luma.oled Documentation

Release 3.13.0

Richard Hull and contributors

Aug 13, 2023

Contents

CHAPTER 1

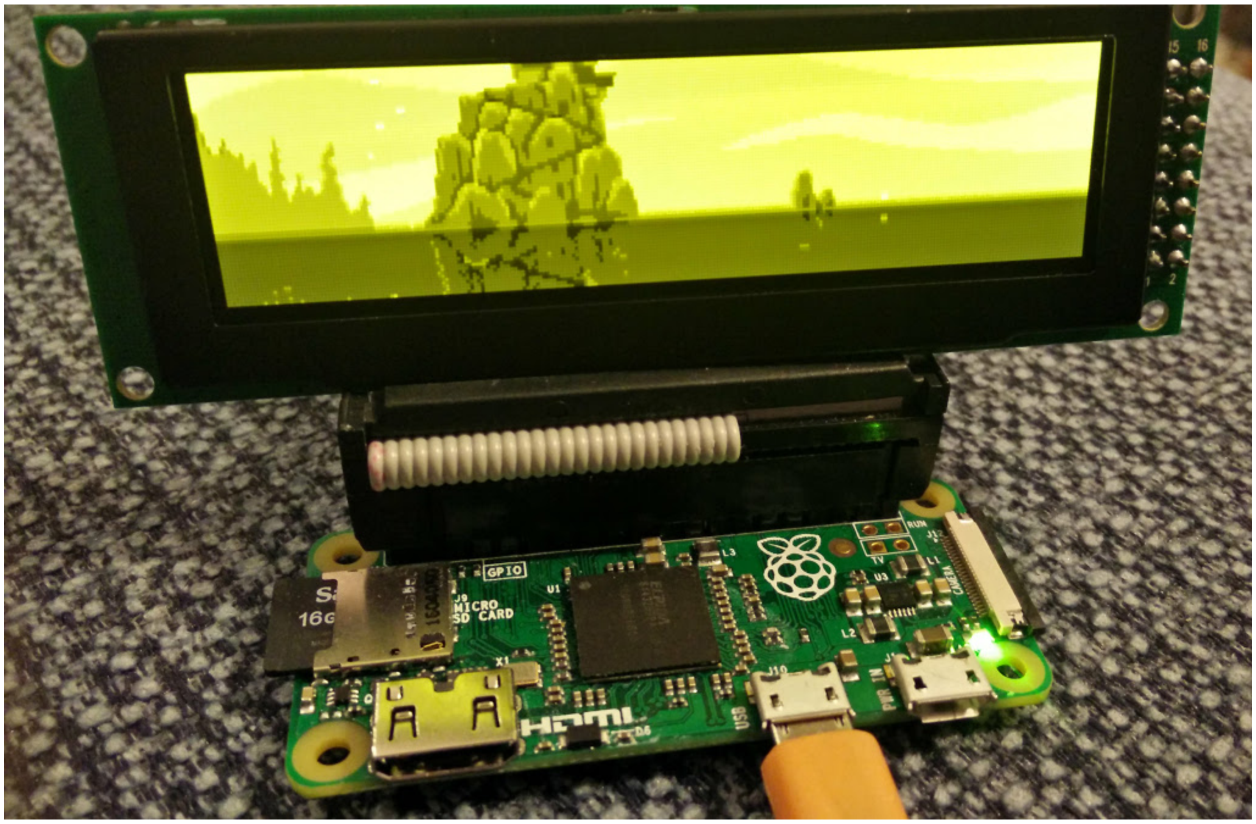
Introduction

luma.oled provides a Python3 interface to OLED matrix displays with the SSD1306, SSD1309, SSD1322, SSD1325, SSD1327, SSD1331, SSD1351, SSD1362, SH1106, SH1107 or WS0010 controllers to connect with Raspberry Pi and other Linux-based single-board computers (SBC). It 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



See also:

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

- `SSD1306`
- `SSD1309`
- `SSD1322`
- `SSD1325`
- `SSD1327`
- `SSD1331`
- `SSD1351`
- `SSD1362`
- `SH1106`
- `WS0010`
- `WEH001602`

Benchmarks for tested devices can be found in the wiki.

