ssd1306 Documentation

Release 1.2.0

Richard Hull

Dec 09, 2016

Contents

CHAPTER 1

Introduction

Interfacing OLED matrix displays with the SSD1306 (or SH1106) driver in Python 2 or 3 using I2C/SPI on the Raspberry Pi.

The SSD1306 display pictured below is 128x64 pixels, and the board is *tiny*, and will fit neatly inside the RPi case (the SH1106 is slightly different, in that it supports 132 x 64 pixels). My intention is to solder the wires directly to the underside of the RPi GPIO pins (P5 header) so that the pins are still available for other purposes, but the regular, top GPIO pins (P1 header) can also be used of course.

See also:

Further technical details for the SSD1306 OLED display can be found in the `datasheet`. See also the datasheet for the `SH1106 chipset`. Benchmarks for tested devices can be found in the wiki.

As well as display drivers for SSD1306- and SH1106-class OLED devices there are emulators that run in real-time (with pygame) and others that can take screenshots, or assemble animated GIFs, as per the examples below (source code for these is available in the examples directory:

CHAPTER 2

Python usage

The screen can be driven with python using the `oled/device.py` script. There are two device classes and usage is very simple if you have ever used Pillow or PIL.

First, import and initialise the device:

```
from oled.serial import i2c
from oled.device import ssd1306, sh1106
from oled.render import canvas

# rev.1 users set port=0
serial = i2c(port=1, address=0x3C)

# substitute sh1106(...) below if using that device
device = ssd1306(serial)
```

The display device should now be configured for use. The specific `ssd1306` or `sh1106` classes both expose a `display()` method which takes a 1-bit depth image. However, for most cases, for drawing text and graphics primitives, the canvas class should be used as follows:

```
with canvas(device) as draw:
    draw.rectangle((0, 0, device.width, device.height), outline="white", fill="black")
    draw.text((30, 40), "Hello World", fill="white")
```

The *`oled.render.canvas`* class automatically creates an `PIL.ImageDraw` object of the correct dimensions and bit depth suitable for the device, so you may then call the usual Pillow methods to draw onto the canvas.

As soon as the with scope is ended, the resultant image is automatically flushed to the device's display memory and the `PIL.ImageDraw` object is garbage collected.

Note: Any of the standard `PIL.ImageColor` color formats may be used, but since the OLED is monochrome, only the HTML color names `"black"` and `"white"` values should really be used.

2.1 Examples

After installing the library, enter the `examples` directory and try running the following examples:

Example	Description
bounce.py	Display a bouncing ball animation and frames per second
carousel.py	Showcase viewport and hotspot functionality
clock.py	An analog clockface with date & time
crawl.py	A vertical scrolling demo, which should be familiar
demo.py	Use misc draw commands to create a simple image
invaders.py	Space Invaders demo
maze.py	Maze generator
perfloop.py	Simpel benchmarking utility to measure performance
pi_logo.py	Display the Raspberry Pi logo (loads image as .png)
sys_info.py	Display basic system information

