Luma.Core Documentation

Release 1.8.2

Richard Hull and contributors

Dec 22, 2018

Contents

CHAPTER 1

Introduction

luma.core is a component library providing a Pillow-compatible drawing canvas, and other functionality to support drawing primitives and text-rendering capabilities for small displays on the Raspberry Pi and other single board computers:

- scrolling/panning capability,
- terminal-style printing,
- state management,
- color/greyscale (where supported),
- dithering to monochrome,
- sprite animation,
- flexible framebuffering (depending on device capabilities)

Device drivers extend **luma.core** to provide the correct initialization sequences for specific physical display devices/chipsets.

There are several drivers for different classes of device available:

- luma.oled
- luma.lcd
- luma.led_matrix

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 directory:

CHAPTER 2

Installation

There is no need to directly install this package - it should get installed automatically as a dependency when installing a specific display driver. See the instructions in the relevant documentation

CHAPTER 3

API Documentation

3.1 luma.core.ansi_color

ANSI Escape-code parser

New in version 0.5.5.

luma.core.ansi_color.**find_directives**(*text*, *klass*)
Find directives on class `klass` in string `text`.

Returns list of `(method, args)` tuples.

New in version 0.9.0.

Parameters **text** (*str*) – String containing directives.

Return type list

luma.core.ansi_color.**parse_str**(*text*)
Given a string of characters, for each normal ASCII character, yields a directive consisting of a 'putch' instruction followed by the character itself.

If a valid ANSI escape sequence is detected within the string, the supported codes are translated into directives. For example `\033[42m` would emit a directive of `["background_color", "green"]`.

Note that unrecognised escape sequences are silently ignored: Only reset, reverse colours and 8 foreground and background colours are supported.

It is up to the consumer to interpret the directives and update its state accordingly.

Parameters **text** (*str*) – An ASCII string which may or may not include valid ANSI Color escape codes.

luma.core.ansi_color.**strip_ansi_codes**(*text*)
Remove ANSI color codes from the string `text`.

New in version 0.9.0.

Parameters **text** (*str*) – String containing ANSI color codes.

Return type str

3.2 luma.core.cmdline

luma.core.cmdline.**create_device**(*args*, *display_types=None*)
Create and return device.

luma.core.cmdline.**create_parser**(*description*)
Create and return command-line argument parser.

luma.core.cmdline.**get_choices**(*module_name*)
Retrieve members from module_name's __all__ list.

Return type list

luma.core.cmdline.**get_display_types**()
Get ordered dict containing available display types from available luma sub-projects.

Return type collections.OrderedDict

luma.core.cmdline.**get_interface_types**()
Get list of available interface types, e.g. ['spi', 'i2c'].

Return type list

luma.core.cmdline.**get_library_for_display_type**(*display_type*)
Get library name for display_type, e.g. ssd1306 should return oled.

New in version 1.2.0.

Parameters **display_type** (*str*) – Display type, e.g. ssd1306.

Return type str or None

luma.core.cmdline.**get_library_version**(*module_name*)
Get version number from module_name's __version__ attribute.

New in version 1.2.0.

Parameters **module_name** (*str*) – The module name, e.g. luma.oled.

Return type str

luma.core.cmdline.**get_supported_libraries**()
Get list of supported libraries for the parser.

Return type list

luma.core.cmdline.**get_transformer_choices**()

Return type list

luma.core.cmdline.**load_config**(*path*)
Load device configuration from file path and return list with parsed lines.

Parameters **path** (*str*) – Location of configuration file.

Return type list

class luma.core.cmdline.**make_serial**(*opts*, *gpio=None*)
Bases: object

Serial factory.

i2c()

spi()

3.3 luma.core.const

class luma.core.const.**common**
Bases: object

DISPLAYALLON = 165

DISPLAYALLON_RESUME = 164

DISPLAYOFF = 174

DISPLAYON = 175

INVERTDISPLAY = 167

NORMALDISPLAY = 166

SETCONTRAST = 129

SETMULTIPLEX = 168

SETREMAP = 160

3.4 luma.core.device

class luma.core.device.**device**(*const=None*, *serial_interface=None*)
Bases: *luma.core.mixin.capabilities*

Base class for display driver classes

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

---

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*)
Should be overridden in sub-classed implementations.