1.1 Examples and Emulators

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

Installation

The successful installation of a display module to your SBC requires a combination of tasks to be completed before the display will operate correctly.

First, the device needs to be wired up correctly to your single-board computer (SBC) and the interface that will be used needs to be enabled in the kernel of the operating system of the SBC. Instructions to for this are provided in *Hardware*.

Equally important, the `luma.oled` software needs to be installed including the build dependencies that for the python modules it uses. Instructions to complete that task are provided in *Software*.

Finally, you need to leverage the appropriate interface class and display class for your device to implement your application. Instructions for that are included in *Python usage*.

Note: This library has been tested against Python 3.5, 3.6, 3.7 and 3.8.

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 models A, B2, 3B, Zero, Zero W, OrangePi Zero (Armbian Jessie), and 4B with Raspberry Pi OS Jessie, Stretch, Buster and Bullseye operating systems.

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

CHAPTER 3

Hardware

It is essential that you get your device correctly wired to your single-board computer (SBC). The needed connections vary based upon the type of interface your device supports. There three major styles of interfaces that are popular with small LCD displays. These include I2C, SPI and 6800 style parallel-bus interfaces.

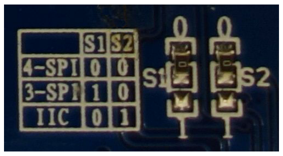
3.1 I2C vs. SPI vs Parallel

If you have not yet purchased your display, you may be wondering if you should get an I2C, SPI or parallel-bus display. The basic trade-off is implementation complexity and speed. The I2C is the easiest 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. Parallel-bus displays are both slower and harder to connect but are typically less expensive.

3.2 Identifying your interface

You can determine if you have an I2C, SPI or parallel-bus 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, and a parallel-bus interface will typically need to have at least 9 pins connected on a device but can requires 16 or more depending on the size of the bus and what other signals are required.

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.

Tip:

- 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**.

3.3 I2C

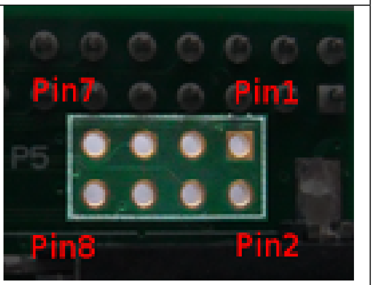
I2C interfaces are the simplest to wire as they contain the smallest number of pins. The only signal lines are serial data (SDA) and serial clock (SCL). That plus the power and ground connections complete the required connections.

If you are using a Raspberry Pi you will normally attach to header P1. The P1 header pins should be connected as follows:

Device 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)

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.

Device 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)	

3.3.1 Enabling The I2C Interface

The I2C interface may not be enabled by default on your SBC. To check if it 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.

For Raspberry Pi OS, enable the I2C driver 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

3.3.2 Determining Address

Check that the device is communicating properly (if using a rev.1 board, use 0 for the bus, not 1) and determine its address using `i2cdetect`:

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

The address for your device will be needed when you initialize the interface. In the example above, the display address is 0x3c. Keep in mind that I2C buses can have more than one device attached. If more than one address is shown when you run i2cdetect, you will need to determine which one is associated with your display. Typically displays will only support a limited set of possible addresses (often just one). Checking the documentation can help determine which device is which.

3.4 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 4 B.

Warning: There appears to be varying pin-out configurations on different display modules! Make sure to verify the pin numbers of your device by their function especially VCC and GND.

Device 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:

- If you're already using the listed GPIO pins for Data/Command and/or Reset, you can select other pins and pass `gpio_DC` and/or `gpio_RST` argument specifying the new *GPIO* pin numbers in your serial interface create call (this applies to PCD8544, ST7567 and ST7735).
- Because CE 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.
- 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 `gpio_DC` and/or a `gpio_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.

