Luma.OLED Documentation

Release 3.1.0

Richard Hull and contributors

Dec 22, 2018

Contents

CHAPTER 1

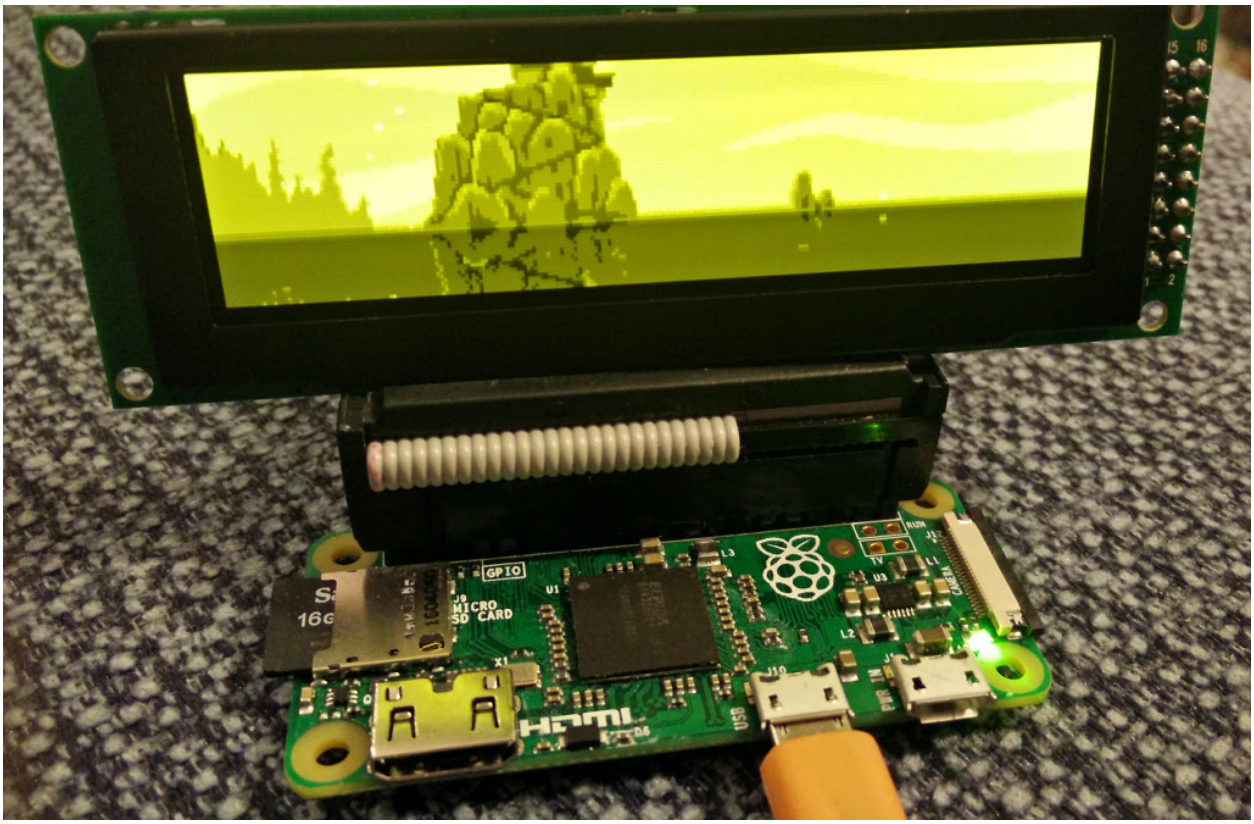
Introduction

Interfacing OLED matrix displays with the SSD1306, SSD1309, SSD1322, SSD1325, SSD1327, SSD1331, SSD1351 or SH1106 driver in Python 2 or 3 using I2C/SPI on the Raspberry Pi and other linux-based single-board computers: the library provides a Pillow-compatible drawing canvas, and other functionality to support:

- scrolling/panning capability,
- terminal-style printing,
- state management,
- color/greyscale (where supported),
- dithering to monochrome

The SSD1306 display pictured below is 128 x 64 pixels, and the board is *tiny*, and will fit neatly inside the RPi case.


Ld:0.2 0.2 0.2 Up: 3:00:30
Mem: 225M 86%
SD: 4G 31%
wlan0: Tx6M Rx4M
HDMI


GPIO
MICRO SD CARD
HDMI

See also:

Further technical information for the specific implemented devices can be found in the following datasheets:

- SSD1306
- SSD1309
- SSD1322
- SSD1325
- SSD1327
- SSD1331
- SSD1351
- SH1106

Benchmarks for tested devices can be found in the wiki.

As well as display drivers for various physical 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 luma.examples git repository:

CHAPTER 2

Python usage

OLED displays can be driven with python using the varous implementations in the *luma.oled.device* package. There are several device classes available and usage is very simple if you have ever used Pillow or PIL.

