ssd1306 Documentation

Release 1.5.0

Richard Hull

Mar 14, 2017

Contents

1 Introduction **1**

2 Python usage **3**
2.1 Color Model 4
2.2 Landscape / Portrait Orientation 4
2.3 Examples 4
2.4 Emulators 6

3 Hardware **7**
3.1 Identifying your serial interface 7
3.2 I2C vs. SPI 7
3.3 Tips for connecting the display 7
3.4 Pre-requisites 8

4 Installation **11**
4.1 From PyPI 11
4.2 From source 12

5 API Documentation **13**
5.1 `oled.device` 13
5.2 `oled.emulator` 15
5.3 `oled.error` 16
5.4 `oled.mixin` 17
5.5 `oled.render` 17
5.6 `oled.serial` 17
5.7 `oled.threadpool` 18
5.8 `oled.virtual` 19

6 References **23**

7 Contributing **25**
7.1 GitHub 25
7.2 Contributors 25

8 ChangeLog **27**

9 The MIT License (MIT) **29**

Python Module Index 31

CHAPTER 1

Introduction

Interfacing OLED matrix displays with the SSD1306, SSD1325, SSD1331 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.

See also:

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

- `SSD1306`
- `SSD1325`
- `SSD1331`
- `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 examples directory:

CHAPTER 2

Python usage

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

First, import and initialise the device:

```
from oled.serial import i2c
from oled.device import ssd1306, ssd1331, sh1106
from oled.render import canvas

# 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 specific `ssd1306`, `ssd1331` or `sh1106` 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 *`oled.render.canvas`* class automatically creates an `PIL.ImageDraw` object of the correct dimensions and bit depth suitable for the device, so you may then call the usual Pillow methods to draw onto the canvas.

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

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 `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 OLED which features 16-bit RGB colors: 24-bit RGB images are downsized to 16-bit using a 565 scheme.

The SSD1325 OLED supports 16 greyscale graduations: 24-bit RGB images are downsized to 4-bit using a Luma conversion which is approximately calculated as follows:

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

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 oled.serial import i2c
from oled.device import ssd1306, ssd1331, sh1106
from oled.render import canvas

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.

Examples

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

Example	Description
3d_box.py	Rotating 3D box wireframe & color dithering
bounce.py	Display a bouncing ball animation and frames per second
carousel.py	Showcase viewport and hotspot functionality
clock.py	An analog clockface with date & time
colors.py	Color rendering demo
crawl.py	A vertical scrolling demo, which should be familiar
demo.py	Use misc draw commands to create a simple image
game_of_life.py	Conway's game of life
grayscale.py	Greyscale rendering demo
invaders.py	Space Invaders demo
maze.py	Maze generator
perfloop.py	Simple benchmarking utility to measure performance
pi_logo.py	Display the Raspberry Pi logo (loads image as .png)
savepoint.py	Example of savepoint/restore functionality
starfield.py	3D starfield simulation
sys_info.py	Display basic system information
terminal.py	Simple println capabilities
tv_snow.py	Example image-blitting
welcome.py	Unicode font rendering & scrolling

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

```
$ python pi_logo.py -h
usage: pi_logo.py [-h] [--config CONFIG]
                  [--display {ssd1306,ssd1331,sh1106,capture,pygame,gifanim}]
                  [--width WIDTH] [--height HEIGHT] [--rotate {0,1,2,3}]
                  [--interface {i2c,spi}] [--i2c-port I2C_PORT]
                  [--i2c-address I2C_ADDRESS] [--spi-port SPI_PORT]
                  [--spi-device SPI_DEVICE] [--spi-bus-speed SPI_BUS_SPEED]
                  [--bcm-data-command BCM_DATA_COMMAND]
                  [--bcm-reset BCM_RESET]
                  [--transform {none,identity,scale2x,smoothscale}]
                  [--scale SCALE] [--mode {1,RGB,RGBA}] [--duration DURATION]
                  [--loop LOOP] [--max-frames MAX_FRAMES]