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

Raises **NotImplementedError** –

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.core.device.**dummy**(*width=128*, *height=64*, *rotate=0*, *mode='RGB'*, ***kwargs*)
Bases: *luma.core.device.device*

Pseudo-device that acts like a physical display, except that it does nothing other than retain a copy of the displayed image. It is mostly useful for testing. Supports 24-bit color depth.

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 `PIL.Image` and makes a copy of it for later use/inspection.

Parameters **image** (*PIL.Image.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.

3.5 luma.core.error

Exceptions for this library.

exception luma.core.error.**DeviceAddressError**
Bases: *luma.core.error.Error*

Exception raised when an invalid device address is detected.

exception luma.core.error.**DeviceDisplayModeError**
Bases: *luma.core.error.Error*

Exception raised when an invalid device display mode is detected.

exception luma.core.error.**DeviceNotFoundError**
Bases: *luma.core.error.Error*

Exception raised when a device cannot be found.

exception luma.core.error.**DevicePermissionError**
Bases: *luma.core.error.Error*

Exception raised when permission to access the device is denied.

exception luma.core.error.**Error**
Bases: exceptions.Exception

Base class for exceptions in this library.

exception luma.core.error.**UnsupportedPlatform**
Bases: *luma.core.error.Error*

Exception raised when trying to use the library on an incompatible system.

New in version 0.5.2.

3.6 luma.core.framebuffer

Different implementation strategies for framebuffering

class luma.core.framebuffer.**diff_to_previous**(*device*)
Bases: object

Compare the current frame to the previous frame and tries to calculate the differences: this will either be None for a perfect match or some bounding box describing the areas that are different, up to the size of the entire image.

The image data for the difference is then be passed to a device for rendering just those small changes. This can be very quick for small screen updates, but suffers from variable render times, depending on the changes applied. The *luma.core.sprite_system.framerate_regulator* may be used to counteract this behavior however.

Parameters **device** (luma.core.device.device) – The target device, used to determine the initial 'previous' image.

getdata()
A sequence of pixel data relating to the changes that occurred since the last time *redraw_required()* was last called.

Returns A sequence of pixels or None.

Return type iterable

inflate_bbox()
Realign the left and right edges of the bounding box such that they are inflated to align modulo 4.

This method is optional, and used mainly to accommodate devices with COM/SEG GDDRAM structures that store pixels in 4-bit nibbles.

redraw_required(*image*)
Calculates the difference from the previous image, return a boolean indicating whether a redraw is required. A side effect is that bounding_box and image attributes are updated accordingly, as is priming *getdata()*.

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

Returns `True` or `False`

Return type bool

class luma.core.framebuffer.**full_frame**(*device*)
Bases: object

Always renders the full frame every time. This is slower than *diff_to_previous* as there are generally more pixels to update on every render, but it has a more consistent render time. Not all display drivers may be able to use the differencing framebuffer, so this is provided as a drop-in replacement.

Parameters **device** (luma.core.device.device) – The target device, used to determine the bounding box.

getdata()
A sequence of pixels representing the full image supplied when the *redraw_required()* method was last called.

Returns A sequence of pixels.

Return type iterable

inflate_bbox()
Just return the original bounding box without any inflation.

redraw_required(*image*)
Caches the image ready for getting the sequence of pixel data with *getdata()*. This method always returns affirmatively.

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

Returns `True` always.

3.7 luma.core.image_composition

Composes scrollable, positionable images into another Image

New in version 1.1.0.

class luma.core.image_composition.**ComposableImage**(*image*, *position=(0, 0)*, *offset=(0, 0)*)
Bases: object

This class encapsulates an image and its attributes that can be rendered onto an *ImageComposition*.

height

Returns The actual height of the image, regardless its position or offset within the image composition.

Return type int

image(*size*)

Parameters **size** (*tuple*) – The width, height of the image composition.

Returns An image, cropped to the boundaries specified by `size`.

Return type PIL.Image.Image

offset
Getter for offset.

Returns A tuple containing the top,left position.

Return type tuple

position
Getter for position

Returns A tuple containing the `x, y` position.

Return type tuple

width

Returns The actual width of the image, regardless its position or offset within the image composition.

Return type int

class luma.core.image_composition.**ImageComposition**(*device*)
Bases: object

Manages a composition of *ComposableImage* instances that can be rendered onto a single PIL.Image.Image.

add_image(*image*)
Adds an image to the composition.