First, import and initialise the device:

```
from luma.core.interface.serial import i2c, spi
from luma.core.render import canvas
from luma.oled.device import ssd1306, ssd1309, ssd1325, ssd1331, sh1106

# rev.1 users set port=0
# substitute spi(device=0, port=0) below if using that interface
serial = i2c(port=1, address=0x3C)

# substitute ssd1331(...) or sh1106(...) below if using that device
device = ssd1306(serial)
```

The display device should now be configured for use. The device classes all expose a `display()` method which takes an image with attributes consistent with the capabilities of the device. 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(device.bounding_box, outline="white", fill="black")
    draw.text((30, 40), "Hello World", fill="white")
```

The `luma.core.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.

2.1 Color Model

Any of the standard `PIL.ImageColor` color formats may be used, but since the SSD1306 and SH1106 OLEDs are monochrome, only the HTML color names `"black"` and `"white"` values should really be used; in fact, by default, any value *other* than black is treated as white. The `luma.core.render.canvas` object does have a `dither` flag which if set to `True`, will convert color drawings to a dithered monochrome effect (see the *3d_box.py* example, below).

```
with canvas(device, dither=True) as draw:
    draw.rectangle((10, 10, 30, 30), outline="white", fill="red")
```

There is no such constraint on the SSD1331 or SSD1351 OLEDs, which features 16-bit RGB colors: 24-bit RGB images are downsized to 16-bit using a 565 scheme.

The SSD1322 and SSD1325 OLEDs both support 16 greyscale graduations: 24-bit RGB images are downsized to 4-bit using a Luma conversion which is approximately calculated as follows:

```
Y' = 0.299 R' + 0.587 G' + 0.114 B'
```

2.2 Landscape / Portrait Orientation

By default the display will be oriented in landscape mode (128x64 pixels for the SSD1306, for example). Should you have an application that requires the display to be mounted in a portrait aspect, then add a `rotate=N` parameter when creating the device:

```
from luma.core.interface.serial import i2c
from luma.core.render import canvas
from luma.oled.device import ssd1306, ssd1325, ssd1331, sh1106

serial = i2c(port=1, address=0x3C)
device = ssd1306(serial, rotate=1)

# Box and text rendered in portrait mode
with canvas(device) as draw:
    draw.rectangle(device.bounding_box, outline="white", fill="black")
    draw.text((10, 40), "Hello World", fill="white")
```

N should be a value of 0, 1, 2 or 3 only, where 0 is no rotation, 1 is rotate 90° clockwise, 2 is 180° rotation and 3 represents 270° rotation.

The `device.size`, `device.width` and `device.height` properties reflect the rotated dimensions rather than the physical dimensions.

2.3 Examples

After installing the library, head over to the luma.examples repository. Details of how to run the examples is shown in the example repo's README.

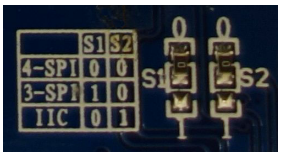
CHAPTER 3

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 trade-off 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

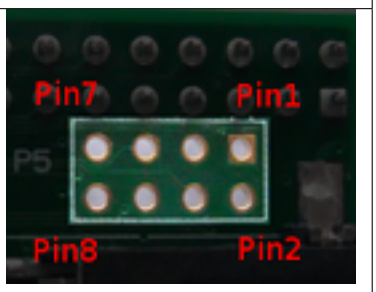
The P1 header pins should be connected as follows:

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:

1. Run `sudo raspi-config`

2. Use the down arrow to select `5 Interfacing Options`
3. Arrow down to `P5 I2C`
4. Select **yes** when it asks you to enable I2C
5. Also select **yes** when it asks about automatically loading the kernel module
6. Use the right arrow to select the **<Finish>** button

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 output 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,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, 3.5 and 3.6.

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 (both Raspbian Jessie and Stretch) models A, B2, 3B, Zero, Zero W and OrangePi Zero (Armbian Jessie).

Install the latest version of the library directly from PyPI:

```
$ sudo apt-get install python-dev python-pip libfreetype6-dev libjpeg-dev build-
↪essential
$ sudo -H pip install --upgrade luma.oled
```

If you are upgrading from a previous version, make sure to read the *upgrade* document.

CHAPTER 5

Upgrade

5.1 2.0.0

Version 2.0.0 was released on 11 January 2017: this came with a rename of the project in github from **ssd1306** to **luma.oled** to reflect the changing nature of the codebase. It introduces some structural changes to the package structure, namely breaking the library up into smaller components and renaming existing packages.

This should largely be restricted to having to update import statements only. To upgrade any existing code that uses the old package structure:

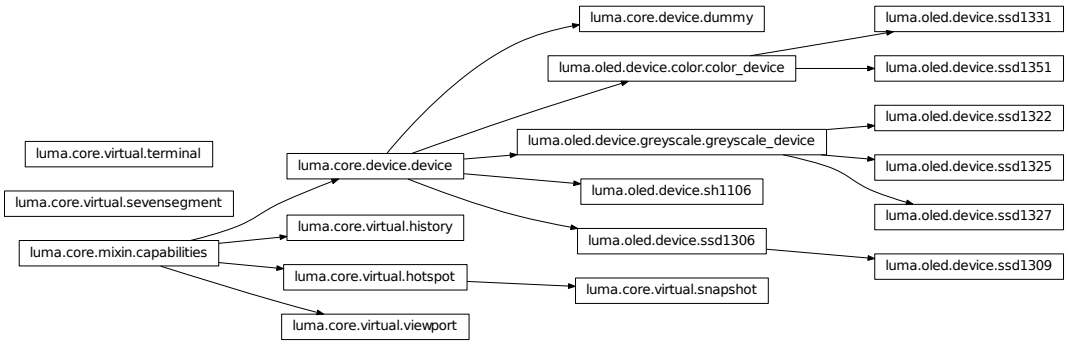
- rename instances of `oled.device` to `luma.oled.device`.
- rename any other usages of `oled.*` to `luma.core.*`.

This breaking change was necessary to be able to add different classes of devices, so that they could reuse core components.

CHAPTER 6

API Documentation

OLED display driver for SSD1306, SSD1309, SSD1322, SSD1325, SSD1327, SSD1331, SSD1351 and SH1106 devices.



6.1 luma.oled.device

Collection of serial interfaces to OLED devices.

class luma.oled.device.**ssd1306**(*serial_interface=None*, *width=128*, *height=64*, *rotate=0*, ***kwargs*)

Bases: luma.core.device.device

Serial interface to a monochrome SSD1306 OLED display.

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.

Parameters

- **serial_interface** – The serial interface (usually a luma.core.interface.serial.i2c instance) to delegate sending data and commands through.
- **width** (*int*) – The number of horizontal pixels (optional, defaults to 128).
- **height** (*int*) – The number of vertical pixels (optional, defaults to 64).
- **rotate** (*int*) – An integer value of 0 (default), 1, 2 or 3 only, where 0 is no rotation, 1 is rotate 90° clockwise, 2 is 180° rotation and 3 represents 270° rotation.

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

Assigns attributes such as `width`, `height`, `size` and `bounding_box` correctly oriented from the supplied parameters.

Parameters

- **width** (*int*) – the device width
- **height** (*int*) – the device height
- **rotate** (*int*) – an integer value of 0 (default), 1, 2 or 3 only, where 0 is no rotation, 1 is rotate 90° clockwise, 2 is 180° rotation and 3 represents 270° rotation.
- **mode** (*str*) – the supported color model, one of "1", "RGB" or "RGBA" only.

cleanup()

Attempt to switch the device off or put into low power mode (this helps prolong the life of the device), clear the screen and close resources associated with the underlying serial interface.

If `persist` is True, the device will not be switched off.

This is a managed function, which is called when the python processs is being shutdown, so shouldn't usually need be called directly in application code.

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.

contrast(*level*)

Switches the display contrast to the desired level, in the range 0-255. Note that setting the level to a low (or zero) value will not necessarily dim the display to nearly off. In other words, this method is **NOT** suitable for fade-in/out animation.

Parameters **level** (*int*) – Desired contrast level in the range of 0-255.

data(*data*)

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

display(*image*)

Takes a 1-bit `PIL.Image` and dumps it to the OLED display.

Parameters **image** (`PIL.Image`) – Image to display.

hide()

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

preprocess(*image*)

Provides a preprocessing facility (which may be overridden) whereby the supplied image is rotated according to the device's rotate capability. If this method is overridden, it is important to call the super

Parameters **image** (*PIL.Image.Image*) – An image to pre-process

Returns A new processed image

Return type PIL.Image.Image

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

class luma.oled.device.**ssd1309**(*serial_interface=None*, *width=128*, *height=64*, *rotate=0*, ***kwargs*)
Bases: *luma.oled.device.ssd1306*

Serial interface to a monochrome SSD1309 OLED display.

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.

Parameters

- **serial_interface** – The serial interface (usually a luma.core.interface.serial.spi instance) to delegate sending data and commands through.
- **width** (*int*) – The number of horizontal pixels (optional, defaults to 128).
- **height** (*int*) – The number of vertical pixels (optional, defaults to 64).
- **rotate** (*int*) – An integer value of 0 (default), 1, 2 or 3 only, where 0 is no rotation, 1 is rotate 90° clockwise, 2 is 180° rotation and 3 represents 270° rotation.

New in version 3.1.0.

capabilities(*width*, *height*, *rotate*, *mode='1'*)
Assigns attributes such as `width`, `height`, `size` and `bounding_box` correctly oriented from the supplied parameters.

Parameters

- **width** (*int*) – the device width
- **height** (*int*) – the device height
