
Luma.LCD Documentation

Release 2.9.0

Richard Hull and contributors

Mar 14, 2021

CONTENTS

1	Introduction	3
1.1	Supported Devices	3
1.2	Examples and Emulators	9
2	Installation	11
3	Hardware	13
3.1	I2C vs. SPI vs Parallel	13
3.2	Identifying your interface	13
3.3	I2C	14
3.4	SPI	16
3.5	Parallel	17
3.6	Suggested Wiring	18
4	Software	23
4.1	Installing Dependencies	23
4.2	Permissions	24
5	Python Usage	25
5.1	Color Model	26
5.2	Landscape / Portrait Orientation	26
5.3	Seven-Segment Drivers	26
5.4	Backlight Control	27
5.5	Examples	27
6	HD44780	29
6.1	Introduction	29
6.2	Capabilities	30
7	API Documentation	35
7.1	Upgrading	35
7.2	<code>luma.lcd.device</code>	35
8	References	49
9	Contributing	51
9.1	GitHub	51
9.2	Contributors	51
10	ChangeLog	53

11 The MIT License (MIT)	57
Python Module Index	59
Index	61

INTRODUCTION

`luma.lcd` provides a Python3 interface to small LCD displays connected to Raspberry Pi and other Linux-based single-board computers (SBC). It provides a Pillow-compatible drawing canvas, and other functionality to support:

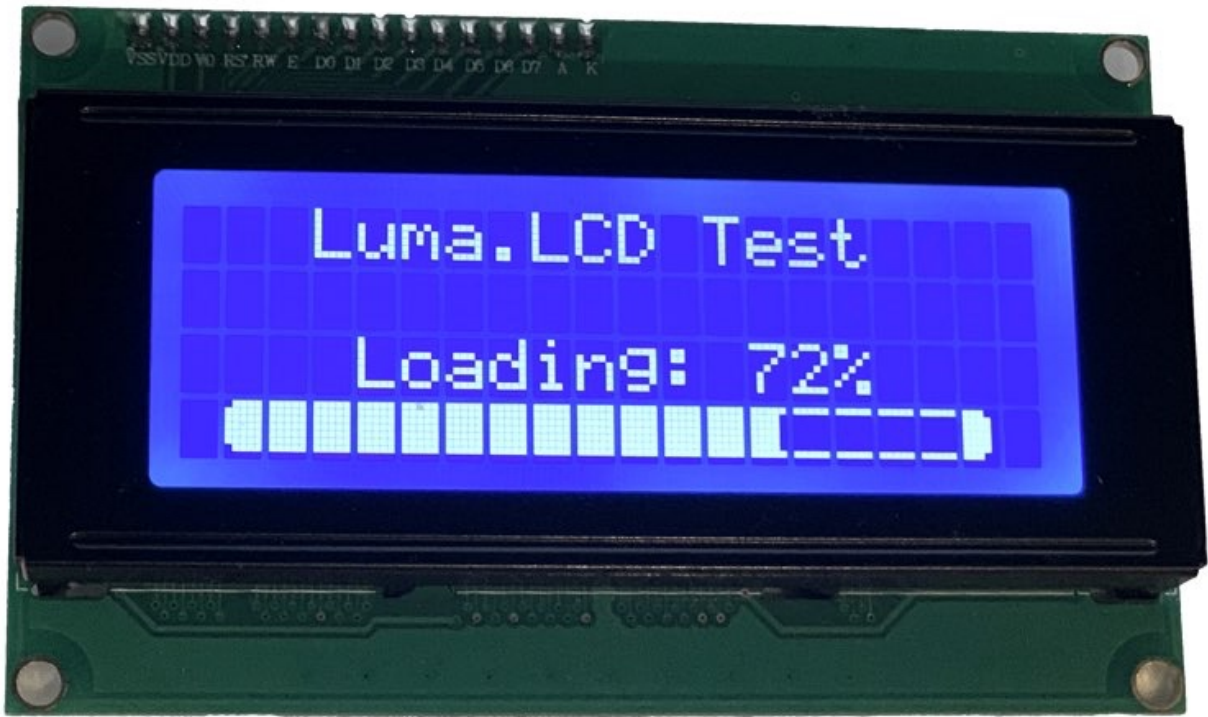
- scrolling/panning capability,
- terminal-style printing,
- state management,
- color/greyscale (where supported),
- dithering to monochrome

1.1 Supported Devices

The library currently supports devices using the HD44780, PCD8544, ST7735, HT1621, and UC1701X controllers.

1.1.1 HD44780

The HD44780 (and similar) devices are some of the most popular small LCD displays available for SBCs. These are character-based displays but the `luma.lcd` driver supports a limited ability to display graphical content on them. See the [HD44780](#) documentation for details.



1.1.2 PCD85744

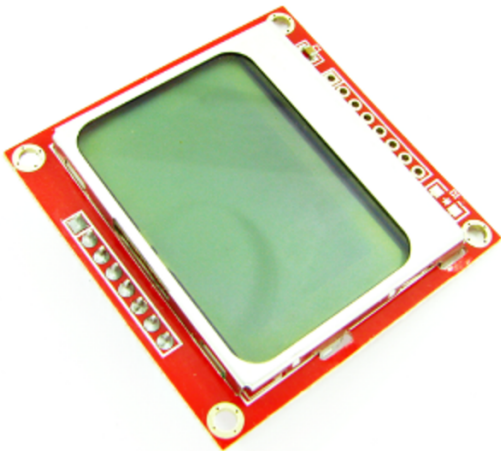
The PCD8544 display pictured below was used originally as the display for [Nokia 5110](#) mobile phones, supporting a resolution of 84 x 48 monochrome pixels and a switchable backlight:

Features:

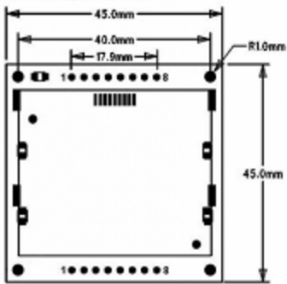
- 1) Built-in Backlight
- 2) Easy communication with common MCU control
- 3) Philips PCD8544 LCD controller with SPI interface
- 4) Graphic LCD module with 84X48 pixel resolution.
- 5) Compatible to Nokia 5110, 3310 LCD

Specification:

Interface	SPI serial connection
Operating voltage	2.7V to 3.3V
Operating current	<5mA (Backlight off), <20mA (Backlight on)
Operating temperature	0 to 50 Degree Celsius
Storage temperature	-10 to 70 Degree Celsius
Size (L x W x H)	45X45X5mm
LCD Controller	Philips PCD8544



Pin Assignment:

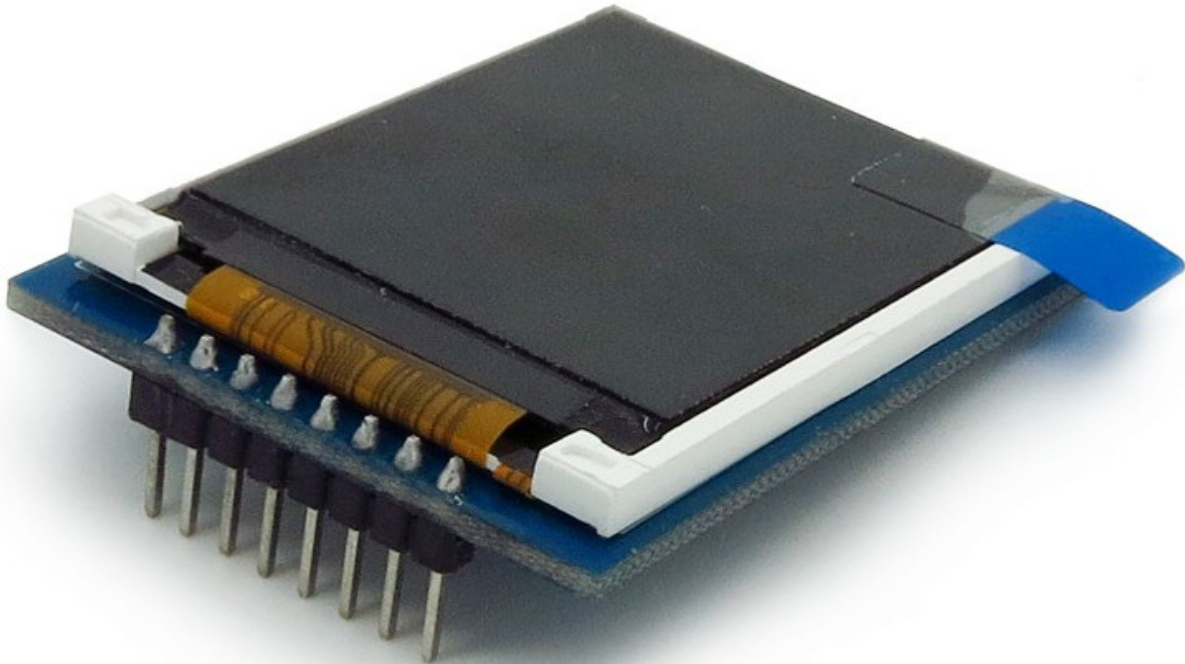


Pin	Name	Description
1	VCC	2.7 to 3.3V
2	GND	Ground
3	SCE	Chip enable (Active Low)
4	RES	Reset (Active Low)
5	D/C	Data/Command selection Low— Write command, High— Write data.
6	SDIN	Serial input
7	SCLK	Clock input
8	LED	Active High 2.7 to 3.2V

They are now commonly recycled, and sold on ebay with a breakout board and SPI interface.