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

refresh()
Clears the composition and renders all the images taking into account their position and offset.

remove_image(*image*)
Removes an image from the composition.

Parameters **image** (*PIL.Image.Image*) – The image to be removed.

3.8 luma.core.interface.serial

Encapsulates sending commands and data over a serial interface, whether that is I²C, SPI or bit-banging GPIO.

class luma.core.interface.serial.**i2c**(*bus=None*, *port=1*, *address=60*)
Bases: object

Wrap an I²C (Inter-Integrated Circuit) interface to provide *data()* and *command()* methods.

Parameters

- **bus** – A *smbus* implementation, if None is supplied (default), smbus2 is used. Typically this is overridden in tests, or if there is a specific reason why pysmbus must be used over smbus2.
- **port** (*int*) – I²C port number, usually 0 or 1 (default).
- **address** (*int*) – I²C address, default: 0x3C.

Raises

- ***luma.core.error.DeviceAddressError*** – I2C device address is invalid.
- ***luma.core.error.DeviceNotFoundError*** – I2C device could not be found.
- ***luma.core.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.

Parameters **cmd** (*int*) – A spread of commands.

Raises ***luma.core.error.DeviceNotFoundError*** – I2C device could not be found.

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.

Parameters **data** (*list, bytearray*) – A data sequence.

class luma.core.interface.serial.**spi**(*spi=None*, *gpio=None*, *port=0*, *device=0*, *bus_speed_hz=8000000*, *cs_high=False*, *transfer_size=4096*, *gpio_DC=24*, *gpio_RST=25*)
Bases: *luma.core.interface.serial.bitbang*

Wraps an SPI (Serial Peripheral Interface) bus to provide `data()` and `command()` methods.

Parameters

- **spi** – SPI implementation (must be compatible with spidev)
- **gpio** – GPIO interface (must be compatible with RPi.GPIO). For slaves that don't need reset or D/C functionality, supply a `noop` implementation instead.
- **port** (*int*) – SPI port, usually 0 (default) or 1.
- **device** (*int*) – SPI device, usually 0 (default) or 1.
- **bus_speed_hz** (*int*) – SPI bus speed, defaults to 8MHz.
- **cs_high** (*bool*) – Whether SPI chip select is high, defaults to `False`.
- **transfer_size** (*int*) – Maximum amount of bytes to transfer in one go. Some implementations only support a maximum of 64 or 128 bytes, whereas RPi/py-spidev supports 4096 (default).
- **gpio_DC** (*int*) – The GPIO pin to connect data/command select (DC) to (defaults to 24).
- **gpio_RST** (*int*) – The GPIO pin to connect reset (RES / RST) to (defaults to 25).

Raises

- ***luma.core.error.DeviceNotFoundError*** – SPI device could not be found.
- ***luma.core.error.UnsupportedPlatform*** – GPIO access not available.

cleanup()
Clean up SPI & GPIO resources.

class luma.core.interface.serial.**bitbang**(*gpio=None*, *transfer_size=4096*, ***kwargs*)
Bases: `object`

Wraps an SPI (Serial Peripheral Interface) bus to provide *data()* and *command()* methods. This is a software implementation and is thus a lot slower than the default SPI interface. Don't use this class directly unless there is a good reason!

Parameters

- **gpio** – GPIO interface (must be compatible with RPi.GPIO). For slaves that don't need reset or D/C functionality, supply a `noop` implementation instead.
- **transfer_size** (*int*) – Max bytes to transfer in one go. Some implementations only support maximum of 64 or 128 bytes, whereas RPi/py-spidev supports 4096 (default).
- **SCLK** (*int*) – The GPIO pin to connect the SPI clock to.
- **SDA** (*int*) – The GPIO pin to connect the SPI data (MOSI) line to.
- **CE** (*int*) – The GPIO pin to connect the SPI chip enable (CE) line to.
- **DC** (*int*) – The GPIO pin to connect data/command select (DC) to.
- **RST** (*int*) – The GPIO pin to connect reset (RES / RST) to.

cleanup()
Clean up GPIO resources if managed.

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

Parameters **cmd** (*int*) – A spread of commands.

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

Parameters **data** (*list, bytearray*) – A data sequence.