- **rotate** (*int*) – an integer value of 0 (default), 1, 2 or 3 only, where 0 is no rotation, 1 is rotate 90° clockwise, 2 is 180° rotation and 3 represents 270° rotation.
- **mode** (*str*) – the supported color model, one of "1", "RGB" or "RGBA" only.

cleanup()
Attempt to switch the device off or put into low power mode (this helps prolong the life of the device), clear the screen and close resources associated with the underlying serial interface.

If `persist` is True, the device will not be switched off.

This is a managed function, which is called when the python processs is being shutdown, so shouldn't usually need be called directly in application code.

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.

contrast(*level*)
Switches the display contrast to the desired level, in the range 0-255. Note that setting the level to a low (or zero) value will not necessarily dim the display to nearly off. In other words, this method is **NOT** suitable for fade-in/out animation.

Parameters **level** (*int*) – Desired contrast level in the range of 0-255.

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

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

Parameters **image** (`PIL.Image`) – Image to display.

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

preprocess(*image*)
Provides a preprocessing facility (which may be overridden) whereby the supplied image is rotated according to the device's rotate capability. If this method is overridden, it is important to call the super

Parameters **image** (*PIL.Image.Image*) – An image to pre-process

Returns A new processed image

Return type PIL.Image.Image

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

class luma.oled.device.**ssd1322**(*serial_interface=None*, *width=256*, *height=64*, *rotate=0*, *mode='RGB'*, *framebuffer='diff_to_previous'*, ***kwargs*)
Bases: `luma.oled.device.greyscale.greyscale_device`

Serial interface to a 4-bit greyscale SSD1322 OLED display.

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.

Parameters

- **serial_interface** – The serial interface (usually a `luma.core.interface.serial.spi` instance) to delegate sending data and commands through.
- **width** (*int*) – The number of horizontal pixels (optional, defaults to 96).
- **height** (*int*) – The number of vertical pixels (optional, defaults to 64).
- **rotate** (*int*) – An integer value of 0 (default), 1, 2 or 3 only, where 0 is no rotation, 1 is rotate 90° clockwise, 2 is 180° rotation and 3 represents 270° rotation.
- **mode** (*str*) – Supplying "1" or "RGB" effects a different rendering mechanism, either to monochrome or 4-bit greyscale.
- **framebuffer** (*str*) – Framebuffering strategy, currently values of `diff_to_previous` or `full_frame` are only supported

capabilities(*width*, *height*, *rotate*, *mode='1'*)
Assigns attributes such as `width`, `height`, `size` and `bounding_box` correctly oriented from the supplied parameters.

Parameters

- **width** (*int*) – the device width
- **height** (*int*) – the device height
- **rotate** (*int*) – an integer value of 0 (default), 1, 2 or 3 only, where 0 is no rotation, 1 is rotate 90° clockwise, 2 is 180° rotation and 3 represents 270° rotation.
- **mode** (*str*) – the supported color model, one of "1", "RGB" or "RGBA" only.

cleanup()
Attempt to switch the device off or put into low power mode (this helps prolong the life of the device), clear the screen and close resources associated with the underlying serial interface.

If `persist` is True, the device will not be switched off.

This is a managed function, which is called when the python processs is being shutdown, so shouldn't usually need be called directly in application code.

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

command(*cmd*, **args*)
Sends a command and an (optional) sequence of arguments through to the delegated serial interface. Note that the arguments are passed through as data.

contrast(*level*)
Switches the display contrast to the desired level, in the range 0-255. Note that setting the level to a low (or zero) value will not necessarily dim the display to nearly off. In other words, this method is **NOT** suitable for fade-in/out animation.

Parameters **level** (*int*) – Desired contrast level in the range of 0-255.

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

display(*image*)
Takes a 1-bit monochrome or 24-bit RGB image and renders it to the greyscale OLED display. RGB pixels are converted to 4-bit greyscale values using a simplified Luma calculation, based on *Y'=0.299R'+0.587G'+0.114B'*.

Parameters **image** (*PIL.Image.Image*) – The image to render.

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

preprocess(*image*)
Provides a preprocessing facility (which may be overridden) whereby the supplied image is rotated according to the device's rotate capability. If this method is overridden, it is important to call the super

Parameters **image** (*PIL.Image.Image*) – An image to pre-process

Returns A new processed image

Return type PIL.Image.Image

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

