CircuitPython

DisplayIO $switchRoundLibraryDocumentation$

Release 1.0

Kevin Matocha

Jul 05, 2021

CONTENTS

A sliding switch widget with a round shape.

CHAPTER

ONE

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 or individual libraries can be installed using circup.

CHAPTER

TWO

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 circuitpython-displayio-switchround
```

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

```
sudo pip3 install circuitpython-displayio-switchround
```

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 circuitpython-displayio-switchround
```

CHAPTER

THREE

USAGE EXAMPLE

See scripts in the examples directory of this repository.

CHAPTER

FOUR

CONTRIBUTING

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

CHAPTER

FIVE

DOCUMENTATION

For information on building library documentation, please check out this guide.

CHAPTER

SIX

TABLE OF CONTENTS

6.1 Simple test

Create a single sliding switch.

Listing 1: examples/displayio_switchround_simpletest.py

```
# SPDX-FileCopyrightText: 2021 Kevin Matocha
#
# SPDX-License-Identifier: MIT
"""
Creates a single sliding switch widget.
"""

import time
import board
import displayio
import adafruit_touchscreen
from adafruit_displayio_layout.widgets.switch_round import SwitchRound as Switch

display = board.DISPLAY

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),
)

# Create the switch
my_switch = Switch(20, 30)


my_group = displayio.Group()
my_group.append(my_switch)

# Add my_group to the display
display.show(my_group)

```

(continues on next page)

(continued from previous page)

```
# Start the main loop
while True:

    p = ts.touch_point  # get any touches on the screen

    if p:  # Check each switch if the touch point is within the switch touch area
        # If touched, then flip the switch with .selected
        if my_switch.contains(p):
            my_switch.selected(p)

    time.sleep(0.05)  # touch response on PyPortal is more accurate with a small delay
```

6.2 Switch test with multiple switches

Create multiple sliding switch with various sizes and orientations.

Listing 2: examples/displayio_switchround_multiple.py

```
# SPDX-FileCopyrightText: 2021 Kevin Matocha
#
# SPDX-License-Identifier: MIT
"""
Creates multiple sliding switch widgets of various size and orientations.
"""

import time
import board
import displayio
import adafruit_touchscreen
from adafruit_displayio_layout.widgets.switch_round import SwitchRound as Switch

display = board.DISPLAY

# setup the touch screen
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),
)


# Create the switches

my_switch = Switch(20, 30)

my_switch2 = Switch(
    x=120,
```

(continues on next page)

(continued from previous page)

```
    y=35,
    height=30,  # Set height to 30 pixels.  If you do not specify width,
    # it is automatically set to a default aspect ratio
    touch_padding=10,  # add extra boundary for touch response
    value=True,
)  # initial value is set to True

my_switch3 = Switch(
    x=20,
    y=85,
    height=40,
    fill_color_off=(255, 0, 0),  # Set off colorred, can use hex code (0xFF0000)
    outline_color_off=(80, 0, 0),
    background_color_off=(150, 0, 0),
    background_outline_color_off=(30, 0, 0),
)

my_switch4 = Switch(
    x=120,
    y=85,
    height=40,
    width=110,  # you can set the width manually but it may look weird
)

my_switch5 = Switch(
    x=20,
    y=140,
    height=40,
    display_button_text=False,  # do not show the 0/1 on the switch
)

my_switch6 = Switch(
    x=120,
    y=140,
    height=40,
    horizontal=False,  # set orientation to vertical
)

my_switch7 = Switch(
    x=180,
    y=140,
    height=40,
    horizontal=False,  # set orientation to vertical
    flip=True,  # swap the direction
)

my_switch8 = Switch(
    x=0,
    y=0,  # this is a larger, vertical orientation switch
    height=60,
    horizontal=False,  # set orientation to vertical
    flip=True,  # swap the direction
```

(continues on next page)