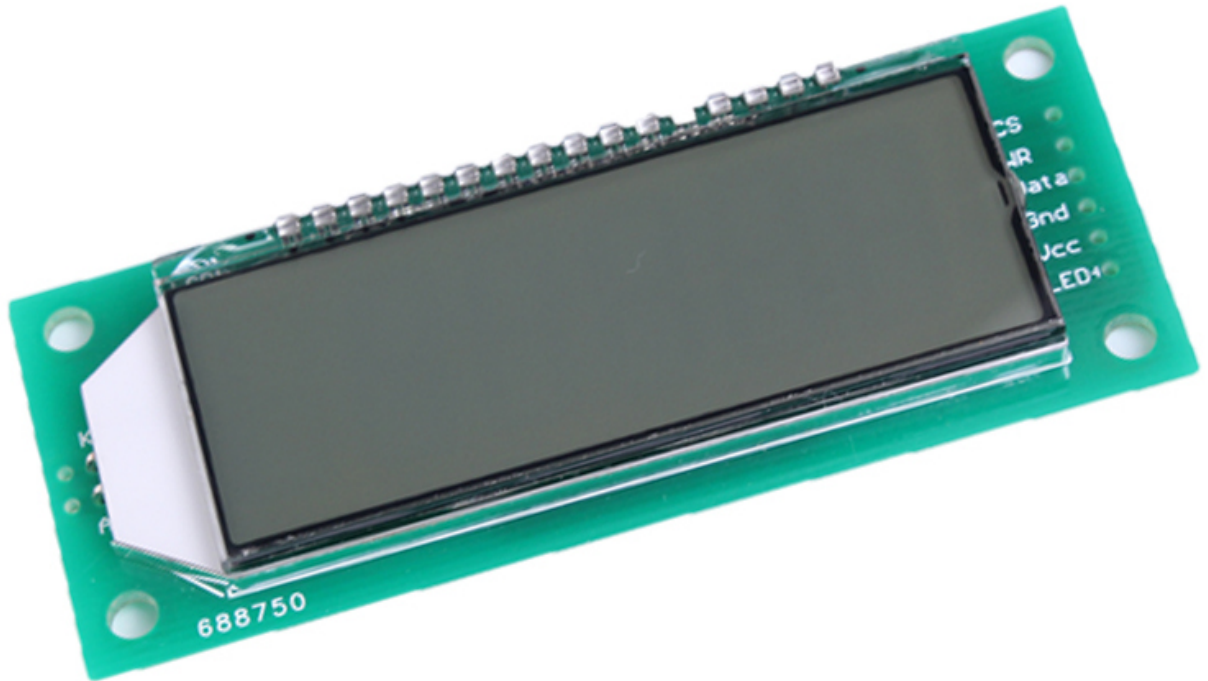
1.1.3 ST7735

The ST7735 display supports a resolution of 160 x 128 RGB pixels (18-bit / 262K colors) with a switchable backlight:



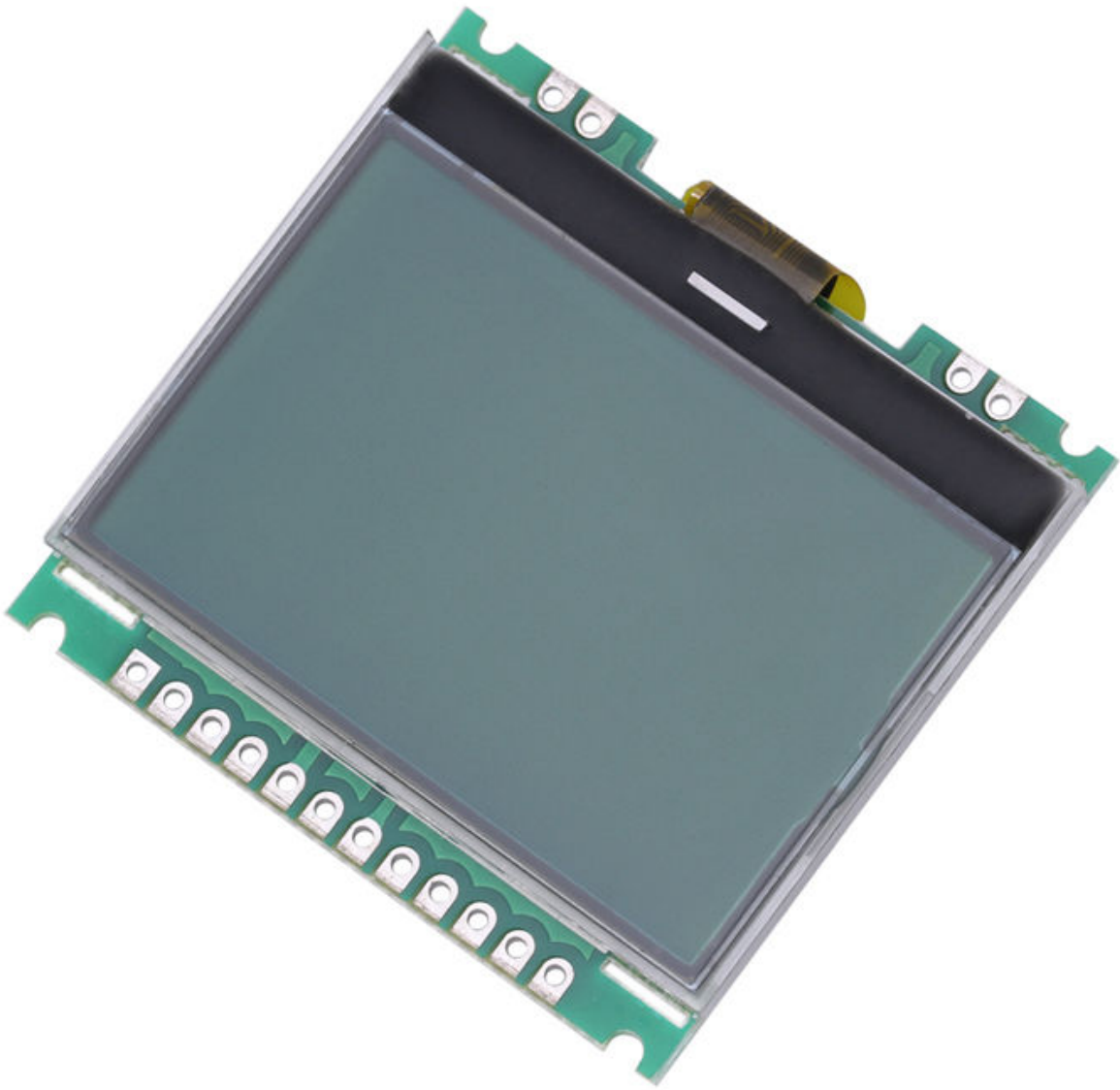
1.1.4 HT1621

The HT1621 display (as purchased) supports six 7-segment characters with a switchable backlight:



1.1.5 UC1701X

The UC1701X display supports a resolution of 128 x 64 monochrome pixels with a switchable backlight:



1.1.6 ST7567

The ST7567 display supports a resolution of 128 x 64 monochrome pixels:



See also:

Further technical information for the specific device can be found in the datasheet below:

- PCD8544
- ST7735
- ST7789
- HT1621
- UC1701X
- ILI9341
- HD44780

1.2 Examples and Emulators

As well as display drivers for the physical device, 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](#) repository).