class luma.oled.device.**ssd1325**(*serial_interface=None*, *width=128*, *height=64*, *rotate=0*, *mode='RGB'*, *framebuffer='full_frame'*, ***kwargs*)
Bases: luma.oled.device.greyscale.greyscale_device

Serial interface to a 4-bit greyscale SSD1325 OLED display.

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.

capabilities(*width*, *height*, *rotate*, *mode='1'*)
Assigns attributes such as `width`, `height`, `size` and `bounding_box` correctly oriented from the supplied parameters.

Parameters

- **width** (*int*) – the device width

- **height** (*int*) – the device height
- **rotate** (*int*) – an integer value of 0 (default), 1, 2 or 3 only, where 0 is no rotation, 1 is rotate 90° clockwise, 2 is 180° rotation and 3 represents 270° rotation.
- **mode** (*str*) – the supported color model, one of "1", "RGB" or "RGBA" only.

cleanup()
Attempt to switch the device off or put into low power mode (this helps prolong the life of the device), clear the screen and close resources associated with the underlying serial interface.

If `persist` is True, the device will not be switched off.

This is a managed function, which is called when the python processs is being shutdown, so shouldn't usually need be called directly in application code.

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.

contrast(*level*)
Switches the display contrast to the desired level, in the range 0-255. Note that setting the level to a low (or zero) value will not necessarily dim the display to nearly off. In other words, this method is **NOT** suitable for fade-in/out animation.

Parameters **level** (*int*) – Desired contrast level in the range of 0-255.

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

display(*image*)
Takes a 1-bit monochrome or 24-bit RGB image and renders it to the greyscale OLED display. RGB pixels are converted to 4-bit greyscale values using a simplified Luma calculation, based on *Y'=0.299R'+0.587G'+0.114B'*.

Parameters **image** (*PIL.Image.Image*) – The image to render.

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

preprocess(*image*)
Provides a preprocessing facility (which may be overridden) whereby the supplied image is rotated according to the device's rotate capability. If this method is overridden, it is important to call the super

Parameters **image** (*PIL.Image.Image*) – An image to pre-process

Returns A new processed image

Return type PIL.Image.Image

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

class luma.oled.device.**ssd1327**(*serial_interface=None*, *width=128*, *height=128*, *rotate=0*, *mode='RGB'*, *framebuffer='full_frame'*, ***kwargs*)
Bases: luma.oled.device.greyscale.greyscale_device

Serial interface to a 4-bit greyscale SSD1327 OLED display.

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.

New in version 2.4.0.

capabilities(*width*, *height*, *rotate*, *mode='1'*)
Assigns attributes such as `width`, `height`, `size` and `bounding_box` correctly oriented from the supplied parameters.

Parameters

- **width** (*int*) – the device width
- **height** (*int*) – the device height
- **rotate** (*int*) – an integer value of 0 (default), 1, 2 or 3 only, where 0 is no rotation, 1 is rotate 90° clockwise, 2 is 180° rotation and 3 represents 270° rotation.
- **mode** (*str*) – the supported color model, one of "1", "RGB" or "RGBA" only.

cleanup()
Attempt to switch the device off or put into low power mode (this helps prolong the life of the device), clear the screen and close resources associated with the underlying serial interface.

If `persist` is True, the device will not be switched off.

This is a managed function, which is called when the python processs is being shutdown, so shouldn't usually need be called directly in application code.

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.

contrast(*level*)
Switches the display contrast to the desired level, in the range 0-255. Note that setting the level to a low (or zero) value will not necessarily dim the display to nearly off. In other words, this method is **NOT** suitable for fade-in/out animation.

Parameters **level** (*int*) – Desired contrast level in the range of 0-255.

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

display(*image*)
Takes a 1-bit monochrome or 24-bit RGB image and renders it to the greyscale OLED display. RGB pixels are converted to 4-bit greyscale values using a simplified Luma calculation, based on *Y'=0.299R'+0.587G'+0.114B'*.

Parameters **image** (*PIL.Image.Image*) – The image to render.

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

preprocess(*image*)
Provides a preprocessing facility (which may be overridden) whereby the supplied image is rotated according to the device's rotate capability. If this method is overridden, it is important to call the super

Parameters **image** (*PIL.Image.Image*) – An image to pre-process

Returns A new processed image

Return type PIL.Image.Image

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

class luma.oled.device.**ssd1331**(*serial_interface=None*, *width=96*, *height=64*, *rotate=0*, *framebuffer='diff_to_previous'*, ***kwargs*)

Bases: luma.oled.device.color.color_device