(continued from previous page)

```
)
# use anchor_point and anchored_position to set the my_switch8 position
# relative to the display size.
my_switch8.anchor_point = (1.0, 1.0)
# the switch anchor_point is the bottom right switch corner
my_switch8.anchored_position = (display.width - 10, display.height - 10)
# the switch anchored_position is 10 pixels from the display
# lower right corner

my_group = displayio.Group()
my_group.append(my_switch)
my_group.append(my_switch2)
my_group.append(my_switch3)
my_group.append(my_switch4)
my_group.append(my_switch5)
my_group.append(my_switch6)
my_group.append(my_switch7)
my_group.append(my_switch8)

# Add my_group to the display
display.show(my_group)


# Start the main loop
while True:

    p = ts.touch_point  # get any touches on the screen

    if p:  # Check each switch if the touch point is within the switch touch area
        # If touched, then flip the switch with .selected
        if my_switch.contains(p):
            my_switch.selected(p)

        elif my_switch2.contains(p):
            my_switch2.selected(p)

        elif my_switch3.contains(p):
            my_switch3.selected(p)

        elif my_switch4.contains(p):
            my_switch4.selected(p)

        elif my_switch5.contains(p):
            my_switch5.selected(p)

        elif my_switch6.contains(p):
            my_switch6.selected(p)

        elif my_switch7.contains(p):
            my_switch7.selected(p)

        elif my_switch8.contains(p):
```

(continues on next page)

(continued from previous page)

```
            my_switch8.selected(p)

    time.sleep(0.05)  # touch response on PyPortal is more accurate with a small delay
```

6.3 Overview of SwitchRound

6.3.1 Quickstart: Importing and using SwitchRound

Here is one way of importing the *SwitchRound* class so you can use it as the name `Switch`:

```
from displayio_switchround import SwitchRound as Switch
```

Now you can create a switch at pixel position x=20, y=30 using:

```
my_switch = Switch(20, 30) # create the switch at x=20, y=30
```

Once you setup your display, you can now add `my_switch` to your display using:

```
display.show(my_switch)  # add the group to the display
```

If you want to have multiple display elements, you can create a group and then append the switch and the other elements to the group. Then, you can add the full group to the display as in this example:

```
my_switch = Switch(20, 30)  # create the switch at x=20, y=30
my_group = displayio.Group()  # make a group
my_group.append(my_switch)  # Add my_switch to the group

#
# Append other display elements to the group
#

display.show(my_group)  # add the group to the display
```

For a full example, including how to respond to screen touches, check out the following examples in the Adafruit_CircuitPython_DisplayIO_Layout library:

- examples/displayio_layout_switch_simpletest.py
- examples/displayio_layout_switch_multiple.py

6.3.2 Summary: SwitchRound Features and input variables

The *SwitchRound* widget has numerous options for controlling its position, visible appearance, orientation, animation speed and value through a collection of input variables:

- **position:** *x* and *y* or *anchor_point* and *anchored_position*
- **size:** *width* and *height*

  It is recommended to leave `width = None` to use the preferred aspect ratio.
- **orientation and movement direction (on vs. off):** `horizontal` and `flip`
- **switch color:** `fill_color_off`, `fill_color_on`, `outline_color_off` and `outline_color_on`

- **background color:** `background_color_off`, `background_color_on`, `background_outline_color_off` and `background_outline_color_on`
- **linewidths:** `switch_stroke` and `text_stroke`
- **0/1 display:** `display_button_text`

  Set to `True` if you want the 0/1 shapes to show on the switch
- **animation:** `animation_time`

  Set the duration (in seconds) it will take to transition the switch, use `0` if you want it to snap into position immediately. The default value of `0.2` seconds is a good starting point, and larger values for bigger switches.
- **value:** *value*

  Set to the initial value (`True` or `False`)
- **touch boundaries:** `touch_padding`

  This defines the number of additional pixels surrounding the switch that should respond to a touch. (Note: The `touch_padding` variable updates the `touch_boundary` Control class variable. The definition of the `touch_boundary` is used to determine the region on the Widget that returns `True` in the *contains()* method.)

