CircuitPython

DisplayIO$_{Cartesian Library Documentation}$

Release 1.0

Jose David M.

Mar 13, 2022

CONTENTS

A cartesian plane widget for displaying graphical information.

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-cartesian
```

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

```
sudo pip3 install circuitpython-displayio-cartesian
```

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-cartesian
```

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

Ensure your device works with this simple test.

Listing 1: examples/displayio_cartesian_simpletest.py

```
# SPDX-FileCopyrightText: 2021 Jose David M.
#
# SPDX-License-Identifier: MIT
##############################
"""
This is a basic demonstration of a Cartesian widget.
"""

import time
import board
import displayio
import terminalio
from displayio_cartesian import Cartesian

# Fonts used for the Dial tick labels
tick_font = terminalio.FONT

display = board.DISPLAY  # create the display on the PyPortal or Clue (for example)
# otherwise change this to setup the display
# for display chip driver and pinout you have (e.g. ILI9341)


# Create a Cartesian widget
my_plane = Cartesian(
    x=150,  # x position for the plane
    y=100,  # y plane position
    width=100,  # display width
    height=100,  # display height
    axes_color=0xFFFFFF,  # axes line color
    axes_stroke=2,  # axes lines width in pixels
    tick_color=0xFFFFFF,  # ticks color
    major_tick_stroke=1,  # ticks width in pixels
    major_tick_length=5,  # ticks length in pixels
    tick_label_font=tick_font,  # the font used for the tick labels
```

(continues on next page)

(continued from previous page)

```
    font_color=0xFFFFFF,  # ticks line color
)

my_group = displayio.Group()
my_group.append(my_plane)
display.show(my_group)  # add high level Group to the display

posx = 0
posy = 0

while True:
    for i in range(0, 90, 2):
        my_plane.update_pointer(i, i)
        time.sleep(0.5)
```

6.2 Advanced test

Advanced test showing illustrating usage of more features.

Listing 2: examples/displayio_cartesian_advanced_test.py

```
# SPDX-FileCopyrightText: 2021 Jose David M.
#
# SPDX-License-Identifier: MIT
##############################
"""
This is a more advance demonstration of a Cartesian widget and some configurable
parameters.
"""

import board
import displayio
import terminalio
from displayio_cartesian import Cartesian

# Fonts used for the Dial tick labels
tick_font = terminalio.FONT

display = board.DISPLAY  # create the display on the PyPortal or Clue (for example)
# otherwise change this to setup the display
# for display chip driver and pinout you have (e.g. ILI9341)


# Create different Cartesian widgets
my_group = displayio.Group()

car = Cartesian(
    x=25,
    y=10,
    width=100,
```

(continues on next page)

(continued from previous page)

```
    height=100,
    subticks=True,
)
my_group.append(car)

car3 = Cartesian(
    x=150,
    y=10,
    width=150,
    height=100,
    xrange=(0, 160),
    axes_stroke=1,
    axes_color=0x990099,
    subticks=True,
)
my_group.append(car3)

car4 = Cartesian(
    x=30,
    y=140,
    width=80,
    height=80,
    axes_stroke=1,
    tick_color=0xFFFFFF,
    subticks=True,
)

my_group.append(car4)

car5 = Cartesian(
    x=180,
    y=140,
    width=70,
    height=70,
    xrange=(0, 120),
    yrange=(0, 90),
    tick_color=0x990099,
    axes_stroke=3,
    major_tick_length=10,
)
my_group.append(car5)

display.show(my_group)

while True:
    pass
```

6.3 displayio_cartesian

A cartesian plane widget for displaying graphical information.

- Author(s): Jose David M.