By default, all the examples will asume I2C port 1, address `0x3C` and the `ssd1306` driver. If you need to use a different setting, these can be specified on the command line - each program can be invoked with a `--help` flag to show the options:

```
$ python pi_logo.py -h
usage: pi_logo.py [-h] [--display DISPLAY] [--width WIDTH] [--height HEIGHT]
                  [--interface INTERFACE] [--i2c-port I2C_PORT]
                  [--i2c-address I2C_ADDRESS] [--spi-port SPI_PORT]
                  [--spi-device SPI_DEVICE] [--spi-bus-speed SPI_BUS_SPEED]
                  [--bcm-data-command BCM_DATA_COMMAND]
                  [--bcm-reset BCM_RESET] [--transform TRANSFORM]
                  [--scale SCALE] [--mode MODE] [--duration DURATION]
                  [--loop LOOP] [--max-frames MAX_FRAMES]

oled arguments

optional arguments:
  -h, --help            show this help message and exit
  --display DISPLAY, -d DISPLAY
                        Display type, one of: ssd1306, sh1106, capture,
                        pygame, gifanim (default: ssd1306)
  --width WIDTH         Width of the device in pixels (default: 128)
  --height HEIGHT       Height of the device in pixels (default: 64)
  --interface INTERFACE, -i INTERFACE
                        Serial interface type, one of: i2c, spi (default: i2c)
  --i2c-port I2C_PORT   I2C bus number (default: 1)
  --i2c-address I2C_ADDRESS
                        I2C display address (default: 0x3C)
  --spi-port SPI_PORT   SPI port number (default: 0)
  --spi-device SPI_DEVICE
                        SPI device (default: 0)
  --spi-bus-speed SPI_BUS_SPEED
                        SPI max bus speed (Hz) (default: 8000000)
  --bcm-data-command BCM_DATA_COMMAND
                        BCM pin for D/C RESET (SPI devices only) (default: 24)
  --bcm-reset BCM_RESET
                        BCM pin for RESET (SPI devices only) (default: 25)
  --transform TRANSFORM
                        Scaling transform to apply, one of: none, identity,
                        scale2x, smoothscale (emulator only) (default:
                        scale2x)
```

```
  --scale SCALE         Scaling factor to apply (emulator only) (default: 2)
  --mode MODE           Colour mode, one of: 1, RGB, RGBA (emulator only)
                        (default: RGB)
  --duration DURATION   Animation frame duration (gifanim emulator only)
                        (default: 0.01)
  --loop LOOP           Repeat loop, zero=forever (gifanim emulator only)
                        (default: 0)
  --max-frames MAX_FRAMES
                        Maximum frames to record (gifanim emulator only)
                        (default: None)
```

Note:

1. Substitute `python3` for `python` in the above examples if you are using python3.
2. `python-dev` (apt-get) and `psutil` (pip/pip3) are required to run the `sys_info.py` example. See install instructions for the exact commands to use.

2.2 Emulators

There are three display emulators available for running code against, for debugging and screen capture functionality:

- The `oled.device.capture` device will persist a numbered PNG file to disk every time its `display` method is called.
- The `oled.device.gifanim` device will record every image when its `display` method is called, and on program exit (or Ctrl-C), will assemble the images into an animated GIF.
- The `oled.device.pygame` device uses the `pygame` library to render the displayed image to a pygame display surface.

Invoke the demos with:

```
$ python examples/clock.py -d capture
```

or:

```
$ python examples/clock.py -d pygame
```

Note: *Pygame* is required to use any of the emulated devices, but it is **NOT** installed as a dependency by default, and so must be manually installed before using any of these emulation devices.

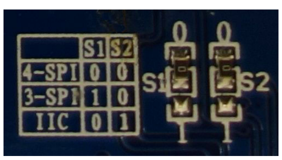
CHAPTER 3

Hardware

3.1 Identifying your serial interface

You can determine if you have an I2C or a SPI interface by counting the number of pins on your card. An I2C display will have 4 pins while an SPI interface will have 6 or 7 pins.

If you have a SPI display, check the back of your display for a configuration such as this:



For this display, the two 0 Ohm (jumper) resistors have been connected to "0" and the table shows that "0 0" is 4-wire SPI. That is the type of connection that is currently supported by the SPI mode of this library.

A list of tested devices can be found in the wiki.

3.2 I2C vs. SPI

If you have not yet purchased your display, you may be wondering if you should get an I2C or SPI display. The basic tradeoff is that I2C will be easier to connect because it has fewer pins while SPI may have a faster display update rate due to running at a higher frequency and having less overhead (see benchmarks).

3.3 Tips for connecting the display

- If you don't want to solder directly on the Pi, get 2.54mm 40 pin female single row headers, cut them to length, push them onto the Pi pins, then solder wires to the headers.
- If you need to remove existing pins to connect wires, be careful to heat each pin thoroughly, or circuit board traces may be broken.

- Triple check your connections. In particular, do not reverse VCC and GND.