oled arguments

optional arguments:
  -h, --help            show this help message and exit
  --config CONFIG, -f CONFIG
                        Load configuration settings from a file (default:
                        None)
  --display {ssd1306,ssd1331,sh1106,capture,pygame,gifanim}, -d {ssd1306,ssd1331,
↪sh1106,capture,pygame,gifanim}
                        Display type, supports real devices or emulators
                        (default: ssd1306)
  --width WIDTH         Width of the device in pixels (default: 128)
  --height HEIGHT       Height of the device in pixels (default: 64)
  --rotate {0,1,2,3}, -r {0,1,2,3}
                        Rotation factor (default: 0)
  --interface {i2c,spi}, -i {i2c,spi}
                        Serial interface type (default: i2c)
  --i2c-port I2C_PORT   I2C bus number (default: 1)
  --i2c-address I2C_ADDRESS
```

```
                        I2C display address (default: 0x3C)
  --spi-port SPI_PORT   SPI port number (default: 0)
  --spi-device SPI_DEVICE
                        SPI device (default: 0)
  --spi-bus-speed SPI_BUS_SPEED
                        SPI max bus speed (Hz) (default: 8000000)
  --bcm-data-command BCM_DATA_COMMAND
                        BCM pin for D/C RESET (SPI devices only) (default: 24)
  --bcm-reset BCM_RESET
                        BCM pin for RESET (SPI devices only) (default: 25)
  --transform {none,identity,scale2x,smoothscale}
                        Scaling transform to apply (emulator only) (default:
                        scale2x)
  --scale SCALE         Scaling factor to apply (emulator only) (default: 2)
  --mode {1,RGB,RGBA}   Colour mode (emulator only) (default: RGB)
  --duration DURATION   Animation frame duration (gifanim emulator only)
                        (default: 0.01)
  --loop LOOP           Repeat loop, zero=forever (gifanim emulator only)
                        (default: 0)
  --max-frames MAX_FRAMES
                        Maximum frames to record (gifanim emulator only)
                        (default: None)
```

Note:

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

Emulators

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

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

Invoke the demos with:

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

or:

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

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

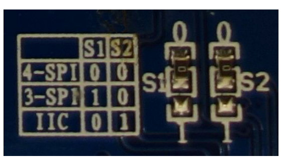
CHAPTER 3

Hardware

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.

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).

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.

Pre-requisites

I2C

The P1 header pins should be connected as follows:

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

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

See also:

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

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

Ensure that the I2C kernel driver is enabled:

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

or:

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

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

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

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

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

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

Then reboot.

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

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

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

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

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

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

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

CHAPTER 4

Installation

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

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

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

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

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

From PyPI

Note: This is the preferred installation mechanism.

Install the latest version of the library directly from PyPI:

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

From source

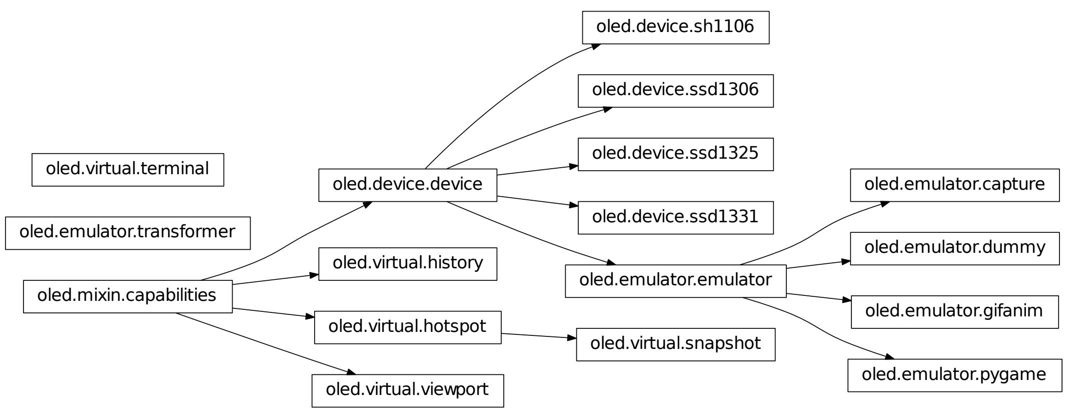
For Python 2, from the bash prompt, enter:

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

CHAPTER 5

API Documentation

OLED display driver for SSD1306, SSD1325, SSD1331 and SH1106 devices.



oled.device

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

Bases: *`oled.mixin.capabilities`*

Base class for OLED driver classes

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

cleanup()

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.

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