3.9 luma.core.legacy

These methods were originally present in the old MAX7219 driver, and are preserved here only to aid migration away from that library. The functions are denoted 'legacy' to discourage use - you are encouraged to use the various drawing capabilities in the Pillow library instead.

luma.core.legacy.**show_message**(*device*, *msg*, *y_offset=0*, *fill=None*, *font=None*, *scroll_delay=0.03*)
Scrolls a message right-to-left across the devices display.

Parameters

- **device** – The device to scroll across.
- **msg** (*str*) – The text message to display (must be ASCII only).
- **y_offset** (*int*) – The row to use to display the text.
- **fill** – The fill color to use (standard Pillow color name or RGB tuple).
- **font** – The font (from *luma.core.legacy.font*) to use.
- **scroll_delay** (*float*) – The number of seconds to delay between scrolling.

luma.core.legacy.**text**(*draw*, *xy*, *txt*, *fill=None*, *font=None*)
Draw a legacy font starting at `x`, `y` using the prescribed fill and font.

Parameters

- **draw** (*PIL.ImageDraw*) – A valid canvas to draw the text onto.
- **txt** (*str*) – The text string to display (must be ASCII only).
- **xy** (*tuple*) – An `(x, y)` tuple denoting the top-left corner to draw the text.
- **fill** – The fill color to use (standard Pillow color name or RGB tuple).
- **font** – The font (from *luma.core.legacy.font*) to use.

luma.core.legacy.**textsize**(*txt*, *font=None*)

Calculates the bounding box of the text, as drawn in the specified font. This method is most useful when the *proportional* wrapper is used.

Parameters

- **txt** (*str*) – The text string to calculate the bounds for
- **font** – The font (from *luma.core.legacy.font*) to use.

3.10 luma.core.legacy.font

Fixed-width font definitions. The following fonts are available:

Font name	Notes	Source
CP437_FONT	See https://en.wikipedia.org/wiki/Code_page_437 for further details.	*unascribed*
LCD_FONT	Only contains characters 0x20-0x7F inclusive and Cyrillic chars 0x80-0xBF (except ‘’); all others will appear as blanks.	http://www.avrfreaks.net/forum/code-57-512-and-712-fonts?name=PNphpBB2&file=viewtopic&t=69880
SEG7_FONT	Only contains characters 0x41-0x5A & 0x30-0x39 inclusive; all others will appear as blanks.	https://www.dafont.com/digital-7.font
SINCLAIR_FONT	Based on the character set from the Sinclair ZX Spectrum. Only contains characters 0x20-0x7E inclusive; all others will appear as blanks.	http://www.henningkarlsen.com/electronics/r_fonts.php
TINY_FONT	Only contains characters 0x20-0x7E inclusive; all others will appear as blanks.	http://www.dafont.com/tiny.font
UKR_FONT	Cyrillic Ukrainian font	*unascribed*

class luma.core.legacy.font.**proportional**(*font*)

Bases: object

Wraps an existing font array, and on on indexing, trims any leading or trailing zero column definitions. This works especially well with scrolling messages, as interspace columns are squeezed to a single pixel.

class luma.core.legacy.font.**tolerant**(*font*, *missing='_'*)

Bases: object

Wraps an existing font array, and on indexing, if the supplied ascii_code does not exist, will return the column definitions for the given `missing` parameter.

New in version 0.9.4.

3.11 luma.core.mixin

class luma.core.mixin.**capabilities**
Bases: object

This class should be 'mixed-in' to any *luma.core.device.device* display implementation that should have "device-like" capabilities.

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.

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

display(*image*)
Should be overridden in sub-classed implementations.

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

Raises **NotImplementedError** –

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

3.12 luma.core.render

class luma.core.render.**canvas**(*device*, *background=None*, *dither=False*)
Bases: object

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 *generally* treated as white when displayed on monochrome devices. 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. If a `background` parameter is provided, the canvas is based on the given background. This is useful to e.g. write text on a given background image.

3.13 luma.core.sprite_system

Simplified sprite animation framework.

Note: this module is an evolving "work-in-progress" and should be treated as such until such time as this notice disappears.

class luma.core.sprite_system.**dict_wrapper**(*d*)
Bases: object

Helper class to turn dictionaries into objects.

class luma.core.sprite_system.**framerate_regulator**(*fps=16.67*)
Bases: object

Implements a variable sleep mechanism to give the appearance of a consistent frame rate. Using a fixed-time sleep will cause animations to be jittery (looking like they are speeding up or slowing down, depending on what other work is occurring), whereas this class keeps track of when the last time the `sleep()` method was called, and calculates a sleep period to smooth out the jitter.