3.4 Pre-requisites

3.4.1 I2C

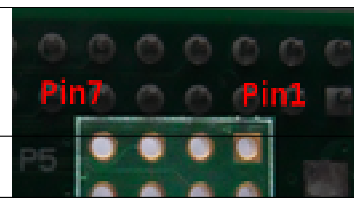
The P1 header pins should be connected as follows:

OLED Pin	Name	Remarks	RPi Pin	RPi Function
1	GND	Ground	P01-6	GND
2	VCC	+3.3V Power	P01-1	3V3
3	SCL	Clock	P01-5	GPIO 3 (SCL)
4	SDA	Data	P01-3	GPIO 2 (SDA)

You can also solder the wires directly to the underside of the RPi GPIO pins.

See also:

Alternatively, on rev.2 RPi's, right next to the male pins of the P1 header, there is a bare P5 header which features I2C channel 0, although this doesn't appear to be initially enabled and may be configured for use with the Camera module.

OLED Pin	Name	Remarks	RPi Pin	RPi Function	Location
1	GND	Ground	P5-07	GND	
2	VCC	+3.3V Power	P5-02	3V3	
3	SCL	Clock	P5-04	GPIO 29 (SCL)	
4	SDA	Data	P5-03	GPIO 28 (SDA)	

Ensure that the I2C kernel driver is enabled:

```
$ dmesg | grep i2c
[    4.925554] bcm2708_i2c 20804000.i2c: BSC1 Controller at 0x20804000 (irq 79) (baudrate 100000)
[    4.929325] i2c /dev entries driver
```

or:

```
$ lsmod | grep i2c
i2c_dev                 5769  0
i2c_bcm2708             4943  0
regmap_i2c              1661  3 snd_soc_pcm512x,snd_soc_wm8804,snd_soc_core
```

If you have no kernel modules listed and nothing is showing using `dmesg` then this implies the kernel I2C driver is not loaded. Enable the I2C as follows:

```
$ sudo raspi-config
> Advanced Options > A7 I2C
```

After rebooting re-check that the `dmesg | grep i2c` command shows whether I2C driver is loaded before proceeding. You can also enable I2C manually if the `raspi-config` utility is not available.

Optionally, to improve performance, increase the I2C baudrate from the default of 100KHz to 400KHz by altering `/boot/config.txt` to include:

```
dtparam=i2c_arm=on,i2c_baudrate=400000
```

Then reboot.

Next, add your user to the *i2c* group and install `i2c-tools`:

```
$ sudo usermod -a G i2c pi
$ sudo apt-get install i2c-tools
```

Logout and in again so that the group membership permissions take effect, and then check that the device is communicating properly (if using a rev.1 board, use 0 for the bus, not 1):

```
$ i2cdetect -y 1
       0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
  00:          -- -- -- -- -- -- -- -- -- -- -- -- --
  10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
  20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
  30: -- -- -- -- -- -- -- -- -- -- -- UU 3c -- -- --
  40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
  50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
  60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
  70: -- -- -- -- -- -- -- --
```

According to the man-page, "UU" means that probing was skipped, because the address was in use by a driver. It suggest that there is a chip at that address. Indeed the documentation for the device indicates it uses two addresses.

3.4.2 SPI

The GPIO pins used for this SPI connection are the same for all versions of the Raspberry Pi, up to and including the Raspberry Pi 3 B.

OLED Pin	Name	Remarks	RPi Pin	RPi Function
1	VCC	+3.3V Power	P01-17	3V3
2	GND	Ground	P01-20	GND
3	D0	Clock	P01-23	GPIO 11 (SCLK)
4	D1	MOSI	P01-19	GPIO 10 (MOSI)
5	RST	Reset	P01-22	GPIO 25
6	DC	Data/Command	P01-18	GPIO 24
7	CS	Chip Select	P01-24	GPIO 8 (CE0)

Note:

- When using the 4-wire SPI connection, Data/Command is an "out of band" signal that tells the controller if you're sending commands or display data. This line is not a part of SPI and the library controls it with a separate GPIO pin. With 3-wire SPI and I2C, the Data/Command signal is sent "in band".
- If you're already using the listed GPIO pins for Data/Command and/or Reset, you can select other pins and pass a `bcm_DC` and/or a `bcm_RST` argument specifying the new *BCM* pin numbers in your serial interface create call.
- The use of the terms 4-wire and 3-wire SPI are a bit confusing because, in most SPI documentation, the terms are used to describe the regular 4-wire configuration of SPI and a 3-wire mode where the input and ouput lines, MOSI and MISO, have been combined into a single line called SISO. However, in the context of these OLED controllers, 4-wire means MOSI + Data/Command and 3-wire means Data/Command sent as an extra bit over MOSI.
- Because CS is connected to CE0, the display is available on SPI port 0. You can connect it to CE1 to have it available on port 1. If so, pass `port=1` in your serial interface create call.

Enable the SPI port:

```
$ sudo raspi-config
> Advanced Options > A6 SPI
```

If `raspi-config` is not available, enabling the SPI port can be done manually.

Ensure that the SPI kernel driver is enabled:

```
$ ls -l /dev/spi*
crw-rw---- 1 root spi 153, 0 Nov 25 08:32 /dev/spidev0.0
crw-rw---- 1 root spi 153, 1 Nov 25 08:32 /dev/spidev0.1
```

or:

```
$ lsmod | grep spi
spi_bcm2835             6678  0
```

Then add your user to the *spi* and *gpio* groups:

```
$ sudo usermod -a G spi pi
$ sudo usermod -a G gpio pi
```

Log out and back in again to ensure that the group permissions are applied successfully.

CHAPTER 4

Installation

Warning: Ensure that the *Pre-requisites* from the previous section have been performed, checked and tested before proceeding.

Note: The library has been tested against Python 2.7, 3.4 and 3.5.

For **Python3** installation, substitute the following in the instructions below.

- `pip pip3`,
- `python python3`,
- `python-dev python3-dev`,
- `python-pip python3-pip`.

It was *originally* tested with Raspbian on a rev.2 model B, with a vanilla kernel version 4.1.16+, and has subsequently been tested on Raspberry Pi model A, model B2 and 3B (Debian Jessie) and OrangePi Zero (Armbian Jessie).

4.1 From PyPI

Note: This is the preferred installation mechanism.

Install the latest version of the library directly from PyPI:

```
$ sudo apt-get install python-dev python-pip libfreetype6-dev libjpeg8-dev libsdl1.2-
↪dev
$ sudo pip install --upgrade ssd1306
```

4.2 From source

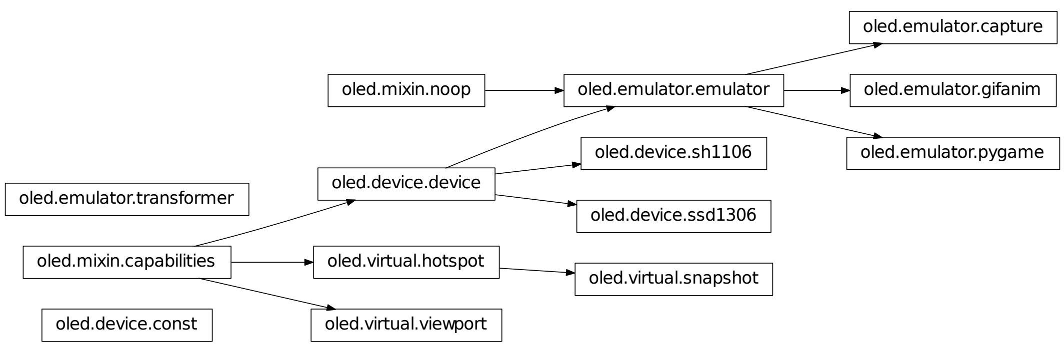
For Python 2, from the bash prompt, enter:

```
$ sudo apt-get install python-dev python-pip libfreetype6-dev libjpeg8-dev libsdl1.2-
↪dev
$ sudo python setup.py install
```

CHAPTER 5

API Documentation

OLED display driver for SSD1306 and SH1106 devices.