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

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

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

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

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

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

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

class oled.device.**ssd1325**(*serial_interface=None*, *width=128*, *height=64*, *rotate=0*)
Bases: `oled.device.device`

Encapsulates the serial interface to the 4-bit greyscale SSD1325 OLED display hardware. On creation, an initialization sequence is pumped to the display to properly configure it. Further control commands can then be called to affect the brightness and other settings.

display(*image*)
Takes a 24-bit RGB `PIL.Image` and dumps it to the SSD1325 OLED display, converting the image pixels to 4-bit greyscale using a simplified Luma calculation, based on *Y'=0.299R'+0.587G'+0.114B'*.

class oled.device.**ssd1331**(*serial_interface=None*, *width=96*, *height=64*, *rotate=0*)
Bases: `oled.device.device`

Encapsulates the serial interface to the 16-bit color (5-6-5 RGB) SSD1331 OLED display hardware. On creation, an initialization sequence is pumped to the display to properly configure it. Further control commands can then be called to affect the brightness and other settings.

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.

display (*image*)
Takes a 24-bit RGB `PIL.Image` and dumps it to the SSD1331 OLED display.

oled.emulator

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

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

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

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

class oled.emulator.**dummy** (*width=128*, *height=64*, *rotate=0*, *mode='RGB'*, *transform='scale2x'*, *scale=2*, ***kwargs*)
Bases: *oled.emulator.emulator*

Pseudo-device that acts like an OLED display, except that it does nothing other than retain a copy of the displayed image. It is mostly useful for testing. While the capability of an OLED device is monochrome, there is no limitation here, and hence supports 24-bit color depth.

display (*image*)
Takes a `PIL.Image` and makes a copy of it for later use/inspection.

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

Base class for emulated OLED driver classes

cleanup ()

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

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

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

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

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

write_animation()

class oled.emulator.**pygame**(*width=128*, *height=64*, *rotate=0*, *mode='RGB'*, *transform='scale2x'*, *scale=2*, *frame_rate=60*, ***kwargs*)
Bases: *oled.emulator.emulator*

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

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

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

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

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

Helper class used to dispatch transformation operations.

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

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

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

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

oled.error

Exceptions for this library.

exception oled.error.**DeviceAddressError**
Bases: *oled.error.Error*

Exception raised when an invalid device address is detected.

exception oled.error.**DeviceDisplayModeError**
Bases: *oled.error.Error*

Exception raised when an invalid device display mode is detected.

exception oled.error.**DeviceNotFoundError**
Bases: *oled.error.Error*

Exception raised when a device cannot be found.

exception oled.error.**DevicePermissionError**
Bases: *oled.error.Error*

Exception raised when permission to access the device is denied.

exception `oled.error.`**`Error`**
Bases: `exceptions.Exception`

Base class for exceptions in this library.

`oled.mixin`

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

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

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

`display`(*image*)

`preprocess`(*image*)

`oled.render`

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

By default, any color (other than black) will be treated as white and displayed on the device. However, this behaviour can be changed by adding `dither=True` and the image will be converted from RGB space into a 1-bit monochrome image where dithering is employed to differentiate colors at the expense of resolution.

`oled.serial`

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

Wrap an I2C interface to provide data and command methods.

Parameters