3.4.1 Enabling The SPI Interface

To enable the SPI port on a Raspberry Pi running Raspbian:

```
$ 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.

3.5 Parallel

Beyond the power and ground connections, you can choose which ever GPIO pins you like to connect up your display. The following is one example for how to wire popular displays such as the Winstar WEH001602A.

Device Pin	Name	Remarks	RPi Pin	RPi Function
1	GND	Ground	P01-6	GND
2	VDD	+5.0V Power	P01-2	5V Power
3	NC	Not Connect		
4	RS	Register Select	P01-26	GPIO 7
5	R/W	Read/Write	P01-14	GND
6	E	Enable	P01-24	GPIO 8
7	D0	Not Connected		
8	D1	Not Connected		
9	D2	Not Connected		
10	D3	Not Connected		
11	D4	Databus line 4	P01-22	GPIO 25
12	D5	Databus line 5	P01-18	GPIO 24
13	D6	Databus line 6	P01-16	GPIO 23
14	D7	Databus line 7	P01-13	GPIO 27
15	NC	Not Connect		
16	NC	Not Connect		

Note:

- You have the choice on whether to wire your device with 4 or 8 data-lines. Wiring with 8 provides a faster interface but at the cost of increased wiring complexity. Most implementations use 4 data-lines which provides acceptable performance and is the default setting for the parallel class.
- Reading from the display is not supported by the `luma.core.interface.parallel.bitbang_6800` class so it needs to be connected to ground in order to always be set for writes (assuming the device uses logic-low for write).

Warning:

- Be careful with the logic level of the device you are using. Many SBCs including the Raspberry Pi uses 3.3V logic. If your device supplies 5V to one of the GPIO pins of an SBC that uses 3.3V logic, you may damage your SBC.

CHAPTER 4

Software

Install the latest version of the library directly from PyPI with:

```
$ sudo -H pip3 install --upgrade luma.oled
```

This will normally retrieve all of the dependencies `luma.oled` requires and install them automatically.

4.1 Installing Dependencies

If `pip` is unable to automatically install its dependencies you will have to add them manually. To resolve the issues you will need to add the appropriate development packages to continue.

If you are using Raspberry Pi OS you should be able to use the following commands to add the required packages:

```
$ sudo apt-get update
$ sudo apt-get install python3 python3-pip python3-pil libjpeg-dev zlib1g-dev␣
↪libfreetype6-dev liblcms2-dev libopenjp2-7 libtiff5 -y
$ sudo -H pip3 install luma.oled
```

If you are not using Raspberry Pi OS you will need to consult the documentation for your Linux distribution to determine the correct procedure to install the dependencies.

Tip: If your distribution includes a pre-packaged version of Pillow, use it instead of installing from pip. On many distributions the correct package to install is `python3-imaging`. Another common package name for Pillow is `python3-pil`.:

```
$ sudo apt-get install python3-imaging
```

or:

```
$ sudo apt-get install python3-pil
```

4.2 Permissions

`luma.oled` uses hardware interfaces that require permission to access. After you have successfully installed `luma.oled` you may want to add the user account that your `luma.oled` program will run as to the groups that grant access to these interfaces.:

```
$ sudo usermod -a -G spi,gpio,i2c pi
```

Replace `pi` with the name of the account you will be using.

CHAPTER 5

Python usage

OLED displays can be driven with python using the various 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.

To begin you must import the device class you will be using and the interface class that you will use to communicate with your device:

In this example, we are using an I2C interface with a ssd1306 display.

```
from luma.core.interface.serial import i2c, spi, pcf8574
from luma.core.interface.parallel import bitbang_6800
from luma.core.render import canvas
from luma.oled.device import ssd1306, ssd1309, ssd1325, ssd1331, sh1106, sh1107,␣
↪ws0010

# rev.1 users set port=0
# substitute spi(device=0, port=0) below if using that interface
# substitute bitbang_6800(RS=7, E=8, PINS=[25,24,23,27]) 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 when 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.