Parameters **fps** (*float*) – The desired frame rate, expressed numerically in frames-per-second. By default, this is set at 16.67, to give a frame render time of approximately 60ms. This can be overridden as necessary, and if no FPS limiting is required, the `fps` can be set to zero.

average_transit_time()
Calculates the average transit time between the enter and exit methods, and return the time in milliseconds.

Returns The average transit in milliseconds.

Return type float

effective_FPS()
Calculates the effective frames-per-second - this should largely correlate to the desired FPS supplied in the constructor, but no guarantees are given.

Returns The effective frame rate.

Return type float

class luma.core.sprite_system.**spritesheet**(*image*, *frames*, *animations*)
Bases: object

A sprite sheet is a series of images (usually animation frames) combined into a larger image. A dictionary is usually spread into the object constructor parameters with the following top-level attributes:

Parameters

- **image** (*str*) – A path to a sprite map image.
- **frames** (*dict*) – A dictionary of settings that defines how to extract individual frames from the supplied image, as follows
  - `width` & `height` are required and specify the dimensions of the frames
  - `regX` & `regY` indicate the registration point or "origin" of the frames
  - `count` allows you to specify the total number of frames in the spritesheet; if omitted, this will be calculated based on the dimensions of the source images and the frames. Frames will be assigned indexes based on their position in the source images (left to right, top to bottom).

- **animations** (*dict*) – A dictionary of key/value pairs where the key is the name of of the animation sequence, and the value are settings that defines an animation sequence as follows:
  - `frames` is a list of frame to show in sequence. Usually this comprises of frame numbers, but can refer to other animation sequences (which are handled much like a subroutine call).
  - `speed` determines how quickly the animation frames are cycled through compared to the how often the animation sequence yields.
  - `next` is optional, but if supplied, determines what happens when the animation sequence is exhausted. Typically this can be used to self-reference, so that it forms an infinite loop, but can hand off to any other animation sequence.

Loosely based on http://www.createjs.com/docs/easeljs/classes/SpriteSheet.html

animate(*seq_name*)

Returns a generator which "executes" an animation sequence for the given `seq_name`, inasmuch as the next frame for the given animation is yielded when requested.

Parameters **seq_name** (*str*) – The name of a previously defined animation sequence.

Returns A generator that yields all frames from the animation sequence.

Raises **AttributeError** – If the `seq_name` is unknown.

3.14 luma.core.threadpool

class luma.core.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 luma.core.threadpool.**worker**(*tasks*)

Bases: `threading.Thread`

Thread executing tasks from a given tasks queue.

run()

Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

3.15 luma.core.util

class luma.core.util.**mutable_string**(*value*)

Bases: `object`

class luma.core.util.**observable**(*target*, *observer*)

Bases: `object`

Wraps any container object such that on inserting, updating or deleting, an observer is notified with a payload of the target. All other special name methods are passed through parameters unhindered.

3.16 luma.core.virtual

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

class luma.core.virtual.**history**(*device*)
Bases: *luma.core.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*)
Should be overridden in sub-classed implementations.

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

Raises **NotImplementedError** –

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.

Parameters **drop** (*int*) –

savepoint()
Copies the last displayed image.

class luma.core.virtual.**hotspot**(*width*, *height*, *draw_fn=None*)
Bases: *luma.core.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 *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*)

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

class luma.core.virtual.**sevensegment**(*device*, *undefined='_'*, *segment_mapper=None*)
Bases: object

Abstraction that wraps a device, this class provides a `text` property which can be used to set and get a text value, which when combined with a `segment_mapper` sets the correct bit representation for seven-segment displays and propagates that onto the underlying device.