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.lcd` 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 3.6 and newer.

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 Raspbian Jessie, Stretch and Buster operating systems.

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.

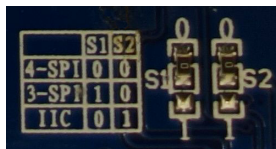
3.1 I2C vs. SPI vs Parallel

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

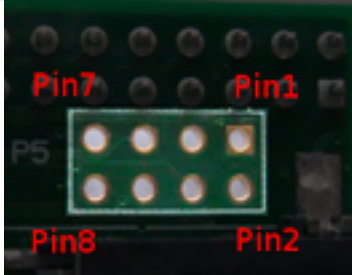
I2C interfaces are the simplest to wire as they contain the smallest number of pins. The only signal lines are serial data (SDA) and serial clock (SCL). That plus the power and ground connections complete the required connections.

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 the Raspberry Pi running Raspbian, enable the I2C as follows:

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

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

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

```
dtoverlay=i2c-arms, 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, ST7735 and ST7789).
 - 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 LCD 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 `py:class: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 5Vs to one of the GPIO pins of an SBC that uses 3.3V logic you may damage your SBC.

3.6 Suggested Wiring

3.6.1 PCD8544

LCD Pin	Remarks	RPi Pin	RPi Function
RST	Reset	P01-18	GPIO 24
CE	Chip Enable	P01-24	GPIO 8 (CE0)
DC	Data/Command	P01-16	GPIO 23
DIN	Data In	P01-19	GPIO 10 (MOSI)
CLK	Clock	P01-23	GPIO 11 (SCLK)
VCC	+3.3V Power	P01-01	3V3
LIGHT	Backlight	P01-12	GPIO 18 (PCM_CLK)
GND	Ground	P01-06	GND

3.6.2 ST7735

Depending on the board you bought, there may be different names for the same pins, as detailed below.

LCD Pin	Remarks	RPi Pin	RPi Function
GND	Ground	P01-06	GND
VCC	+3.3V Power	P01-01	3V3
RESET or RST	Reset	P01-18	GPIO 24
A0 or D/C	Data/command	P01-16	GPIO 23
SDA or DIN	SPI data	P01-19	GPIO 10 (MOSI)
SCK or CLK	SPI clock	P01-23	GPIO 11 (SCLK)
CS	SPI chip select	P01-24	GPIO 8 (CE0)
LED+ or BL	Backlight control	P01-12	GPIO 18 (PCM_CLK)
LED-	Backlight ground	P01-06	GND

3.6.3 ILI9341

No support for the touch-screen, leave the MISO and Touch pins disconnected. Depending on the board you bought, there may be different names for the same pins, as detailed below.

LCD Pin	Remarks	RPi Pin	RPi Function
VCC	+3.3V Power	P01-01	3V3
GND	Ground	P01-06	GND
CS	SPI chip select	P01-24	GPIO 8 (CE0)
RESET or RST	Reset	P01-18	GPIO 24
DC	Data/command	P01-16	GPIO 23
SDI(MOSI)	SPI data	P01-19	GPIO 10 (MOSI)
SCK or CLK	SPI clock	P01-23	GPIO 11 (SCLK)
LED	Backlight control	P01-12	GPIO 18

3.6.4 ST7567

This driver is designed for the ST7567 in 4-line SPI mode and does not include parallel bus support.

Pin names may differ across different breakouts, but will generally be something like the below.

LCD Pin	Remarks	RPi Pin	RPi Function
GND	Ground	P01-06	GND
3v3	+3.3V Power	P01-01	3V3
RESET or RST	Reset	P01-18	GPIO 24
SA0 or D/C	Data/command	P01-16	GPIO 23
SDA or DATA	SPI data	P01-19	GPIO 10 (MOSI)
SCK or CLK	SPI clock	P01-23	GPIO 11 (SCLK)
CS	SPI chip select	P01-24	GPIO 8 (CE0)

3.6.5 HT1621

LCD Pin	Remarks	RPi Pin	RPi Function
GND	Ground	P01-06	GND
VCC	+3.3V Power	P01-01	3V3
DAT	SPI data	P01-19	GPIO 10 (MOSI)
WR	SPI clock	P01-23	GPIO 11 (SCLK)
CS	SPI chip select	P01-24	GPIO 8 (CE0)
LED	Backlight control	P01-12	GPIO 18 (PCM_CLK)

3.6.6 UC1701X

The UC1701X doesn't appear to work from 3.3V, but does on the 5.0V rail.

LCD Pin	Remarks	RPi Pin	RPi Function
ROM_IN	Unused		
ROM_OUT	Unused		
ROM_SCK	Unused		
ROM_CS	Unused		
LED A	Backlight control	P01-12	GPIO 18 (PCM_CLK)
VSS	Ground	P01-06	GND
VDD	+5.0V	P01-02	5V0
SCK	SPI clock	P01-23	GPIO 11 (SCLK)
SDA	SPI data	P01-19	GPIO 10 (MOSI)
RS	Data/command	P01-16	GPIO 23
RST	Reset	P01-18	GPIO 24
CS	SPI chip select	P01-24	GPIO 8 (CE0) Chip Select

3.6.7 HD44780 Parallel

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	VDD	+5.0V Power	P01-4	5V Power
16	GND	Ground	P01-9	GND

Warning: A resister needs to be connected in series between pin 15 and the SBC to avoid excessive current draw and to control brightness. Many displays include a built-in resister for this purpose but if yours does not you will need to include an appropriately sized resister in your wiring between the SBC and pin 15 of the display.

3.6.8 HD44780 w/PCF8574

Device Pin Name	Remarks	RPi Pin	RPi Function
GND	Ground	P01-6	GND
VDD	+5.0V Power *	P01-2	5V Power
SDA	Serial Data Line	P01-3	GPIO 2 (SDA)
SCL	Serial Clock Line	P01-5	GPIO 3 (SCL)

Note: You should verify which pins from the PCF8574 are connected to the pins of the HD44780 display. There is no set standard for this wiring so different vendors will likely have used different approaches. This information will be needed when initializing `luma.core.interface.serial.pcf8574`.

SOFTWARE

Install the latest version of the library directly from [PyPI](#) with:

```
$ sudo -H pip install --upgrade luma.lcd
```

This will normally retrieve all of the dependencies `luma.lcd` 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 Raspbian Stretch or Raspberry Pi OS (Buster) 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.lcd
```

If you are not using Raspbian 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.lcd` uses hardware interfaces that require permission to access. After you have successfully installed `luma.lcd` you may want to add the user account that your `luma.lcd` 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.

PYTHON USAGE

LCD displays can be driven with python using the various implementations in the `luma.lcd.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 SPI interface with a pcd8544 display.

```
from luma.core.interface.serial import i2c, spi, parallel, pcf8574
from luma.core.render import canvas
from luma.lcd.device import pcd8544, st7735, st7789, st7567, uc1701x, ili9341, ili9486, hd44780

serial = spi(port=0, device=0, gpio_DC=23, gpio_RST=24)
device = pcd8544(serial)
```

The display device should now be properly configured.

The `pcd8544`, `st7735`, `st7789`, `st7567`, `uc1701x`, `ili9341`, `ili9486` and `hd44780` classes all expose a `display()` method which takes an image with attributes consistent with the capabilities of the device.

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="red")
```

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.

Note: The use of the display method for the HD44780 is more limited than the other LCDs. The `text` property is the preferred interface for displaying characters. See [HD44780](#) for more details.

5.1 Color Model

Any of the standard `PIL.ImageColor` color formats may be used if your device supports them. For monochrome LCDs, 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")
```

The ST7735, ST7789 and ILI9341 devices can display 262K colour RGB images. When supplying 24-bit RGB images, they are automatically downscaled to 18-bit RGB to fit these device's 262K color-space.

5.2 Landscape / Portrait Orientation

By default the PCD8544, ST7735, ST7789, UC1701X and ILI9341 displays will all be oriented in landscape mode (84x48, 160x128, 128x64 and 320x240 pixels respectively). 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 spi
from luma.core.render import canvas
from luma.lcd.device import pcd8544

serial = spi(port=0, device=0, gpio_DC=23, gpio_RST=24)
device = pcd8544(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="red")
```

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.

The HD44780 does not support display rotation.

The ILI9486 display defaults to a portrait orientation (320x480), and rotation is required to use the display in landscape mode.

5.3 Seven-Segment Drivers

The HT1621 is driven with the `luma.lcd.device.ht1621` class, but is not accessed directly: it should be wrapped with the `luma.core.virtual.sevensegment` wrapper, as follows:

```
from luma.core.virtual import sevensegment
from luma.lcd.device import ht1621

device = ht1621()
seg = sevensegment(device)
```

The `seg` instance now has a `text` property which may be assigned, and when it does will update all digits according to the limited alphabet the 7-segment displays support. For example, assuming there are 2 cascaded modules, we have 16 character available, and so can write:

```
seg.text = "HELLO"
```

Rather than updating the whole display buffer, it is possible to update ‘slices’, as per the below example:

```
seg.text[0:5] = "BYE"
```

This replaces `HELLO` in the previous example, replacing it with `BYE`. The usual python idioms for slicing (inserting / replacing / deleting) can be used here, but note if inserted text exceeds the underlying buffer size, a `ValueError` is raised.

Floating point numbers (or text with ‘.’) are handled slightly differently - the decimal-place is fused in place on the character immediately preceding it. This means that it is technically possible to get more characters displayed than the buffer allows, but only because dots are folded into their host character.

5.4 Backlight Control

These displays typically require a backlight to illuminate the liquid crystal display. If the display’s backlight is connected to one of the single-board computer’s gpio pins, you can activate the backlight by specifying `gpio_LIGHT=n` where `n` = the pin number when initializing the device (default GPIO 18 (PWM_CLK0)).

If the display uses an I2C backpack with a pin from the backpack connected to the display’s backlight pin, you can activate the backlight by specifying `backpack_pin=n` where `n` = the pin number on the backpack.

The backlight can be programmatically switched on and off by calling `device.backlight(True)` or `device.backlight(False)` respectively.

Note: If you are using an I2C backpack based device, the backlight will not change until the next time you send a command or data to the device.

5.5 Examples

After installing the library, head over to the [luma.examples](#) repository. Details of how to run the examples is shown in the [README](#).

HD44780

6.1 Introduction

The HD44780 is sufficiently different from the other supported LCD displays to warrant a dedicated page to describe its features (and limitations).



It is a very popular LCD display that is widely available, cheap and available in several form factors with 16 character x 2 line and 20 character by 4 line displays the most common. As with most LCD displays, it comes with a backlight which `luma.lcd` can control using a GPIO pin. PWM modulation is supported if varying the intensity of the backlight is desired.

They are normally connected to SBCs using a parallel 6800-series interface implemented bitbang-style using the SBCs GPIO pins. Alternatively, they are also sold pre-connected to a few different I2C backpacks including the PCF8574, MCP23008 and MCP23017 I2C expanders.

6.2 Capabilities

The HD44780 is mainly a character-based device and can be purchased with different font tables installed in its ROM. The two most popular are the A00 ROM (English/Japanese) and the A02 ROM (English/European). See HD44780 for details on which characters are included in each.

In addition to the built-in characters, HD44780s can have up to eight custom written into their RAM. This feature is used by `luma.lcd` to provide a limited graphics capability.

6.2.1 Character-mode Usage

When using `luma.lcd.device.hd44780` the most common way of controlling the display is to use the `text` property which operates similarly to the `luma.core.virtual.sevensegment` wrapper class. To use it, assign the `text` property a string containing the values that you want displayed.