Note: When a program ends, the display is automatically cleared. This means that a fast program that ends quickly may never display a visible image.

5.1 Color Model

Any of the standard `PIL.ImageColor` color formats may be used, but since the SSD1306, SH1106, SH1107 and WS0010 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, SSD1325 and SSD1362 OLEDs all 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'
```

5.1.1 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
from time import sleep

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")
sleep(10)
```

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.

5.1.2 Examples

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

CHAPTER 6

Upgrade

6.1 3.6.0

Version 3.6.0 was released on 24 September 2020: This was the last version that supported Python 3.5.

6.2 3.4.0

Version 3.4.0 was released on 19 January 2020: This was the last version that supported Python 2.7.

6.3 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:

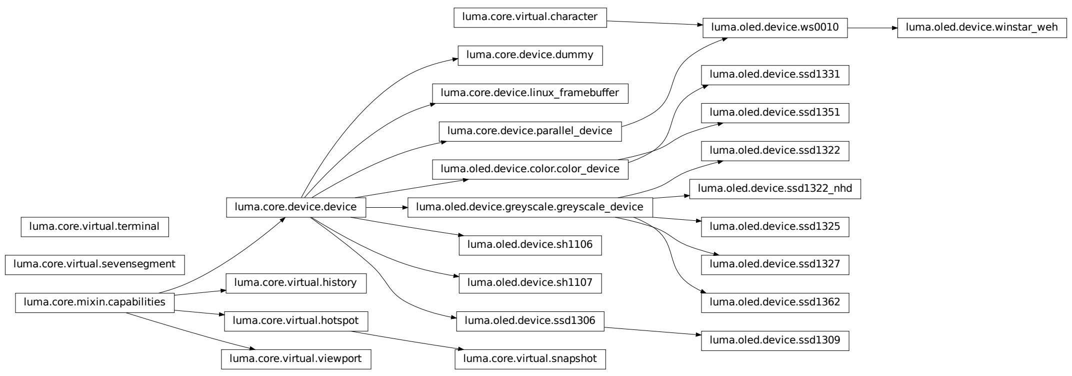
- rename instances of `oled.device` to `luma.oled.device`.
- rename any other usages of `oled.*` to `luma.core.*`.

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

CHAPTER 7

API Documentation

OLED display driver for SSD1306, SSD1309, SSD1322, SSD1325, SSD1327, SSD1331, SSD1351, SSD1362, SH1106 and SH1107 devices.


luma.core.virtual.character
luma.oled.device.ws0010
luma.oled.device.winstar_weh
luma.core.device.dummy
luma.oled.device.ssd1331
luma.core.device.linux_framebuffer
luma.oled.device.ssd1351
luma.core.device.parallel_device
luma.oled.device.ssd1322
luma.oled.device.color.color_device
luma.oled.device.ssd1322_nhd
luma.core.device.device
luma.oled.device.greyscale.greyscale_device
luma.core.virtual.terminal
luma.oled.device.ssd1325
luma.oled.device.sh1106
luma.core.virtual.sevensegment
luma.oled.device.ssd1327
luma.core.virtual.history
luma.oled.device.sh1107
luma.core.mixin.capabilities
luma.oled.device.ssd1362
luma.core.virtual.hotspot
luma.oled.device.ssd1306
luma.oled.device.ssd1309
luma.core.virtual.viewport
luma.core.virtual.snapshot

7.1 `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` method.

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` method.

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=None, **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 instances 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.

`init_framebuffer`(*framebuffer*)

`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` method.

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.`**`ssd1362`**(*serial_interface=None*, *width=256*, *height=64*, *rotate=0*, *mode='RGB'*, *framebuffer=None*, ***kwargs*)

Bases: `luma.oled.device.greyscale.greyscale_device`

Serial interface to a 4-bit greyscale SSD1362 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 instances of `diff_to_previous` or `full_frame` are only supported

New in version 3.4.0.