Parameters

- **device** – A device instance.
- **segment_mapper** – An optional function that maps strings into the correct representation for the 7-segment physical layout. If not provided, the default mapper from compatible devices is used instead.
- **undefined** (*char*) – The default character to substitute when an unrenderable character is supplied to the text property.

text
Returns the current state of the text buffer. This may not reflect accurately what is displayed on the seven-segment device, as certain alpha-numerics and most punctuation cannot be rendered on the limited display.

class luma.core.virtual.**snapshot**(*width*, *height*, *draw_fn=None*, *interval=1.0*)
Bases: *luma.core.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 luma.core.virtual.**terminal**(*device*, *font=None*, *color='white'*, *bgcolor='black'*, *tabstop=4*, *line_height=None*, *animate=True*, *word_wrap=False*)
Bases: object

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

background_color(*value*)
Sets the background color.

Parameters **value** (*str or tuple*) – The new color value, either string name or RGB tuple.

backspace()
Moves the cursor one place to the left, erasing the character at the current position. Cannot move beyond 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 position without moving the cursor's position.

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

foreground_color(*value*)
Sets the foreground color.

Parameters **value** (*str or tuple*) – The new color value, either string name or RGB tuple.

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.

Parameters **text** (*str*) – The text to print.

putch(*char*)
Prints the specific character, which must be a valid printable ASCII value in the range 32..127 only, or one of carriage return (r), newline (n), backspace (b) or tab (t).

Parameters **char** – The character to print.

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

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.

Parameters **text** (*str*) – The text to print.

reset()
Resets the foreground and background color value back to the original when initialised.

reverse_colors()
Flips the foreground and background colors.

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

class luma.core.virtual.**viewport**(*device*, *width*, *height*)
Bases: *luma.core.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*)
Should be overridden in sub-classed implementations.

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

Raises **NotImplementedError** –

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 for `(x, y)`, a `ValueError` is raised.

set_position(*xy*)

CHAPTER 4

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.

4.1 GitHub

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

4.2 Contributors

- Thijs Triemstra (@thijstriemstra)
- Christoph Handel (@fragfutter)
- @Boeeerb
- @xes
- Roger Dahl (@rogerdahl)
- Václav Šmilauer (@eudoxos)
- Claus Bjerre (@bjerrep)
- @bkntrx
- @7754359337
- @theraspydev
- @vortigont
- Maarten Los (@mlos)

- Jonathan Pereira (@jonathanrjpereira)
- Daniel Smullen (@drspangle)
- Hans Liss (@hansliss)
- Phil Howard (@gadgetoid)

CHAPTER 5

ChangeLog

Version	Description	Date
1.8.2	• Fix type hint for SPI's cs_high parameter	2018/11/05
1.8.1	• Mutable string now works over unicode (for both py2/3)	2018/09/18
1.8.0	• Namespace packaging fix • Correct implementation of pkgutil style namespace • Support for Python 3.7 • Docstring updates	2018/09/04
1.7.2	• Fix upside-down SEG7_FONT	2018/03/29
1.7.1	• Support unicode in terminal class	2018/03/22
1.7.0	• Add `persist` flag on device	2018/03/21

Continued on next page

Table 1 – continued from previous page

Version	Description	Date
1.6.0	• Add `--spi-transfer-size=...` flag in cmdline args	2018/02/21
1.5.0	• Add SEG7_FONT: Compact 7x3 font for LED Matrix	2018/02/06
1.4.0	• Add `--spi-cs-high=...` flag in cmdline args	2018/01/29
1.3.0	• Add `--gpio-mode=...` flag in cmdline args	2018/01/02
1.2.1	• Use `extras_require` in `setup.py` for Linux dependencies	2017/11/26
1.2.0	• Added `get_library_version` & `get_library_for_display_type`	2017/11/23
1.1.1	• Version number available as `luma.core.__version__` now	2017/11/23
1.1.0	• Added image composition classes	2017/10/28
1.0.3	• Explicitly state 'UTF-8' encoding in setup when reading files	2017/10/18
1.0.2	• Fix conditional install on wheel	2017/09/15
1.0.1	• Don't install RPi.GPIO & spidev if setup running on OSX	2017/09/04