```
from luma.core.interface.parallel import bitbang_6800
from luma.lcd.device import hd44780

interface = bitbang_6800(RS=7, E=8, PINS=[25, 24, 23, 27])
device = hd44780(interface)

device.text = 'Hello World'
```

The newline character can be used to display characters on the next line of the display.

```
device.text = 'Hello\nWorld'
```

It is the only control character that works however. Other control characters such as carriage return (ascii 13) and line feed (ascii 10) will be silently ignored.

6.2.2 Graphics-mode Usage

It is also possible to display graphical content using the `display` method which works similarly to other `luma.lcd` and `luma.oled` displays. However there is a significant limitation. There can only be 8 characters worth of ‘custom’ content displayed at any point in time. If you exceed this limit, the `undefined` character (a value set during initialization of the `hd44780`) will be displayed instead. While 8 custom characters may seem limited, with some creativity it can be very useful.

To understand how to leverage this capability requires some explanation of how the screen of the display is organized and how `luma.lcd` manages the custom character space.

Each character position on an HD44780 display is made up of a 5 pixel by 8 pixel grid. When a character is requested to be displayed, the HD44780 looks up the character from its font table and copies the pixels from the font table ROM to the address in RAM that corresponds to the current character position. If there is not a character within the font table that matches what needs to be displayed, a custom character can be created within a small space in the display’s RAM that is reserved for this purpose. There are several restrictions that have to be kept in mind though.

- The custom character must be a 5x8 image
- It can only be displayed in alignment with the other characters on the display
- There can be a maximum of 8 special characters on the screen at any given time
- If the driver has run out of custom characters for a screen, the remaining cells with non-standard content will display the `undefined` character instead.

However there are a few features that can be leveraged to extend beyond these restrictions.

- The HD44780 class will automatically create the appropriate custom characters
- The content of a special character can be used multiple times on the screen
- The content of all of the special characters can be changed every time the screen is redrawn.

Here is a small example of how this can be leveraged.

```
from luma.core.interface.parallel import bitbang_6800
from luma.lcd.device import hd44780
from luma.core.render import canvas
from PIL import Image, ImageDraw

interface = bitbang_6800(RS=7, E=8, PINS=[25, 24, 23, 27])
device = hd44780(interface)

def progress_bar(width, height, percentage):
    img = Image.new('1', (width, height))
    drw = ImageDraw.Draw(img)
    drw.rectangle((0, 0, width-1, height-1), fill='black', outline='white')
    drw.rectangle((0, 0, width*percentage, height-1), fill='white', outline='white')
    return img

progress = 0.25
fnt = device.get_font('A00')
with canvas(device) as draw:
    draw.text((5,0), f'Installing {progress*100:.0f}%', font=fnt, fill='white')
    draw.bitmap((5,8), progress_bar(70, 8, progress), fill='white')
```



There are a few of things that deserve highlighting in this code.

- We have used the `hd44780` classes `get_font` method to retrieve the internal font used by the device. This enables us to place exact replicas of the characters within the font tables on the canvas. When these characters are displayed, because they are already normal characters, they do not consume any of the customer character RAM. The `hd44780` class contains both the A00 and A02 font tables. You should request the table that matches what is installed in your display.
- The progress bar is drawn using normal `PIL.ImageDraw` primitives in this case a couple of calls to the

`rectangle` method and a call to the `text` method.

- The size of the progress bar was carefully chosen. It is 70 pixels wide by 8 pixels high. This will fill 14 characters worth of space. This is because each cell is 5 pixels wide ($70/5=14$) and it is 8 pixels high ($8/8=1$). Normally 14 cells worth of graphical content would be a problem. However, the progress bar only requires four custom characters regardless of what position the progress value is set to. To see why that is you need to look at what each cell looks like within the progress bar at each state the progress bar.

The different conditions of the progress bar can be expressed in 5 states:

Table 1: CUSTOM CHARACTERS

First	Middle	End
Left Empty	Middle Empty	Right Empty
Filling	Middle Empty	Right Empty
Filled	Filling, Middle Empty	Right Empty
Filled	Filled, Filling, Middle Empty	Right Empty
Filled	Filled	Right Filling
Filled	Filled	Filled

So the maximum number of custom characters is as little as one but never exceeds four.

- The progress bar was carefully placed to align with the character cell boundaries. If we had placed the progress bar image one pixel to the left it would have consumed two additional custom characters as the beginning and end of the progress bar would be spread across character cell boundaries. In this particular case, it would still have displayed correctly because we would be under the 8 character limit.

Tip:

When displaying text, to avoid using custom character space you should:

- Use the internal font installed in your device which can be retrieved using `get_font`
- Make sure to align the placement of the text to the 5x8 cell structure of the display

Good:

```
draw.text( (5,0), 'Good', fill='white' font=device.get_font('A00'))
```

Bad:

```
draw.text( (6,1), 'Bad', fill='white' font=device.get_font('A00'))
```

6.2.3 Embedded Font Tables

The `luma.lcd.device.hd44780` class leverages `luma.core.bitmap_font` to include two fonts that replicate the two font tables that are commonly available for the hd44780. These are:

Font Table	Font Name
A00	English Japanese
A02	English European

You can retrieve either of these fonts using the `get_font` method.