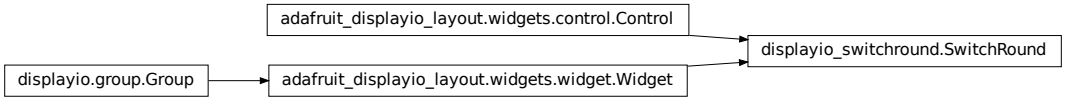
6.3.3 Description of features

The *SwitchRound* widget is a sliding switch that changes state whenever it is touched. The color gradually changes from the off-state color scheme to the on-state color scheme as the switch transfers from off to the on position. The switch has an optional display of "0" and "1" on the sliding switch. The switch can be oriented using the `horizontal` input variable, and the sliding direction can be changed using the `flip` input variable.

Regarding switch sizing, it is recommended to set the height dimension but to leave the `width = None`. Setting `width = None` will allow the width to resize to maintain a recommended aspect ratio of width/height. Alternately, the switch can be resized using the *resize()* method, and it will adjust the width and height to the maximum size that will fit inside the requested width and height dimensions, while keeping the preferred aspect ratio. To make the switch easier to be selected, additional padding around the switch can be defined using the `touch_padding` input variable to increase the touch-responsive area. The duration of animation between on/off can be set using the `animation_time` input variable.

6.3.4 Internal details: How the SwitchRound widget works

The *SwitchRound* widget is a graphical element that responds to touch elements to provide sliding switch on/off behavior. Whenever touched, the switch toggles to its alternate value. The following sections describe the construction of the *SwitchRound* widget, in the hopes that it will serve as a first example of the key properties and responses for widgets.



The *SwitchRound* widget inherits from two classes, it is a subclass of `Widget`, which itself is a subclass of `displayio.Group`, and a subclass of `Control`. The `Widget` class helps define the positioning and sizing of the switch, while th `Control` class helps define the touch-response behavior.

The following sections describe the structure and inner workings of *SwitchRound*.

6.3.5 Group structure: Display elements that make up SwitchRound

The `Widget` class is a subclass of `displayio.Group`, thus we can append graphical elements to the Widget for displaying on the screen. The switch consists of the following graphical elements:

0. switch_roundrect: The switch background
1. switch_circle: The switch button that slides back and forth
2. text_0 [Optional]: The "0" circle shape on the switch button
3. text_1 [Optional]: The "1" rectangle shape on the switch button

The optional text items can be displayed or hidden using the `display_button_text` input variable.

6.3.6 Coordinate systems and use of anchor_point and anchored_position

See the `Widget` class definition for clarification on the methods for positioning the switch, including the difference in the display coordinate system and the Widget's local coordinate system.

6.3.7 The Widget construction sequence

Here is the set of steps used to define this sliding switch widget.

1. Initialize the stationary display items
2. Initialize the moving display elements
3. Store initial position of the moving display elements
4. Define "keyframes" to determine the translation vector
5. Define the `SwitchRound._draw_position()` method between 0.0 to 1.0 (and slightly beyond)
6. Select the motion "easing" function
7. **Extra**. Go check out the `SwitchRound._animate_switch()` method

First, the stationary background rounded rectangle (RoundRect is created). Second, the moving display elements are created, the circle for the switch, the circle for the text "0" and the rectangle for the text "1". Note that either the "0" or "1" is set as hidden, depending upon the switch value. Third, we store away the initial position of the three moving elements, these initial values will be used in the functions that move these display elements. Next, we define the motion of the moving element, by setting the `self._x_motion` and `self._y_motion` values that depending upon the `horizontal` and `flip` variables. These motion variables set the two "keyframes" for the moving elements, basically the endpoints of the switch motion. (Note: other widgets may need an `_angle_motion` variable if they require some form of rotation.) Next, we define the `SwitchRound._draw_function()` method. This method takes an input between 0.0 and 1.0 and adjusts the position relative to the motion variables, where 0.0 is the initial position and 1.0 represents the final position (as defined by the `_x_motion` and `_y_motion` values). In the case of the sliding

switch, we also use this `SwitchRound.position` value (0.0 to 1.0) to gradually grade the color of the components between their "on" and "off" colors.