Serial interface to a 16-bit color (5-6-5 RGB) SSD1331 OLED display.

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.

Parameters

- **serial_interface** – The serial interface (usually a luma.core.interface.serial.spi instance) to delegate sending data and commands through.
- **width** (*int*) – The number of horizontal pixels (optional, defaults to 96).
- **height** (*int*) – The number of vertical pixels (optional, defaults to 64).
- **rotate** (*int*) – An integer value of 0 (default), 1, 2 or 3 only, where 0 is no rotation, 1 is rotate 90° clockwise, 2 is 180° rotation and 3 represents 270° rotation.
- **framebuffer** (*str*) – Framebuffering strategy, currently values of diff_to_previous or full_frame are only supported.

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

Assigns attributes such as width, height, size and bounding_box correctly oriented from the supplied parameters.

Parameters

- **width** (*int*) – the device width
- **height** (*int*) – the device height
- **rotate** (*int*) – an integer value of 0 (default), 1, 2 or 3 only, where 0 is no rotation, 1 is rotate 90° clockwise, 2 is 180° rotation and 3 represents 270° rotation.
- **mode** (*str*) – the supported color model, one of "1", "RGB" or "RGBA" only.

cleanup()

Attempt to switch the device off or put into low power mode (this helps prolong the life of the device), clear the screen and close resources associated with the underlying serial interface.

If persist is True, the device will not be switched off.

This is a managed function, which is called when the python processs is being shutdown, so shouldn't usually need be called directly in application code.

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.

contrast(*level*)

Switches the display contrast to the desired level, in the range 0-255. Note that setting the level to a low (or zero) value will not necessarily dim the display to nearly off. In other words, this method is **NOT** suitable for fade-in/out animation.

Parameters **level** (*int*) – Desired contrast level in the range of 0-255.

data(*data*)

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

display(*image*)

Renders a 24-bit RGB image to the Color OLED display.

Parameters **image** (*PIL.Image.Image*) – The image to render.

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

preprocess(*image*)
Provides a preprocessing facility (which may be overridden) whereby the supplied image is rotated according to the device's rotate capability. If this method is overridden, it is important to call the super

Parameters **image** (*PIL.Image.Image*) – An image to pre-process

Returns A new processed image

Return type PIL.Image.Image

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

class luma.oled.device.**ssd1351**(*serial_interface=None*, *width=128*, *height=128*, *rotate=0*, *framebuffer='diff_to_previous'*, *h_offset=0*, *v_offset=0*, *bgr=False*, ***kwargs*)
Bases: luma.oled.device.color.color_device

Serial interface to the 16-bit color (5-6-5 RGB) SSD1351 OLED display.

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.

Parameters

- **serial_interface** – The serial interface (usually a luma.core.interface.serial.spi instance) to delegate sending data and commands through.
- **width** (*int*) – The number of horizontal pixels (optional, defaults to 128).
- **height** (*int*) – The number of vertical pixels (optional, defaults to 128).
- **rotate** (*int*) – An integer value of 0 (default), 1, 2 or 3 only, where 0 is no rotation, 1 is rotate 90° clockwise, 2 is 180° rotation and 3 represents 270° rotation.
- **framebuffer** (*str*) – Framebuffering strategy, currently values of diff_to_previous or full_frame are only supported.
- **bgr** (*bool*) – Set to True if device pixels are BGR order (rather than RGB).
- **h_offset** (*int*) – Horizontal offset (in pixels) of screen to device memory (default: 0)
- **v_offset** – Vertical offset (in pixels) of screen to device memory (default: 0)

New in version 2.3.0.

capabilities(*width*, *height*, *rotate*, *mode='1'*)
Assigns attributes such as width, height, size and bounding_box correctly oriented from the supplied parameters.

Parameters

- **width** (*int*) – the device width
- **height** (*int*) – the device height
- **rotate** (*int*) – an integer value of 0 (default), 1, 2 or 3 only, where 0 is no rotation, 1 is rotate 90° clockwise, 2 is 180° rotation and 3 represents 270° rotation.
- **mode** (*str*) – the supported color model, one of "1", "RGB" or "RGBA" only.

cleanup()
Attempt to switch the device off or put into low power mode (this helps prolong the life of the device), clear the screen and close resources associated with the underlying serial interface.

If `persist` is True, the device will not be switched off.

This is a managed function, which is called when the python processs is being shutdown, so shouldn't usually need be called directly in application code.

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

command(*cmd*, **args*)
Sends a command and an (optional) sequence of arguments through to the delegated serial interface. Note that the arguments are passed through as data.