`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.

init_framebuffer(*framebuffer*)

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` method.

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_nhd**(*serial_interface=None*, *width=128*, *height=64*, *rotate=0*, *mode='RGB'*, *framebuffer=<luma.core.framebuffer.full_frame object>*, ***kwargs*)
Bases: `luma.oled.device.greyscale.greyscale_device`

Similar to ssd1322 but several options are hard coded: width, height and frame buffer

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 8-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.

`init_framebuffer`(*framebuffer*)

`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` method.

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=None*, ***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.

init_framebuffer(*framebuffer*)

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` method.

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=None*, ***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.

init_framebuffer(*framebuffer*)

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` method.

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=None*, ***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 instances 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.

init_framebuffer(*framebuffer*)

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` method.

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=None*, *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 instances 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.

`init_framebuffer`(*framebuffer*)

`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` method.

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` method.

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.**sh1107**(*serial_interface=None*, *width=64*, *height=128*, *rotate=0*, ***kwargs*)

Bases: `luma.core.device.device`

Serial interface to a monochrome SH1107 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 64).
- **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.

New in version 3.11.0.

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 SH1107 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` method.

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.**ws0010**(*serial_interface=None*, *width=100*, *height=16*, *undefined='_'*, *font=None*, *selected_font=0*, *exec_time=4.9999999999999996e-05*, *rotate=0*, *framebuffer=None*, *const=<class 'luma.oled.const.ws0010'>*, ***kwargs*)

Bases: `luma.core.device.parallel_device`, `luma.core.virtual.character`, `luma.oled.device.framebuffer_mixin.__framebuffer_mixin`

Serial interface to a monochrome Winstar WS0010 OLED display. This interface will work with most ws0010 powered devices including the weg010016.

Parameters

- **serial_interface** – The serial interface (usually a `luma.core.interface.serial.parallel` instance) to delegate sending data and commands through.
- **width** (*int*) – The number of pixels laid out horizontally.
- **height** (*int*) – The number of pixels laid out vertically.
- **undefined** (*str*) – The character to display if the font doesn't contain the requested character
- **font** (`PIL.ImageFont`) – Allows you to override the internal font by passing in an alternate
- **selected_font** (*int or str*) – Select one of the ws0010's embedded font tables (see note). Can be selected by number or name. Default is 'FT00' (English Japanese).
- **exec_time** (*float*) – Time in seconds to wait for a command to complete. Default is 50 μs (1e-6 * 50) which is enough for all ws0010 commands.
- **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.

To place text on the display, simply assign the text to the 'text' instance variable:

```
p = parallel(RS=7, E=8, PINS=[25,24,23,18])
my_display = ws0010(p, selected_font='FT01')
my_display.text = 'WS0010 Display\nFont FT01 5x8'
```

For more details on how to use the 'text' interface see `luma.core.virtual.character`

New in version 3.6.0.

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*, *exec_time=None*, *only_low_bits=False*)
: Sends a command or sequence of commands through to the serial interface. If operating in four bit mode, expands each command from one byte values (8 bits) to two nibble values (4 bits each)

  Parameters

  - **cmd** (*int*) – A spread of commands.
  - **exec_time** (*float*) – Amount of time to wait for the command to finish execution. If not provided, the device default will be used instead
  - **only_low_bits** (*bool*) – If `True`, only the lowest four bits of the command will be sent. This is necessary on some devices during initialization

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 sequence of bytes through to the serial interface. If operating in four bit mode, expands each byte from a single value (8 bits) to two nibble values (4 bits each)

  Parameters **data** (*list*) – a sequence of bytes to send to the display

display(*image*)
: Takes a 1-bit `PIL.Image` and dumps it to the ws0010 OLED display.

get_font(*ft*)
: Load one of the devices embedded fonts by its index value or name and return it

  Parameters **val** (*int or str*) – The index or the name of the font to return

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

init_framebuffer(*framebuffer*)

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` method.