6.3.8 Making it move

Everything above has set the ground rules for motion, but doesn't cause it to move. However, you have set almost all the pieces in place to respond to requests to change the position. All that is left is the **Extra** method that performs the animation, called `SwitchRound._animate_switch()`. The `SwitchRound._animate_switch()` method is triggered by a touch event through the `selected()` Control class method. Once triggered, this method checks how much time has elapsed. Based on the elapsed time and the `SwitchRound.animation_time` input variable, the `SwitchRound._animate_switch()` method calculates the `SwitchRound.position` where the switch should be. Then, it takes this `SwitchRound.position` to call the `SwitchRound._draw_position()` method that will update the display elements based on the requested position.

But there's even one more trick to the animation. The `SwitchRound._animate_switch()` calculates the target position based on a linear relationship between the time and the position. However, to give the animation a better "feel", it is desirable to tweak the motion function depending upon how this widget should behave or what suits your fancy. To do this we can use an *"easing"* function. In short, this adjusts the constant speed (linear) movement to a variable speed during the movement. Said another way, it changes the position versus time function according to a specific waveform equation. There are a lot of different "easing" functions that folks have used or you can make up your own. Some common easing functions are provided in the `adafruit_displayio_layout.widgets.easing` module. You can change the easing function based on changing which function is imported at the top of this file. You can see where the position is tweaked by the easing function in the line in the `SwitchRound._animate_switch()` method:

```
self._draw_position(easing(position))  # update the switch position
```

Go play around with the different easing functions and observe how the motion behavior changes. You can use these functions in multiple dimensions to get all varieties of behavior that you can take advantage of. The website easings.net can help you visualize some of the behavior of the easing functions.

Note: Some of the "springy" easing functions require position values slightly below 0.0 and slightly above 1.0, so if you want to use these, be sure to check that your `_draw_position()` method behaves itself for that range of position inputs.

6.3.9 Orientation and a peculiarity of width and height definitions for SwitchRound

In setting the switch sizing, use height and width to set the narrow and wide dimension of the switch. To try and reduce confusion, the orientation is modified after the height and width are selected. That is, if the switch is set to vertical, the height and still mean the "narrow" and the width will still mean the dimensions in the direction of the sliding.

If you need the switch to fit within a specific bounding box, it's preferred to use the *`resize()`* function. This will put the switch (in whatever orientation) at the maximum size where it can fit within the bounding box that you specified. The Switch aspect ratio will remain at the "preferred" aspect ratio of 2:1 (width:height) after the resizing.

6.3.10 Setting the touch response boundary

The touch response area is defined by the Control class variable called `touch_boundary`. In the case of the *SwitchRound* widget, we provide an `SwitchRound.touch_padding` input variable. The use of `SwitchRound.touch_padding` defines an additional number of pixels surrounding the display elements that respond to touch events. To achieve this additional space, the `touch_boundary` increases in size in all dimensions by the number of pixels specified in the `SwitchRound.touch_padding` parameter.

The `touch_boundary` is used in the Control function *contains()* that checks whether any touch_points are within the boundary. Please pay particular attention to the *SwitchRound contains()* method, since it calls the `contains()` superclass method with the touch_point value adjusted for the switch's *x* and *y* values. This offset adjustment is required since the `contains()` function operates only on the widget's local coordinate system. It's good to keep in mind which coordinate system you are working in, to ensure your code responds to the right inputs!