contrast(*level*)
Switches the display contrast to the desired level, in the range 0-255. Note that setting the level to a low (or zero) value will not necessarily dim the display to nearly off. In other words, this method is **NOT** suitable for fade-in/out animation.

Parameters **level** (*int*) – Desired contrast level in the range of 0-255.

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

display(*image*)
Renders a 24-bit RGB image to the Color OLED display.

Parameters **image** (*PIL.Image.Image*) – The image to render.

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

preprocess(*image*)
Provides a preprocessing facility (which may be overridden) whereby the supplied image is rotated according to the device's rotate capability. If this method is overridden, it is important to call the super

Parameters **image** (*PIL.Image.Image*) – An image to pre-process

Returns A new processed image

Return type PIL.Image.Image

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

class luma.oled.device.**sh1106**(*serial_interface=None*, *width=128*, *height=64*, *rotate=0*, ***kwargs*)
Bases: luma.core.device.device

Serial interface to a monochrome SH1106 OLED display.

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.

capabilities(*width*, *height*, *rotate*, *mode='1'*)
Assigns attributes such as `width`, `height`, `size` and `bounding_box` correctly oriented from the supplied parameters.

Parameters

- **width** (*int*) – the device width
- **height** (*int*) – the device height

- **rotate** (*int*) – an integer value of 0 (default), 1, 2 or 3 only, where 0 is no rotation, 1 is rotate 90° clockwise, 2 is 180° rotation and 3 represents 270° rotation.
- **mode** (*str*) – the supported color model, one of "1", "RGB" or "RGBA" only.

cleanup()
Attempt to switch the device off or put into low power mode (this helps prolong the life of the device), clear the screen and close resources associated with the underlying serial interface.

If `persist` is True, the device will not be switched off.

This is a managed function, which is called when the python processs is being shutdown, so shouldn't usually need be called directly in application code.

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.

contrast(*level*)
Switches the display contrast to the desired level, in the range 0-255. Note that setting the level to a low (or zero) value will not necessarily dim the display to nearly off. In other words, this method is **NOT** suitable for fade-in/out animation.

Parameters **level** (*int*) – Desired contrast level in the range of 0-255.

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

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

Parameters **image** (`PIL.Image`) – Image to display.

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

preprocess(*image*)
Provides a preprocessing facility (which may be overridden) whereby the supplied image is rotated according to the device's rotate capability. If this method is overridden, it is important to call the super

Parameters **image** (*PIL.Image.Image*) – An image to pre-process

Returns A new processed image

Return type PIL.Image.Image

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

CHAPTER 7

Troubleshooting

7.1 Corrupted display due to using incorrect driver

Using the SSD1306 driver on a display that has a SH1106 controller can result in the display showing a small section of the expected output with the rest of the display consisting of semi-random pixels (uninitialized memory).

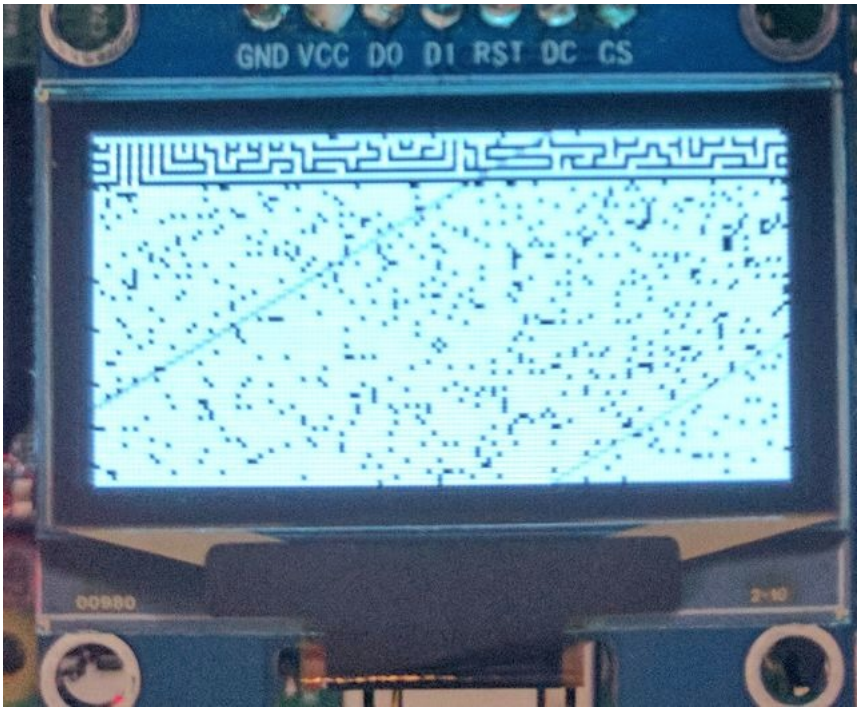


Fig. 1: Display corruption due to using driver for incorrect controller when running the *maze.py* example

This is due to differences in required initialization sequences and how memory is mapped in the two controllers.

The included examples default to the SSD1306 driver. To use the SH1106 driver instead, include the *–display sh1106* command line switch. To use the SSH1106 driver in code, use the *luma.oled.device.sh1106* serial interface class.

CHAPTER 8

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 9

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.

9.1 GitHub

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

9.2 Contributors

- Thijs Triemstra (@thijstriemstra)
- Christoph Handel (@fragfutter)
- Boeeerb (@Boeeerb)
- xes (@xes)
- Roger Dahl (@rogerdahl)
- Václav Šmilauer (@eudoxos)
- Claus Bjerre (@bjerrep)
- Vx Displays LLC (@VxGeeks)
- Christopher Arndt (@SpotlightKid)
- Sascha Walther (@leragequit)
- Marcus Kellerman (@sharkusk)
- Phil Howard (@gadgetoid)

CHAPTER 10

ChangeLog

Version	Description	Date
3.1.0	• Add support for 128x64 monochrome OLED (SSD1309)	2018/12/21
3.0.1	• Fix bug where SSD1325/1327 didn't handle `framebuffer` properly	2018/12/21
3.0.0	• **BREAKING** Fix SSD1351 init sequence didn't set RGB/BGR color order properly. Users of this device should verify proper color rendering and add `bgr=True` if blue/red color components appear to be reversed • Device consolidation - greyscale and colour SSD13xx devices now share common base classes.	2018/12/02