6.3.1 Implementation Notes

Hardware:

Software and Dependencies:

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

class displayio_cartesian.**Cartesian**(*background_color: int = 0*, *xrange: typing.Tuple[int, int] = (0, 100)*, *yrange: typing.Tuple[int, int] = (0, 100)*, *axes_color: int = 16777215*, *axes_stroke: int = 1*, *tick_color: int = 16777215*, *major_tick_stroke: int = 1*, *major_tick_length: int = 5*, *tick_label_font=<fontio.BuiltinFont object>*, *font_color: int = 16777215*, *pointer_radius: int = 1*, *pointer_color: int = 16777215*, *subticks: bool = False*, *nudge_x: int = 0*, *nudge_y: int = 0*, *verbose: bool = False*, *fill_area: bool = False*, ***kwargs*)

A cartesian widget. The origin is set using `x` and `y`.

Parameters

- **x** (*int*) – x position of the plane origin
- **y** (*int*) – y position of the plane origin
- **background_color** (*int*) – background color to use defaults to black (0x000000)
- **width** (*int*) – requested width, in pixels.
- **height** (*int*) – requested height, in pixels.
- **xrange** ((*int, int*)) – X axes range. Defaults to (0, 100)
- **yrange** ((*int, int*)) – Y axes range. Defaults to (0, 100)
- **axes_color** (*int*) – axes lines color defaults to white (0xFFFFFF)
- **axes_stroke** (*int*) – axes lines thickness in pixels defaults to 2
- **major_tick_stroke** (*int*) – tick lines thickness in pixels defaults to 1
- **major_tick_length** (*int*) – tick lines length in pixels defaults to 5
- **tick_label_font** (*terminalio.FONT*) – tick label text font
- **font_color** (*int*) – font color. Defaults to white (0xFFFFFF)
- **pointer_radius** (*int*) – pointer radius in pixels defaults to 1
- **pointer_color** (*int*) – pointer color. Defaults to white (0xFFFFFF)
- **subticks** (*bool*) – inclusion of subticks in the plot area. Default to False
- **nudge_x** (*int*) – movement in pixels in the x direction to move the origin. Defaults to 0
- **nudge_y** (*int*) – movement in pixels in the y direction to move the origin. Defaults to 0
- **verbose** (*bool*) – print debugging information in some internal functions. Default to False

Quickstart: Importing and using Cartesian

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

```
from displayio_cartesian import Cartesian as Plane
```

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

```
my_plane=Plane(x=20, y=30) # instance the plane at x=20, y=30
```