6.3.11 Summary

The *SwitchRound* widget is an example to explain the use of the `Widget` and `Control` class methods. The `Widget` class handles the overall sizing and positioning function and is the group that holds all the graphical elements. The `Control` class is used to define the response of the widget to touch events (or could be generalized to other inputs). Anything that only displays (such as a graph or an indicator light) won't need to inherit the `Control` class. But anything that responds to touch inputs should inherit the `Control` class to define the `touch_boundary` and the touch response functions.

I hope this *SwitchRound* widget will help turn on some new ideas and highlight some of the new capabilities of the `Widget` and `Control` classes. Now go see what else you can create and extend from here!

6.3.12 A Final Word

The design of the Widget and Control classes are open for inputs. If you think any additions or changes are useful, add it and please submit a pull request so others can use it too! Also, keep in mind you don't even need to follow these classes to get the job done. The Widget and Class definitions are designed to give guidance about one way to make things work, and to try to share some code. If it's standing in your way, do something else! If you want to use the `grid_layout` or other layout tools in this library, you only *really* need to have methods for positioning and resizing.

Note: Never let any of these class definitions hold you back, let your imagination run wild and make some cool widgets!

6.4 displayio_switchround

A sliding switch widget with a round shape.

- Author(s): Kevin Matocha

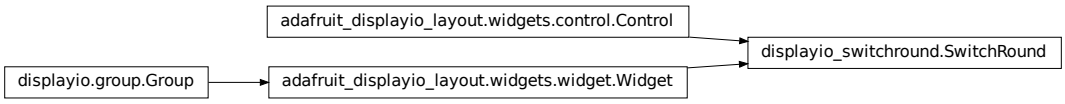
6.4.1 Implementation Notes

Hardware:

Software and Dependencies:

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

6.4.2 Inheritance



class displayio_switchround.**SwitchRound**(*x=0*, *y=0*, *width=None*, *height=40*, *touch_padding=0*, *horizontal=True*, *flip=False*, *anchor_point=None*, *anchored_position=None*, *fill_color_off=(66, 44, 66)*, *fill_color_on=(0, 100, 0)*, *outline_color_off=(30, 30, 30)*, *outline_color_on=(0, 60, 0)*, *background_color_off=(255, 255, 255)*, *background_color_on=(0, 60, 0)*, *background_outline_color_off=None*, *background_outline_color_on=None*, *switch_stroke=2*, *text_stroke=None*, *display_button_text=True*, *animation_time=0.2*, *value=False*, ***kwargs*)

Create a SwitchRound. See *Overview* for more details.

Parameters

- **x** (*int*) – pixel position, defaults to 0
- **y** (*int*) – pixel position, defaults to 0
- **width** (*None*, *int*) – width of the switch in pixels, if set to `None` (**recommended**) the width will auto-size relative to the height, defaults to `None`
- **height** (*int*) – height of the switch in pixels, defaults to 40 pixels
- **touch_padding** (*int*) – the width of an additional border surrounding the switch that extends the touch response boundary, defaults to 0
- **horizontal** (*bool*) – To set vertical orientation, set `horizontal` to `False`, defaults to `True`
- **flip** (*bool*) – If `True` the on and off direction will be flipped, default is `True`
- **anchor_point** (*None*, *tuple(float, float)*) – (X,Y) values from 0.0 to 1.0 to define the anchor point relative to the switch bounding box, default is `None`

