)
color_palette = displayio.Palette(1)
color_palette[0] = 0x404040
bg_sprite = displayio.TileGrid(color_bitmap, pixel_shader=color_palette, x=0, y=0)
print(bg_sprite.x, bg_sprite.y)
splash.append(bg_sprite)

################################################################################

# Load the font
font = bitmap_font.load_font(THE_FONT)

buttons = []
# Default button styling:
button_0 = Button(
    x=BUTTON_MARGIN,
    y=BUTTON_MARGIN,
    width=BUTTON_WIDTH,
    height=BUTTON_HEIGHT,
    label="button0",
    label_font=font,
)
buttons.append(button_0)

# a button with no indicators at all
button_1 = Button(
```

(continues on next page)

(continued from previous page)

```
    x=BUTTON_MARGIN * 2 + BUTTON_WIDTH,
    y=BUTTON_MARGIN,
    width=BUTTON_WIDTH,
    height=BUTTON_HEIGHT,
    fill_color=None,
    outline_color=None,
)
buttons.append(button_1)

# various colorings
button_2 = Button(
    x=BUTTON_MARGIN * 3 + 2 * BUTTON_WIDTH,
    y=BUTTON_MARGIN,
    width=BUTTON_WIDTH,
    height=BUTTON_HEIGHT,
    label="button2",
    label_font=font,
    label_color=0x0000FF,
    fill_color=0x00FF00,
    outline_color=0xFF0000,
)
buttons.append(button_2)

# Transparent button with text
button_3 = Button(
    x=BUTTON_MARGIN,
    y=BUTTON_MARGIN * 2 + BUTTON_HEIGHT,
    width=BUTTON_WIDTH,
    height=BUTTON_HEIGHT,
    label="button3",
    label_font=font,
    label_color=0x0,
    fill_color=None,
    outline_color=None,
)
buttons.append(button_3)

# a roundrect
button_4 = Button(
    x=BUTTON_MARGIN * 2 + BUTTON_WIDTH,
    y=BUTTON_MARGIN * 2 + BUTTON_HEIGHT,
    width=BUTTON_WIDTH,
    height=BUTTON_HEIGHT,
    label="button4",
    label_font=font,
    style=Button.ROUNDRECT,
)
buttons.append(button_4)

# a shadowrect
button_5 = Button(
    x=BUTTON_MARGIN * 3 + BUTTON_WIDTH * 2,
    y=BUTTON_MARGIN * 2 + BUTTON_HEIGHT,
    width=BUTTON_WIDTH,
    height=BUTTON_HEIGHT,
    label="button5",
    label_font=font,
```

(continues on next page)

(continued from previous page)

```
    style=Button.SHADOWRECT,
)
buttons.append(button_5)

# a shadowroundrect
button_6 = Button(
    x=BUTTON_MARGIN,
    y=BUTTON_MARGIN * 3 + BUTTON_HEIGHT * 2,
    width=BUTTON_WIDTH,
    height=BUTTON_HEIGHT,
    label="button6",
    label_font=font,
    style=Button.SHADOWROUNDRECT,
)
buttons.append(button_6)

for b in buttons:
    splash.append(b)

while True:
    p = ts.touch_point
    if p:
        print(p)
        for i, b in enumerate(buttons):
            if b.contains(p):
                print("Button %d pressed" % i)
                b.selected = True
            else:
                b.selected = False
```

6.4 Soundboard

A soundboard made with buttons

Listing 4: examples/display_button_soundboard.py

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT
"""
Soundboard example with buttons.
"""

import time
from adafruit_pyportal import PyPortal
from adafruit_button import Button

SHOW_BUTTONS = False

# the current working directory (where this file is)
cwd = ("/" + __file__).rsplit("/", 1)[0]
# No internet use version of pyportal
pyportal = PyPortal(default_bg=cwd + "/button_background.bmp")

spots = []
```

(continues on next page)