```
fnt = device.get_font('A02')
```

You can also combine fonts together in order to display characters not included within your device's character table.

As an example, the 'Black Right-Pointing Triable' symbol Unicode U+25b6 is not contained in the A00 character table but is frequently used as a 'Play' symbol for multi-media systems. It is however included in the A02 font table. We can pull the symbol from A02 and add it to the current embedded font to enable us to use it.

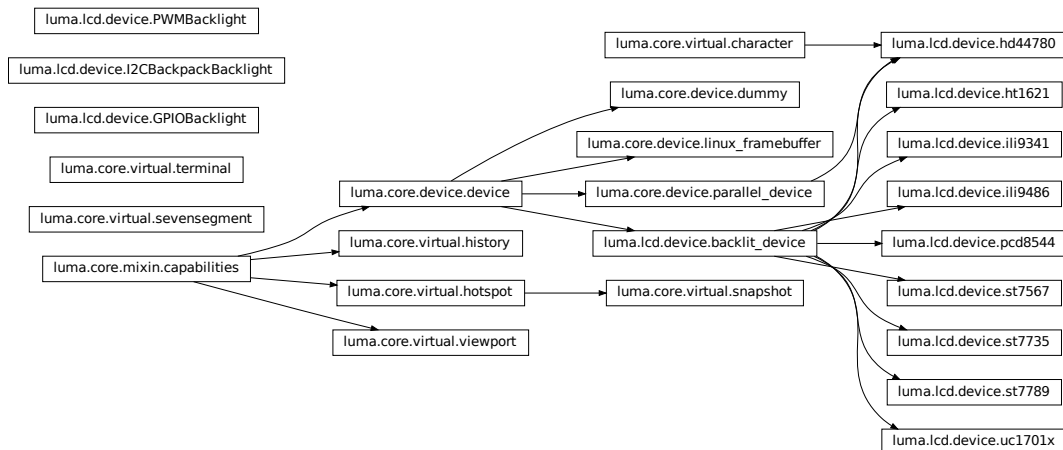
```
fnt = device.get_font('A02')
device.font.combine(fnt, '\u25b6')
device.text = '\u25b6 Play'
```

This feature leverages the custom character capability so it has the same 8 character limitation. If you exceed 8 characters within a screen, the undefined character will be used for all additional characters that are not contained within the devices font table.

See the documentation for `luma.core.bitmap_font` for more information on how to use the `bitmap_font` module.

API DOCUMENTATION

LCD display drivers.



7.1 Upgrading

Warning: Version 2.0.0 was released on 2 June 2019; this came with the removal of the `luma.lcd.aux.backlight` class. The equivalent functionality has now been subsumed into the device classes that have a backlight capability.

7.2 luma.lcd.device

Collection of serial interfaces to LCD devices.

```
class luma.lcd.device.hd44780 (serial_interface=None, width=16, height=2, undefined='_', selected_font=0, exec_time=1e-06, framebuffer=None, **kwargs)
Bases: luma.lcd.device.backlit_device, luma.core.device.parallel_device, luma.core.virtual.character, luma.lcd.device.__framebuffer_mixin
```

Driver for a HD44780 style LCD display. This class provides a `text` property which can be used to set and get a text value, which will be rendered to the display's screen using the display's built-in font.

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 line.
- **height** (*int*) – The number of lines the display supports.
- **undefined** (*str*) – character to use if a requested character is not in the font tables
- **selected_font** (*int or str*) – the font table appropriate for the model of display you are using. The hd44780 normally comes in a version with font A00 (ENGLISH_JAPANESE) or A02 (ENGLISH_EUROPEAN). You can provide either the name ('A00' or 'A02') or the number (0 for 'A00', 1 for 'A02') for the font your display contains.
- **exec_time** (*float*) – Time in seconds to wait for a command to complete. Default is 50 s (1e-6 * 50) which typically is long enough for commands to finish. If your display is not working correctly, you may want to try increasing the `exec_time` delay.
- **gpio_LIGHT** (*int*) – The GPIO pin to use for the backlight if it is controlled by one of the GPIO pins.
- **active_low** (*bool*) – Set to true if backlight is active low (default), false otherwise.
- **pwm_frequency** (*float*) – Use PWM for backlight brightness control with the specified frequency when provided.
- **framebuffer** (*luma.core.framebuffer.framebuffer*) – Framebuffering strategy, currently instances 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 = hd44780(p, selected_font='A00')
my_display.text = 'HD44780 Display\nFont A00 Eng/Jap'
```

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

..note: This driver currently only supports the hd44780 5x8 display mode.

New in version 2.5.0.

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

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 reset the device & switching it off prior to exiting the python process.

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 (*args)

Not support on this device. Ignore.

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 converts it to text data by reversing from glyphs from the image back to the correct value from the displays font table.