- **anchored_position** (*None*, *tuple(int, int)*) – (x,y) pixel value for the location of the anchor_point, default is None
- **fill_color_off** (*tuple(int, int, int)*, *int*) – switch off-state fill color, *as an RGB-tuple or 24-bit hex*, default is (66, 44, 66) gray.
- **fill_color_on** (*tuple(int, int, int)*, *int*) – switch on-state fill color, *as an RGB-tuple or 24-bit hex*, default is (0, 100, 0) green.
- **outline_color_off** (*tuple(int, int, int)*, *int*) – switch off-state outline color, *as an RGB-tuple or 24-bit hex*, default is (30, 30, 30) dark gray.
- **outline_color_on** (*tuple(int, int, int)*, *int*) – switch on-state outline color, *as an RGB-tuple or 24-bit hex*, default is (0, 60, 0) green
- **background_color_off** (*tuple(int, int, int)*, *int*) – background off-state color, *as an RGB-tuple or 24-bit hex*, default is (255, 255, 255) white
- **background_color_on** (*tuple(int, int, int)*, *int*) – background on-state color, *as an RGB-tuple or 24-bit hex*, default is (0, 60, 0) dark green
- **background_outline_color_off** (*None*, *tuple(int, int, int)*, *int*) – background outline color in off-state, *as an RGB-tuple or 24-bit hex or `None`*. If set to None this will default to background_color_off, default is None
- **background_outline_color_on** (*None*, *tuple(int, int, int)*, *int*) – background outline color in on-state, *as an RGB-tuple or 24-bit hex or `None`*. If set to None this will default to background_color_on, default is None
- **switch_stroke** (*int*) – outline stroke width for the switch and background, in pixels, default is 2
- **text_stroke** (*None*, *int*) – outline stroke width (in pixels) for the 0/1 text shape outlines, if set to None it will use the value for switch_stroke, default value is None
- **display_button_text** (*bool*) – If True display the 0/1 text shapes on the sliding switch. If False hide the 0/1 text shapes, default value is True
- **animation_time** (*float*) – time for the switching animation, in seconds, default value is 0.2 seconds.
- **value** (*bool*) – the initial value for the switch, default is False

selected(*touch_point*)

Response function when Switch is selected. When selected, the switch position and value is changed with an animation.

Parameters **touch_point** (*tuple(int, int)*) – x,y location of the screen, in absolute display coordinates.

Returns None

contains(*touch_point*)

Checks if the Widget was touched. Returns True if the touch_point is within the Control's touch_boundary.

Parameters **touch_point** (*tuple(int, int)*) – x,y location of the screen, in absolute display coordinates.

Returns bool

property value

The current Switch value (Boolean).

Returns bool

property width
The width of the Switch (int).

Returns int

property anchor_point
The anchor point for positioning the widget, works in concert with *anchored_position* The relative (X,Y) position of the widget where the anchored_position is placed. For example (0.0, 0.0) is the Widget's upper left corner, (0.5, 0.5) is the Widget's center point, and (1.0, 1.0) is the Widget's lower right corner.

Parameters **anchor_point** (*Tuple[float, float]*) – In relative units of the Widget size.

property anchored_position
The anchored position (in pixels) for positioning the widget, works in concert with *anchor_point*. The *anchored_position* is the x,y pixel position for the placement of the Widget's *anchor_point*.

Parameters **anchored_position** (*Tuple[int, int]*) – The (x,y) pixel position for the anchored_position (in pixels).

append(*layer*)
Append a layer to the group. It will be drawn above other layers.

property bounding_box
The boundary of the widget. [x, y, width, height] in Widget's local coordinates (in pixels). (getter only)

Returns Tuple[int, int, int, int]

property height
The height of the Switch (int).

Returns int

property hidden
True when the Group and all of it's layers are not visible. When False, the Group's layers are visible if they haven't been hidden.

index(*layer*)
Returns the index of the first copy of layer. Raises ValueError if not found.

insert(*index*, *layer*)
Insert a layer into the group.

pop(*index=- 1*)
Remove the ith item and return it.

remove(*layer*)
Remove the first copy of layer. Raises ValueError if it is not present.

property scale
Scales each pixel within the Group in both directions. For example, when scale=2 each pixel will be represented by 2x2 pixels.

update_transform(*parent_transform*)
Update the parent transform and child transforms

property x
X position of the Group in the parent.

property y
Y position of the Group in the parent.

resize(*new_width*, *new_height*)
Resize the Switch to a new requested width and height.

Parameters

- **new_width** (*int*) – requested maximum width
- **new_height** (*int*) – requested maximum height

Returns None

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

d

INDEX