Continued on next page

Table 1 – continued from previous page

Version	Description	Date
2.5.1	• Fix bug where SSD1331/1351 didn't render green accurately	2018/09/14
2.5.0	• Add support form 128x128 Monochrome OLED (SH1106) (by @Gadgetoid) • Dependency and documentation updates • Minor packaging changes	2018/09/07
2.4.1	• Fix bug where SSD1327 init sequence exceeds serial command size	2018/05/28
2.4.0	• Support for 128x128 4-bit OLED (SSD1327)	2018/04/18
2.3.2	• Support for 96x96 color OLED (SSD1351)	2018/03/03
2.3.1	• Changed version number to inside `luma/oled/__init__.py`	2017/11/23
2.3.0	• Support for 128x128 color OLED (SSD1351)	2017/10/30
2.2.12	• Explicitly state 'UTF-8' encoding in setup when reading files	2017/10/18
2.2.11	• Update dependencies • Additional troubleshooting documentation	2017/09/19
2.2.10	• Add support for 128x32 mode for SH1106	2017/05/01

Continued on next page

Table 1 – continued from previous page

Version	Description	Date
2.2.9	• luma.core 0.9.0 or newer is required now • Documentation amends	2017/04/22
2.2.8	• SSD1331 & SSD1322 framebuffer & API docstrings	2017/04/13
2.2.7	• Add support for 64x32 SSD1306 OLED	2017/04/12
2.2.6	• Add support for 64x48 SSD1306 OLED	2017/03/30
2.2.5	• Restrict exported Python symbols from `luma.oled.device`	2017/03/02
2.2.4	• Tweaked SSD1325 init settings & replaced constants • Update dependencies	2017/02/17
2.2.3	• Monochrome rendering on SSD1322 & SSD1325	2017/02/14
2.2.2	• SSD1325 performance improvements (perfloop: 25.50 –> 34.31 FPS) • SSD1331 performance improvements (perfloop: 34.64 –> 51.89 FPS)	2017/02/02
2.2.1	• Support for 256x64 4-bit greyscale OLED (SSD1322) • Improved API documentation (shows inherited members)	2017/01/29
2.1.0	• Simplify/optimize SSD1306 display logic	2017/01/22

Continued on next page

Table 1 – continued from previous page

Version	Description	Date
2.0.1	• Moved examples to separate git repo • Add notes about breaking changes	2017/01/15
2.0.0	• Package rename to `luma.oled` (**Note:** Breaking changes)	2017/01/11
1.5.0	• Performance improvements for SH1106 driver (2x frame rate!) • Support for 4-bit greyscale OLED (SSD1325) • Landscape/portrait orientation with rotate=N parameter	2017/01/09
1.4.0	• Add savepoint/restore functionality • Add terminal functionality • Canvas image dithering • Additional & improved examples • Load config settings from file (for examples) • Universal wheel distribution • Improved/simplified error reporting • Documentation updates	2016/12/23
1.3.1	• Add ability to adjust brightness of screen • Fix for wrong value NORMALDISPLAY for SSD1331 device	2016/12/11

Continued on next page

Table 1 – continued from previous page

Version	Description	Date
1.3.0	• Support for 16-bit color OLED (SSD1331) • Viewport/scrolling support • Remove pygame as an install dependency in setup • Ensure SH1106 device collapses color images to monochrome • Fix for emulated devices: do not need cleanup • Fix to allow gifanim emulator to process 1-bit images • Establish a single threadpool for all virtual viewports • Fix issue preventing multiple threads from running concurrently • Documentation updates	2016/12/11
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

Continued on next page

Table 1 – continued from previous page

Version	Description	Date
1.0.0	• Add HQX scaling to capture and pygame emulators • SPI support (**NOTE:** contains breaking changes) • Improve benchmarking examples • Fix resource leakage & noops on emulated devices • Additional tests	2016/12/03
0.3.5	• Pygame-based device emulator & screen capture device emulator • Add bouncing balls demo, clock & Space Invaders examples • Auto cleanup on exit • Add `bounding_box` attribute to devices • Demote buffer & pages attributes to "internal use" only • Replaced SH1106 data sheet with version that is not "preliminary" • Add font attribution • Tests for SSD1306 & SSH1106 devices • Add code coverage & upload to coveralls.io • flake8 code compliance • Documentation updates	2016/11/30
0.3.4	• Performance improvements - render speeds ~2x faster • Documentation updates	2016/11/15
0.3.3	• Add PyPi badge • Use smbus2	2016/11/15

Continued on next page

Table 1 – continued from previous page

Version	Description	Date
0.3.2	• Fix bug in maze example (integer division on python 3) • Use latest pip • Add tox & travis config (+ badge) • Add RTFD config • Documentation updates	2016/11/13
0.3.1	• Adjust requirements (remove smbus) • Default RTFD theme • Documentation updates	2016/11/13
0.3.0	• Allow SMBus implementation to be supplied • Add show, hide and clear methods • Catch & rethrow `IOError` exceptions • Fix error in 'hello world' example • Cleanup imports • Allow setting width/height • Documentation updates	2016/11/13
0.2.0	• Add Python 3 support • Add options to demos • Micro-optimizations • Remove unused optional arg • Fix bug in rendering image data • Added more examples • Add setup file • Support SH1106 • Documentation updates	2016/09/06

CHAPTER 11

The MIT License (MIT)

Python Module Index

l

Index

L

P

S