  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.

text
Returns the current state of the text buffer. This may not reflect accurately what is displayed on the device if the font does not have a symbol for a requested text value.

class luma.oled.device.**winstar_weh**(*serial_interface=None*, *width=16*, *height=2*, ***kwargs*)
Bases: `luma.oled.device.ws0010`

Serial interface to a monochrome Winstar WEH OLED display. This is the character version of the display using the ws0010 controller. This class provides the same `text` property as the ws0010 interface so you can set the text value which will be rendered to the display's screen. This interface uses a variant of the ws0010 controller's built-in font that is designed to match the grid structure of the weh displays (see note below).

Parameters

- **serial_interface** – The serial interface (usually a `luma.core.interface.serial.parallel` instance) to delegate sending data and commands through.
- **width** (*int*) – The number of characters that can be displayed on a single line. Example: the weh001602a has a width of 16 characters.
- **height** (*int*) – The number of lines the display has. Example: the weh001602a has a height of 2 lines.
- **undefined** (*str*) – The character to display if the font doesn't contain the requested character
- **font** (`PIL.ImageFont`) – Allows you to override the internal font by passing in an alternate
- **default_table** (*int*) – Select one of the ws0010's four embedded font tables (see `ws0010` documentation)
- **embedded_font** (*str '5x8' or '5x10'*) – Select the size of the embedded font to use. Allowed sizes are 5x8 (default) and 5x10
- **exec_time** (*float*) – Time in seconds to wait for a command to complete. Default is 50 μs (1e-6 * 50)
- **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.

New in version 3.6.0.

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*, *exec_time=None*, *only_low_bits=False*)
Sends a command or sequence of commands through to the serial interface. If operating in four bit mode, expands each command from one byte values (8 bits) to two nibble values (4 bits each)

Parameters

- **cmd** (*int*) – A spread of commands.
- **exec_time** (*float*) – Amount of time to wait for the command to finish execution. If not provided, the device default will be used instead
- **only_low_bits** (*bool*) – If `True`, only the lowest four bits of the command will be sent. This is necessary on some devices during initialization

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 sequence of bytes through to the serial interface. If operating in four bit mode, expands each byte from a single value (8 bits) to two nibble values (4 bits each)

Parameters **data** (*list*) – a sequence of bytes to send to the display

display(*image*)
Takes a 1-bit `PIL.Image` and dumps it to the ws0010 OLED display.

get_font(*ft*)
Load one of the devices embedded fonts by its index value or name and return it

Parameters **val** (*int or str*) – The index or the name of the font to return

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

init_framebuffer(*framebuffer*)

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` method.

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.

text
Returns the current state of the text buffer. This may not reflect accurately what is displayed on the device if the font does not have a symbol for a requested text value.

CHAPTER 8

Troubleshooting

8.1 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).