- **`bus`** – I2C bus instance.
- **`port`** (*`int`*) – I2C port number.
- **`address`** – I2C address.

Raises

- ***`oled.error.DeviceAddressError`*** – I2C device address is invalid.
- ***`oled.error.DeviceNotFoundError`*** – I2C device could not be found.
- ***`oled.error.DevicePermissionError`*** – Permission to access I2C device denied.

Note:

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

---

cleanup()
Clean up I2C resources

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

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

class oled.serial.**noop**
Bases: object

Does nothing, used for pseudo-devices / emulators, which dont have a serial interface.

cleanup()

command(**cmd*)

data(*data*)

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

Wraps an SPI interface to provide data and command methods.

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

Raises *oled.error.DeviceNotFoundError* – SPI device could not be found.

cleanup()
Clean up SPI & GPIO resources

command(**cmd*)
Sends a command or sequence of commands through to the SPI device.

data(*data*)
Sends a data byte or sequence of data bytes through to the SPI device. If the data is more than 4Kb in size, it is sent in chunks.

oled.threadpool

class oled.threadpool.**threadpool**(*num_threads*)
Pool of threads consuming tasks from a queue

add_task(*func*, **args*, ***kargs*)
Add a task to the queue

wait_completion()
Wait for completion of all the tasks in the queue

class oled.threadpool.**worker**(*tasks*)
Bases: threading.Thread

Thread executing tasks from a given tasks queue

run()

oled.virtual

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

class oled.virtual.**history**(*device*)
Bases: *oled.mixin.capabilities*

Wraps a device (or emulator) to provide a facility to be able to make a savepoint (a point at which the screen display can be "rolled-back" to).

This is mostly useful for displaying transient error/dialog messages which could be subsequently dismissed, reverting back to the previous display.

display(*image*)

restore(*drop=0*)
Restores the last savepoint. If `drop` is supplied and greater than zero, then that many savepoints are dropped, and the next savepoint is restored.

savepoint()
Copies the last displayed image.

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

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

You would either:

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

paste_into(*image*, *xy*)

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

update(*draw*)

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

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

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

paste_into(*image*, *xy*)

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

class oled.virtual.**terminal**(*device*, *font=None*, *color='white'*, *bgcolor='black'*, *tabstop=4*, *line_height=None*, *animate=True*)
Bases: `object`

Provides a terminal-like interface to a device (or a device-like object that has `mixin.capabilities` characteristics).

backspace()
Moves the cursor one place to the left, erasing the character at the current position. Cannot move beyound column zero, nor onto the previous line

carriage_return()
Returns the cursor position to the left-hand side without advancing downwards.

clear()
Clears the display and resets the cursor position to (0, 0).

erase()
Erase the contents of the cursor's current postion without moving the cursor's position.

flush()
Cause the current backing store to be rendered on the nominated device.

newline()
Advances the cursor position ot the left hand side, and to the next line. If the cursor is on the lowest line, the displayed contents are scrolled, causing the top line to be lost.

println(*text=''*)
Prints the supplied text to the device, scrolling where necessary. The text is always followed by a newline.

putch(*ch*, *flush=True*)
Prints the specific character, which must be a valid printable ASCII value in the range 32..127 only.

puts(*text*)
Prints the supplied text, handling special character codes for carriage return (r), newline (n), backspace (b) and tab (t).

If the `animate` flag was set to True (default), then each character is flushed to the device, giving the effect of 1970's teletype device.

tab()
Advances the cursor position to the next (soft) tabstop.

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

add_hotspot(*hotspot*, *xy*)
Add the hotspot at (x, y). The hotspot must fit inside the bounds of the virtual device. If it does not then an AssertError is raised.

display(*image*)

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

refresh()

remove_hotspot(*hotspot*, *xy*)
Remove the hotspot at (x, y): Any previously rendered image where the hotspot was placed is erased from