5.1 `oled.device`

class `oled.device.`**`const`**

`CHARGEPUMP` = **141**

`COLUMNADDR` = **33**

`COMSCANDEC` = **200**

`COMSCANINC` = **192**

`DISPLAYALLON` = **165**

DISPLAYALLON_RESUME = 164

DISPLAYOFF = 174

DISPLAYON = 175

EXTERNALVCC = 1

INVERTDISPLAY = 167

MEMORYMODE = 32

NORMALDISPLAY = 166

PAGEADDR = 34

SEGREMAP = 160

SETCOMPINS = 218

SETCONTRAST = 129

SETDISPLAYCLOCKDIV = 213

SETDISPLAYOFFSET = 211

SETHIGHCOLUMN = 16

SETLOWCOLUMN = 0

SETMULTIPLEX = 168

SETPRECHARGE = 217

SETSEGMENTREMAP = 161

SETSTARTLINE = 64

SETVCOMDETECT = 219

SWITCHCAPVCC = 2

class `oled.device.`**`device`**(*serial_interface=None*)

Bases: `oled.mixin.capabilities`

Base class for OLED driver classes

`clear`()

Initializes the device memory with an empty (blank) image.

`command`(**cmd*)

Sends a command or sequence of commands through to the delegated serial interface.

`data`(*data*)

Sends a data byte or sequence of data bytes through to the delegated serial interface.

`hide`()

Switches the display mode OFF, putting the device in low-power sleep mode.

`show`()

Sets the display mode ON, waking the device out of a prior low-power sleep mode.

class `oled.device.`**`sh1106`**(*serial_interface=None*, *width=128*, *height=64*)

Bases: `oled.device.device`

Encapsulates the serial interface to the SH1106 OLED display hardware. On creation, an initialization sequence is pumped to the display to properly configure it. Further control commands can then be called to affect the brightness and other settings.

> **Warning:** Direct use of the `command()` and `data()` methods are discouraged: Screen updates should be effected through the *display()* method, or preferably with the *oled.render.canvas* context manager.

display (*image*)
Takes a 1-bit `PIL.Image` and dumps it to the SH1106 OLED display.

class `oled.device.`**ssd1306**(*serial_interface=None*, *width=128*, *height=64*)
Bases: *oled.device.device*

Encapsulates the serial interface to the SSD1306 OLED display hardware. On creation, an initialization sequence is pumped to the display to properly configure it. Further control commands can then be called to affect the brightness and other settings.

> **Warning:** Direct use of the `command()` and `data()` methods are discouraged: Screen updates should be effected through the *display()* method, or preferably with the *oled.render.canvas* context manager.

display (*image*)
Takes a 1-bit `PIL.Image` and dumps it to the SSD1306 OLED display.

5.2 oled.emulator

class `oled.emulator.`**capture**(*width=128*, *height=64*, *mode='RGB'*, *transform='scale2x'*, *scale=2*, *file_template='oled_{0:06}.png'*, ***kwargs*)
Bases: *oled.emulator.emulator*

Pseudo-device that acts like an OLED display, except that it writes the image to a numbered PNG file when the *display()* method is called.

While the capability of an OLED device is monochrome, there is no limitation here, and hence supports 24-bit color depth.

display (*image*)
Takes a `PIL.Image` and dumps it to a numbered PNG file.

class `oled.emulator.`**emulator**(*width*, *height*, *mode*, *transform*, *scale*)
Bases: *oled.mixin.noop*, *oled.device.device*

Base class for emulated OLED driver classes

to_surface (*image*)
Converts a `PIL.Image` into a `pygame.Surface`, transforming it according to the `transform` and `scale` constructor arguments.

class `oled.emulator.`**gifanim**(*width=128*, *height=64*, *mode='RGB'*, *transform='scale2x'*, *scale=2*, *filename='oled_anim.gif'*, *duration=0.01*, *loop=0*, *max_frames=None*, ***kwargs*)
Bases: *oled.emulator.emulator*

Pseudo-device that acts like an OLED display, except that it collects the images when the *display()* method is called, and on exit, assembles them into an animated GIF image.