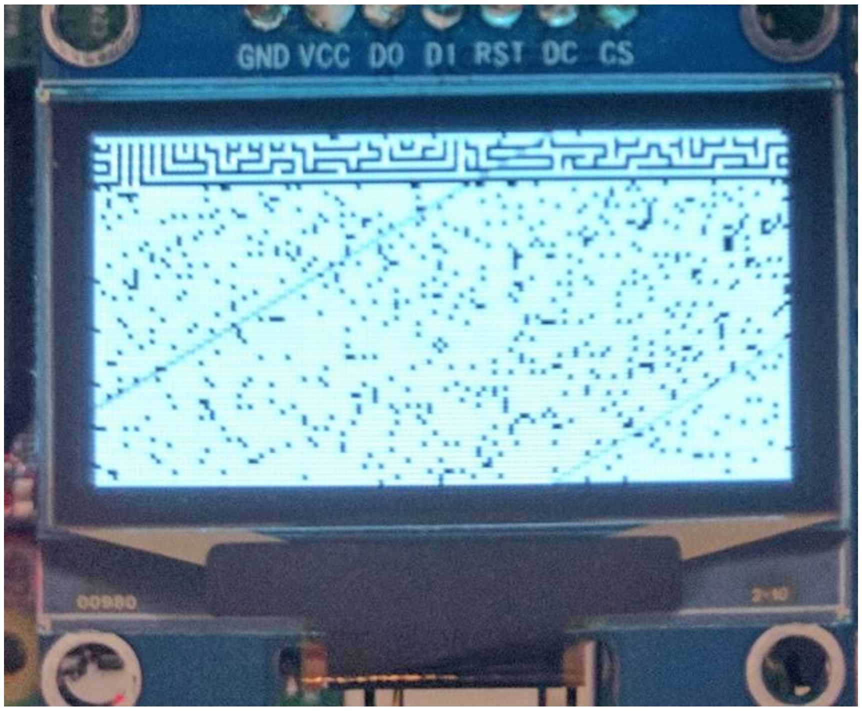


Fig. 1: Display corruption due to using driver for incorrect controller when running the *maze.py* example

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 9

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/
- https://www.circuits.dk/everything-about-raspberry-gpio/

CHAPTER 10

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.

10.1 GitHub

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

10.2 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)

- @wjgeorge
- Doug Burrell (@doug-burrell)
- Xavier Carcelle (@xcarcelle)
- Dhrone (@dhrone)
- George Harker (@georgeharker)
- @Pako2
- Graeme Smecher (@gsmecher)
- Romit Raj (@QuicksandDesignStudio)

CHAPTER 11

ChangeLog

Version	Description	Date
3.13.0	• Add support for 128x128 SH1107	2023/08/12
3.12.0	• Fix 96x16 OLED panel offset	2023/03/19
3.11.0	• Add support for SH1107 greyscale OLED	2023/01/26
3.10.0	• Fix SSD1322 NHD initialization and encode each pixel as 4bit+4bit identical nibbles	2022/11/14
3.9.0	• Use native namespace package configuration • Drop support for Python 3.6	2022/10/19
3.8.1	• Fix mutable default parameter bug when using multiple displays	2020/11/15

Continued on next page

Table 1 – continued from previous page

Version	Description	Date
3.8.0	• Improved diff_to_previous framebuffer performance	2020/11/06
3.7.0	• Drop support for Python 3.5, only 3.6 or newer is supported now • Add support for SSD1351 128x96 display • Pin luma.core to 1.x.y line only, in anticipation of performance improvements in upcoming major release	2020/10/25
3.6.0	• Add support for Winstar OLED displays	2020/09/24
3.5.0	• Drop support for Python 2.7, only 3.5 or newer is supported now	2020/07/04
3.4.0	• Add support for SSD1362 256x64 Greyscale OLED	2020/01/19
3.3.0	• Namespace fix	2019/06/19
3.2.1	• Fix bug where SSD1325 `framebuffer=diff_to_prev` didn't set column address properly, resulting in garbled output	2019/04/30
3.2.0	• Add support for 128x64 OLED (Newhaven SSD1322_NHD))	2019/04/17

Continued on next page

Table 1 – continued from previous page

Version	Description	Date
3.1.1	• Fix bug where SSD1327 `framebuffer=diff_to_prev` didn't set column address properly, resulting in garbled output • Minor API documentation improvements	2019/03/30
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
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

Continued on next page

Table 1 – continued from previous page

Version	Description	Date
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
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

Continued on next page

Table 1 – continued from previous page

Version	Description	Date
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
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

Continued on next page

Table 1 – continued from previous page

Version	Description	Date
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
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

Continued on next page

Table 1 – continued from previous page

Version	Description	Date
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 on emulated devices • Additional tests	2016/12/03

Continued on next page

Table 1 – continued from previous page

Version	Description	Date
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
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

Continued on next page

Table 1 – continued from previous page

Version	Description	Date
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 12

The MIT License (MIT)

Python Module Index

Index