Continued on next page

Table 1 – continued from previous page

Version	Description	Date
1.0.0	• Stable release (remove all deprecated methods & parameters)	2017/07/29
0.9.5	• Remove assert in `terminal` to allow extended characters to be printed (note: this only works for Python3 presently)	2017/07/06
0.9.4	• Add `tolerant` class for legacy font handling non-ASCII chars • Add CP437 chars to fonts.py	2017/07/01
0.9.3	• LCD_FONT: lowercase cyrillic chars added, minor corrections in uppercase chars	2017/06/25
0.9.2	• Add `background=` option to `luma.core.render.canvas` • Add TCA9548A I2C multiplex scanner (contrib) • Display I2C address in hex when error occurs	2017/06/19
0.9.1	• Add cmdline block orientation of 180	2017/05/01
0.9.0	• Add word-wrap capability to `luma.core.virtual.terminal` • Bug fix to `luma.core.virtual.terminal` when scrolling	2017/04/22
0.8.1	• Propagate segment_mapper through other virtual devices	2017/04/14

Continued on next page

Table 1 – continued from previous page

Version	Description	Date
0.8.0	• Migrate seven-segment wrapper from `luma.led_matrix`	2017/04/14
0.7.5	• Allow alternative RPi.GPIO implementations	2017/04/09
0.7.4	• Reduce size of space character in legacy proportional font	2017/04/09
0.7.3	• Cmdline args now supports backlight active high/low	2017/04/07
0.7.2	• Add `--h-offset=N` and `--v-offset=N` params to cmdline args	2017/04/07
0.7.1	• Improve formatting in command line options	2017/04/06
0.7.0	• Add software-based bitbang SPI implementation • Cmdline args parsing • Use monotonic clock	2017/03/27
0.6.2	• Move GPIO.setmode() to point when referenced • Use regex prefix in ANSI color parser (fixes deprecation warning)	2017/03/19
0.6.1	• Deprecate spi params • Fix resource leak in spritesheet	2017/03/13
0.6.0	• Terminal supports ANSI Color escape codes • Catch & rethrow IOErrors	2017/03/13

Continued on next page

Table 1 – continued from previous page

Version	Description	Date
0.5.4	• Rework decorators	2017/03/08
0.5.3	• Catch & rethrow all RPi.GPIO RuntimeExceptions	2017/03/08
0.5.2	• Raise `error.UnsupportedPlatform` if RPi.GPIO is not available • Bug fix to `luma.core.virtual.terminal` to handle multiple \n	2017/03/08
0.5.1	• Bug fix: `legacy.show_message` regression	2017/03/05
0.5.0	• **BREAKING CHANGES:** Rework `framework_regulator` class • Documentation updates	2017/03/05
0.4.4	• Bug fix: `legacy.show_message` off-by-one bug	2017/03/02
0.4.3	• Restrict exported Python symbols from `luma.core.serial`	2017/03/02
0.4.2	• Optional alignment of framebuffer bounding_box to word-boundaries	2017/02/28
0.4.1	• Refactor framebuffer interface	2017/02/27

Continued on next page

Table 1 – continued from previous page

Version	Description	Date
0.4.0	• Add spritesheet and framerate_regulator functionality • Add full-frame and diff-to-previous framebuffer implementations • Remove unnecessary travis/tox dependencies	2017/02/27
0.3.2	• Bug fix: `legacy.show_message` wrong device height • Add Cyrillic chars to legacy font • Make pytest-runner a conditional requirement	2017/02/24
0.3.1	• Imported legacy font handling from `rm-hull/luma.led_matrix`	2017/02/19
0.2.0	• Fix bug in seven_segment transform (display correct char) • Moved emulator code to `rm-hull/luma.emulator` github repo	2017/02/17
0.1.15	• Require at least Pillow 4.0.0 • Configurable `transfer_size` on SPI writes • Documentation updates	2017/02/11
0.1.14	• Use a more flexible no-op implementation • Use spidev's `writebytes()` rather than `xfer2()` • Dont write GIF animation if nothing was displayed • Attempt to optimize palette when saving GIF animations	2017/02/03

Continued on next page

Table 1 – continued from previous page

Version	Description	Date
0.1.13	• Fix bug in setup script	2017/01/23
0.1.12	• Assert valid SPI bus speed • Don't report errors in shutdown • Don't package as zip-safe • Add 7-segment LED emulation transformer	2017/01/21
0.1.11	• Rejig packaging to include emulator assets	2017/01/20
0.1.3	• Reset SPI device on initialization • Add LED matrix emulation transformer	2017/01/19
0.1.2	• Namespace packaging	2017/01/10
0.1.0	• Split out core functionality from `rm-hull/ssd1306`	2017/01/10

CHAPTER 6

The MIT License (MIT)

Python Module Index

l

Index

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W