While the capability of an OLED device is monochrome, there is no limitation here, and hence supports 24-bit color depth, albeit with an indexed color palette.

display (*image*)
Takes an image, scales it according to the nominated transform, and stores it for later building into an animated GIF.

write_animation ()

class oled.emulator.**pygame** (*width=128*, *height=64*, *mode='RGB'*, *transform='scale2x'*, *scale=2*, *frame_rate=60*, ***kwargs*)
Bases: `oled.emulator.emulator`

Pseudo-device that acts like an OLED display, except that it renders to an displayed window. The frame rate is limited to 60FPS (much faster than a Raspberry Pi can acheive, but this can be overridden as necessary).

While the capability of an OLED device is monochrome, there is no limitation here, and hence supports 24-bit color depth.

`pygame` is used to render the emulated display window, and it's event loop is checked to see if the ESC key was pressed or the window was dismissed: if so `sys.exit()` is called.

display (*image*)
Takes a `PIL.Image` and renders it to a pygame display surface.

class oled.emulator.**transformer** (*pygame*, *width*, *height*, *scale*)
Bases: `object`

Helper class used to dispatch transformation operations.

identity (*surface*)
Fast scale operation that does not sample the results

none (*surface*)
No-op transform - used when `scale` = 1

scale2x (*surface*)
Scales using the AdvanceMAME Scale2X algorithm which does a 'jaggie-less' scale of bitmap graphics.

smoothscale (*surface*)
Smooth scaling using MMX or SSE extensions if available

5.3 oled.mixin

class oled.mixin.**capabilities**
Bases: `object`

capabilities (*width*, *height*, *mode='1'*)

class oled.mixin.**noop**
Bases: `object`

command (**cmd*)

data (*data*)

5.4 oled.render

class oled.render.**canvas** (*device*)
A canvas returns a properly-sized `PIL.ImageDraw` object onto which the caller can draw upon. As soon as the with-block completes, the resultant image is flushed onto the device.

5.5 oled.serial

class `oled.serial.`**`i2c`**(*bus=None*, *port=1*, *address=60*)

Bases: `object`

Wrap an I2C interface to provide data and command methods.

Note:

1. Only one of `bus` OR `port` arguments should be supplied; if both are, then `bus` takes precedence.
2. If `bus` is provided, there is an implicit expectation that it has already been opened.

`cleanup`()

Clean up I2C resources

`command`(**cmd*)

Sends a command or sequence of commands through to the I2C address - maximum allowed is 32 bytes in one go.

`data`(*data*)

Sends a data byte or sequence of data bytes through to the I2C address - maximum allowed in one transaction is 32 bytes, so if data is larger than this, it is sent in chunks.

class `oled.serial.`**`spi`**(*spi=None*, *gpio=None*, *port=0*, *device=0*, *bus_speed_hz=8000000*, *bcm_DC=24*, *bcm_RST=25*)

Bases: `object`

Wraps an SPI interface to provide data and command methods.

- The DC pin (Data/Command select) defaults to GPIO 24 (BCM).
- The RST pin (Reset) defaults to GPIO 25 (BCM).

`cleanup`()

Clean up SPI & GPIO resources

`command`(**cmd*)

Sends a command or sequence of commands through to the SPI device.

`data`(*data*)

Sends a data byte or sequence of data bytes through to the SPI device.

5.6 oled.threadpool

class `oled.threadpool.`**`threadpool`**(*num_threads*)

Pool of threads consuming tasks from a queue

`add_task`(*func*, **args*, ***kargs*)

Add a task to the queue

`wait_completion`()

Wait for completion of all the tasks in the queue

class `oled.threadpool.`**`worker`**(*tasks*)

Bases: `threading.Thread`

Thread executing tasks from a given tasks queue

run()

5.7 oled.virtual

oled.virtual.**calc_bounds**(*xy*, *entity*)
For an entity with width and height attributes, determine the bounding box if were positioned at (x, y).

class oled.virtual.**hotspot**(*width*, *height*, *draw_fn=None*)
Bases: *oled.mixin.capabilities*

A hotspot (*a place of more than usual interest, activity, or popularity*) is a live display which may be added to a virtual viewport - if the hotspot and the viewport are overlapping, then the *update()* method will be automatically invoked when the viewport is being refreshed or its position moved (such that an overlap occurs).