(continued from previous page)

```
spots.append({"label": "1", "pos": (10, 10), "size": (60, 60), "file": "01.wav"})
spots.append({"label": "2", "pos": (90, 10), "size": (60, 60), "file": "02.wav"})
spots.append({"label": "3", "pos": (170, 10), "size": (60, 60), "file": "03.wav"})
spots.append({"label": "4", "pos": (250, 10), "size": (60, 60), "file": "04.wav"})
spots.append({"label": "5", "pos": (10, 90), "size": (60, 60), "file": "05.wav"})
spots.append({"label": "6", "pos": (90, 90), "size": (60, 60), "file": "06.wav"})
spots.append({"label": "7", "pos": (170, 90), "size": (60, 60), "file": "07.wav"})
spots.append({"label": "8", "pos": (250, 90), "size": (60, 60), "file": "08.wav"})
spots.append({"label": "9", "pos": (10, 170), "size": (60, 60), "file": "09.wav"})
spots.append({"label": "10", "pos": (90, 170), "size": (60, 60), "file": "10.wav"})
spots.append({"label": "11", "pos": (170, 170), "size": (60, 60), "file": "11.wav"})
spots.append({"label": "12", "pos": (250, 170), "size": (60, 60), "file": "12.wav"})

buttons = []
for spot in spots:
    fill = outline = None
    if SHOW_BUTTONS:
        fill = None
        outline = 0x00FF00
    button = Button(
        x=spot["pos"][0],
        y=spot["pos"][1],
        width=spot["size"][0],
        height=spot["size"][1],
        fill_color=fill,
        outline_color=outline,
        label=spot["label"],
        label_color=None,
        name=spot["file"],
    )
    pyportal.splash.append(button)
    buttons.append(button)

last_pressed = None
currently_pressed = None
while True:
    p = pyportal.touchscreen.touch_point
    if p:
        print(p)
        for b in buttons:
            if b.contains(p):
                print("Touched", b.name)
                if currently_pressed != b:  # don't restart if playing
                    pyportal.play_file(cwd + "/" + b.name, wait_to_finish=False)
                currently_pressed = b
                break
        else:
            currently_pressed = None
    time.sleep(0.05)
```

6.5 adafruit_button

UI Buttons for displayio

- Author(s): Limor Fried

6.5.1 Implementation Notes

Software and Dependencies:

- Adafruit CircuitPython firmware for the supported boards: https://github.com/adafruit/circuitpython/releases

class adafruit_button.**Button**(*, *x*, *y*, *width*, *height*, *name=None*, *style=0*, *fill_color=16777215*, *outline_color=0*, *label=None*, *label_font=None*, *label_color=0*, *selected_fill=None*, *selected_outline=None*, *selected_label=None*)

Helper class for creating UI buttons for `displayio`.

Parameters

- **x** – The x position of the button.
- **y** – The y position of the button.
- **width** – The width of the button in pixels.
- **height** – The height of the button in pixels.
- **name** – The name of the button.
- **style** – The style of the button. Can be RECT, ROUNDRECT, SHADOWRECT, SHADOWROUNDRECT. Defaults to RECT.
- **fill_color** – The color to fill the button. Defaults to 0xFFFFFF.
- **outline_color** – The color of the outline of the button.
- **label** – The text that appears inside the button. Defaults to not displaying the label.
- **label_font** – The button label font.
- **label_color** – The color of the button label text. Defaults to 0x0.
- **selected_fill** – Inverts the fill color.
- **selected_outline** – Inverts the outline color.
- **selected_label** – Inverts the label color.

contains(*point*)

Used to determine if a point is contained within a button. For example, `button.contains(touch)` where `touch` is the touch point on the screen will allow for determining that a button has been touched.

fill_color

The fill color of the button body

group

Return self for compatibility with old API.

height

The height of the button

label

The text label of the button

label_color

The font color of the button

outline_color

The outline color of the button body

resize(*new_width*, *new_height*)

Resize the button to the new width and height given :param new_width int the desired width :param new_height int the desired height

selected
Selected inverts the colors.

selected_fill
The fill color of the button body when selected

selected_label
The font color of the button when selected

selected_outline
The outline color of the button body when selected

width
The width of the button

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index