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

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

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

```
my_plane= Plane(20, 30) # instance the plane at x=20, y=30
my_group = displayio.Group() # make a group
my_group.append(my_plane) # Add my_plane to the group

#
# Append other display elements to the group
#

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

Summary: Cartesian Features and input variables

The *Cartesian* widget has some options for controlling its position, visible appearance, and scale through a collection of input variables:

- **position**: `x`, `y`, `anchor_point`, `anchored_position` and `nudge_x`, `nudge_y`. Nudge parameters are used to account for the float and int conversions required to display different ranges and values. Conversion are required as displays work in integers and not floats
- **size**: `width` and `height`
- **color**: `axes_color`, `font_color`, `tick_color`, `pointer_color`
- **background color**: `background_color`
- **linewidths**: `axes_stroke` and `major_tick_stroke`
- **range**: `xrange` and `yrange` This is the range in absolute units. For example, when using (20-90), the X axis will start at 20 finishing at 90. However the height of the graph is given by the height parameter. The scale is handled internal to provide a 1:1 experience when you update the graph.

Fig. 1: This is a diagram of a cartesian widget with the pointer moving in the plot area.

Parameters

- **scale** (*int*) – Scale of layer pixels in one dimension.
- **x** (*int*) – Initial x position within the parent.
- **y** (*int*) – Initial y position within the parent.

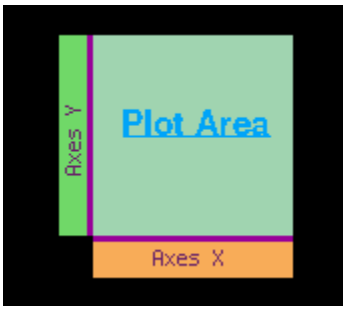


Fig. 2: This is a diagram of a cartesian widget showing the different zones.

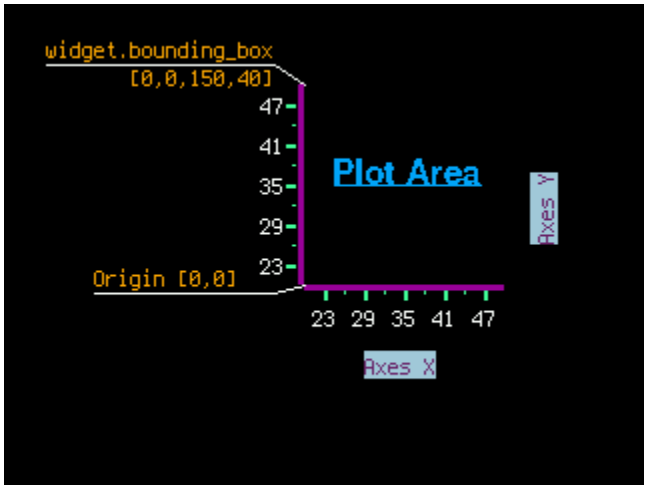


Fig. 3: This is a diagram of a cartesian widget showing localisation scheme.

update_pointer(*x: int*, *y: int*) → None
updater_pointer function helper function to update pointer in the plane :param int x: x coordinate in the local plane :param int y: y coordinate in the local plane :return: None

property fill_area: bool
Whether the area under the graph (integral) should be shaded

add_plot_line(*x: int*, *y: int*) → None
add_plot_line function.

add line to the plane. multiple calls create a line-plot graph.

Parameters

- **x** (*int*) – x coordinate in the local plane
- **y** (*int*) – y coordinate in the local plane

Returns None

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: Union[displayio._group.Group, displayio._tilegrid.TileGrid]*) → None
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 widget height, in pixels. (getter only)

Returns int

property hidden: bool
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: Union[displayio._group.Group, displayio._tilegrid.TileGrid]*) → int
Returns the index of the first copy of layer. Raises ValueError if not found.

insert(*index: int*, *layer: Union[displayio._group.Group, displayio._tilegrid.TileGrid]*) → None
Insert a layer into the group.

pop(*index: int = - 1*) → Union[displayio._group.Group, displayio._tilegrid.TileGrid]
Remove the ith item and return it.

remove(*layer: Union[displayio._group.Group, displayio._tilegrid.TileGrid]*) → None
Remove the first copy of layer. Raises ValueError if it is not present.

resize(*new_width*, *new_height*)
Resizes the widget dimensions (for use with automated layout functions).

IMPORTANT: The *resize* function should be overridden by the subclass definition.

The width and height are provided together so the subclass *resize* function can apply any constraints that require consideration of both width and height (such as maintaining a Widget's preferred aspect ratio). The Widget should be resized to the maximum size that can fit within the dimensions defined by the requested *new_width* and *new_height*. After resizing, the Widget's *bounding_box* should also be updated.

Parameters

- **new_width** (*int*) – target maximum width (in pixels)
- **new_height** (*int*) – target maximum height (in pixels)

Returns None

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

sort(*key: Callable*, *reverse: bool*) → None
Sort the members of the group.

property width
The widget width, in pixels. (getter only)

Returns int

property x: int
X position of the Group in the parent.

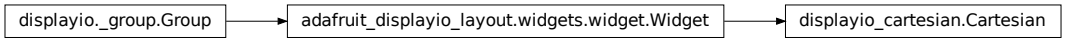
property y: int
Y position of the Group in the parent.

clear_plot_lines(*palette_index: int = 5*) → None
clear_plot_lines function.

clear all added lines (clear line-plot graph)

Parameters **palette_index** (*int*) – color palett index. Defaults to 5

Returns None



CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

d

INDEX