You would either:

- create a `hotspot` instance, suppling a render function (taking an `PIL.ImageDraw` object, `width` & `height` dimensions. The render function should draw within a bounding box of (0, 0, width, height), and render a full frame.
- sub-class `hotspot` and override the :func:`should_redraw` and *update()* methods. This might be more useful for slow-changing values where it is not necessary to update every refresh cycle, or your implementation is stateful.

paste_into(*image*, *xy*)

should_redraw()
Override this method to return true or false on some condition (possibly on last updated member variable) so that for slow changing hotspots they are not updated too frequently.

update(*draw*)

oled.virtual.**range_overlap**(*a_min*, *a_max*, *b_min*, *b_max*)
Neither range is completely greater than the other

class oled.virtual.**snapshot**(*width*, *height*, *draw_fn=None*, *interval=1.0*)
Bases: *oled.virtual.hotspot*

A snapshot is a *type of* hotspot, but only updates once in a given interval, usually much less frequently than the viewport requests refresh updates.

paste_into(*image*, *xy*)

should_redraw()
Only requests a redraw after `interval` seconds have elapsed

class oled.virtual.**viewport**(*device*, *width*, *height*)
Bases: *oled.mixin.capabilities*

add_hotspot(*hotspot*, *xy*)

display(*image*)

is_overlapping_viewport(*hotspot*, *xy*)
Checks to see if the hotspot at position (x, y) is (at least partially) visible according to the position of the viewport

refresh()

set_position(*xy*)

CHAPTER 6

References

- https://learn.adafruit.com/monochrome-oled-breakouts
- https://github.com/adafruit/Adafruit_Python_SSD1306
- http://www.dafont.com/bitmap.php
- http://raspberrypi.znix.com/hipidocs/topic_i2cbus_2.htm
- http://martin-jones.com/2013/08/20/how-to-get-the-second-raspberry-pi-i2c-bus-to-work/
- https://projects.drogon.net/understanding-spi-on-the-raspberry-pi/
- https://pinout.xyz/
- https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi
- http://code.activestate.com/recipes/577187-python-thread-pool/

CHAPTER 7

Contributing

Pull requests (code changes / documentation / typos / feature requests / setup) are gladly accepted. If you are intending to introduce some large-scale changes, please get in touch first to make sure we're on the same page: try to include a docstring for any new method or class, and keep method bodies small, readable and PEP8-compliant. Add tests and strive to keep the code coverage levels high.

7.1 GitHub

The source code is available to clone at: https://github.com/rm-hull/ssd1306.git

7.2 Contributors

- Thijs Triemstra (@thijstriemstra)
- Christoph Handel (@fragfutter)
- Boeeerb (@Boeeerb)
- xes (@xes)
- Roger Dahl (@rogerdahl)
- Václav Šmilauer (@eudoxos)

CHAPTER 8

ChangeLog

Version	Description	Date
Upcoming	• Viewport/scrolling support • Remove pygame as an install dependency in setup • Ensure SH1106 device collapses color images to monochrome • Documentation updates	
1.2.0	• Add support for 128x32, 96x16 OLED screens (SSD1306 chipset only) • Fix boundary condition error when supplying max-frames to gifanim • Bit pattern calc rework when conveting color -> monochrome • Approx 20% performance improvement in `display` method	2016/12/08
1.1.0	• Add animated-GIF emulator • Add color-mode flag to emulator • Fix regression in SPI interface • Rename emulator transform option 'scale' to 'identity'	2016/12/05
1.0.0	• Add HQX scaling to capture and pygame emulators • SPI support (**NOTE:** contains breaking changes) • Improve benchmarking examples • Fix resource leakage & noops	2016/12/03

CHAPTER 9

The MIT License (MIT)

Python Module Index

o

Index

P

R

S

T

U

V

W