the backing image, and will be "undrawn" the next time the virtual device is refreshed. If the specified hotspot is not found (x, y), a ValueError is raised.

set_position(*xy*)

CHAPTER 6

References

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

CHAPTER 7

Contributing

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

GitHub

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

Contributors

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

CHAPTER 8

ChangeLog

Version	Description	Date
Upcoming	*TBD*	
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	2016/12/11

CHAPTER 9

The MIT License (MIT)

Python Module Index

o

oled, 13
oled.device, 13
oled.emulator, 15
oled.error, 16
oled.mixin, 17
oled.render, 17
oled.serial, 17
oled.threadpool, 18
oled.virtual, 19

Index

A

add_hotspot() (oled.virtual.viewport method), 20
add_task() (oled.threadpool.threadpool method), 18

B

backspace() (oled.virtual.terminal method), 20

C

calc_bounds() (in module oled.virtual), 19
canvas (class in oled.render), 17
capabilities (class in oled.mixin), 17
capabilities() (oled.mixin.capabilities method), 17
capture (class in oled.emulator), 15
carriage_return() (oled.virtual.terminal method), 20
cleanup() (oled.device.device method), 14
cleanup() (oled.emulator.emulator method), 15
cleanup() (oled.serial.i2c method), 18
cleanup() (oled.serial.noop method), 18
cleanup() (oled.serial.spi method), 18
clear() (oled.mixin.capabilities method), 17
clear() (oled.virtual.terminal method), 20
command() (oled.device.device method), 14
command() (oled.serial.i2c method), 18
command() (oled.serial.noop method), 18
command() (oled.serial.spi method), 18
contrast() (oled.device.device method), 14
contrast() (oled.device.ssd1331 method), 15

D

data() (oled.device.device method), 14
data() (oled.serial.i2c method), 18
data() (oled.serial.noop method), 18
data() (oled.serial.spi method), 18
device (class in oled.device), 13
DeviceAddressError, 16
DeviceDisplayModeError, 16
DeviceNotFoundError, 16
DevicePermissionError, 16
display() (oled.device.sh1106 method), 14
display() (oled.device.ssd1306 method), 14
display() (oled.device.ssd1325 method), 14
display() (oled.device.ssd1331 method), 15
display() (oled.emulator.capture method), 15
display() (oled.emulator.dummy method), 15
display() (oled.emulator.gifanim method), 15
display() (oled.emulator.pygame method), 16
display() (oled.mixin.capabilities method), 17
display() (oled.virtual.history method), 19
display() (oled.virtual.viewport method), 20
dummy (class in oled.emulator), 15

E

emulator (class in oled.emulator), 15
erase() (oled.virtual.terminal method), 20
Error, 16

F

flush() (oled.virtual.terminal method), 20

G

gifanim (class in oled.emulator), 15

H

hide() (oled.device.device method), 14
history (class in oled.virtual), 19
hotspot (class in oled.virtual), 19

I

i2c (class in oled.serial), 17
identity() (oled.emulator.transformer method), 16
is_overlapping_viewport() (oled.virtual.viewport method), 20

N

newline() (oled.virtual.terminal method), 20
none() (oled.emulator.transformer method), 16
noop (class in oled.serial), 18

O

oled (module), 13
oled.device (module), 13
oled.emulator (module), 15
oled.error (module), 16
oled.mixin (module), 17
oled.render (module), 17
oled.serial (module), 17
oled.threadpool (module), 18
oled.virtual (module), 19

P

paste_into() (oled.virtual.hotspot method), 19
paste_into() (oled.virtual.snapshot method), 19
preprocess() (oled.mixin.capabilities method), 17
println() (oled.virtual.terminal method), 20
putch() (oled.virtual.terminal method), 20
puts() (oled.virtual.terminal method), 20
pygame (class in oled.emulator), 16

R

range_overlap() (in module oled.virtual), 19
refresh() (oled.virtual.viewport method), 20
remove_hotspot() (oled.virtual.viewport method), 20
restore() (oled.virtual.history method), 19
run() (oled.threadpool.worker method), 18

S

savepoint() (oled.virtual.history method), 19
scale2x() (oled.emulator.transformer method), 16
set_position() (oled.virtual.viewport method), 21
sh1106 (class in oled.device), 14
should_redraw() (oled.virtual.hotspot method), 19
should_redraw() (oled.virtual.snapshot method), 19
show() (oled.device.device method), 14
smoothscale() (oled.emulator.transformer method), 16
snapshot (class in oled.virtual), 19
spi (class in oled.serial), 18
ssd1306 (class in oled.device), 14
ssd1325 (class in oled.device), 14
ssd1331 (class in oled.device), 14

T

tab() (oled.virtual.terminal method), 20
terminal (class in oled.virtual), 20
threadpool (class in oled.threadpool), 18
to_surface() (oled.emulator.emulator method), 15
transformer (class in oled.emulator), 16

U

update() (oled.virtual.hotspot method), 19

V

viewport (class in oled.virtual), 20

W

wait_completion() (oled.threadpool.threadpool method), 18
worker (class in oled.threadpool), 18
write_animation() (oled.emulator.gifanim method), 16