Parameters image (`PIL.Image.Image`) – the image to place on the display

If needed, it will create custom characters if a glyph is not found within the font table.

get_font (ft)

Return one of the devices built-in fonts as a `PIL.ImageFont` object

Parameters ft (*int or str*) – name or number of the font to return (0 - A00, 1 - A02)

hide ()

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

init_framebuffer (framebuffer, default_num_segments)

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.

property 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.lcd.device.ht1621` (*gpio=None, width=6, rotate=0, WR=11, DAT=10, CS=8, **kwargs*)

Bases: `luma.lcd.device.backlit_device`

Serial interface to a seven segment HT1621 monochrome LCD 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

- **gpio** – The GPIO library to use (usually RPi.GPIO) to delegate sending data and commands through.
- **width** (*int*) – The number of 7 segment characters laid out horizontally.
- **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.
- **WR** (*int*) – The write (SPI clock) pin to connect to, default BCM 11.
- **DAT** (*int*) – The data pin to connect to, default BCM 10.
- **CS** (*int*) – The chip select pin to connect to, default BCM 8.

New in version 0.4.0.

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

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 reset the device & switching it off prior to exiting the python process.

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 PCD8544 LCD 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.lcd.device.ili9341` (*serial_interface=None, width=320, height=240, rotate=0, framebuffer=None, h_offset=0, v_offset=0, bgr=False, **kwargs*)

Bases: `luma.lcd.device.backlit_device`, `luma.lcd.device.__framebuffer_mixin`

Serial interface to a 262k color (6-6-6 RGB) ILI9341 LCD 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 pixels laid out horizontally.
- **height** (*int*) – The number of pixels laid out vertically.
- **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** (*luma.core.framebuffer.framebuffer*) – 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** (*int*) – Vertical offset (in pixels) of screen to device memory (default: 0).

New in version 2.2.0.

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

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 reset the device & switching it off prior to exiting the python process.

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*)

NOT SUPPORTED

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 ILI9341 LCD display. The 8-bit RGB values are passed directly to the device's internal storage, but only the 6 most-significant bits are used by the 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*, *default_num_segments*)

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.lcd.device.ili9486` (*serial_interface=None*, *width=320*, *height=480*, *rotate=0*, *framebuffer=None*, *h_offset=0*, *v_offset=0*, *bgr=False*, ***kwargs*)

Bases: `luma.lcd.device.backlit_device`, `luma.lcd.device.__framebuffer_mixin`

Serial interface to a 262k color (6-6-6 RGB) ILI9486 LCD 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 (if implemented) and other settings.

Note that the ILI9486 display used for development – a Waveshare 3.5-inch IPS LCD(B) – used a portrait orientation. Images were rendered correctly only when specifying that height was 480 pixels and the width was 320.

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 pixels laid out horizontally.
- **height** (*int*) – The number of pixels laid out vertically.
- **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** (*luma.core.framebuffer.framebuffer*) – 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** (*int*) – Vertical offset (in pixels) of screen to device memory (default: 0).

New in version 2.8.0.

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

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 reset the device & switching it off prior to exiting the python process.

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*)

NOT SUPPORTED

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 ILI9486 LCD display. The 8-bit RGB values are passed directly to the devices internal storage, but only the 6 most-significant bits are used by the 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, default_num_segments*)

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.lcd.device.pcd8544` (*serial_interface=None, rotate=0, **kwargs*)

Bases: `luma.lcd.device.backlit_device`

Serial interface to a monochrome PCD8544 LCD 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.
- **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='I'*)

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 reset the device & switching it off prior to exiting the python process.

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 (*value*)

Sets the LCD contrast

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 PCD8544 LCD 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.lcd.device.st7567` (*serial_interface=None, rotate=0, **kwargs*)

Bases: `luma.lcd.device.backlit_device`

Serial interface to a monochrome ST7567 128x64 pixel LCD 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.
- **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 1.1.0.

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

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 reset the device & switching it off prior to exiting the python process.

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 (*value*)

Sets the LCD contrast

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 ST7567 LCD 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.lcd.device.st7735` (*serial_interface=None*, *width=160*, *height=128*, *rotate=0*, *framebuffer=None*, *h_offset=0*, *v_offset=0*, *bgr=False*, *inverse=False*, ***kwargs*)

Bases: `luma.lcd.device.backlit_device`, `luma.lcd.device.__framebuffer_mixin`

Serial interface to a 262K color (6-6-6 RGB) ST7735 LCD 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 pixels laid out horizontally.
- **height** (*int*) – The number of pixels laid out vertically.
- **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** (*luma.core.framebuffer.framebuffer*) – 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).
- **inverse** (*bool*) – Set to `True` if device pixels are inverted.
- **h_offset** (*int*) – Horizontal offset (in pixels) of screen to device memory (default: 0).
- **v_offset** (*int*) – Vertical offset (in pixels) of screen to device memory (default: 0).

New in version 0.3.0.

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

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 reset the device & switching it off prior to exiting the python process.

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*)

NOT SUPPORTED

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 ST7735 LCD display. The 8-bit RGB values are passed directly to the device's internal storage, but only the 6 most-significant bits are used by the 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, default_num_segments*)

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.lcd.device.st7789` (*serial_interface=None, rotate=0, **kwargs*)

Bases: `luma.lcd.device.backlit_device`

Serial interface to a colour ST7789 240x240 pixel LCD display.

New in version 2.9.0.

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

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 reset the device & switching it off prior to exiting the python process.

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*)

NOT SUPPORTED

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*

set_window (*x1, y1, x2, y2*)

show ()

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

class `luma.lcd.device.uc1701x` (*serial_interface=None, rotate=0, **kwargs*)

Bases: `luma.lcd.device.backlit_device`

Serial interface to a monochrome UC1701X LCD 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.
- **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 0.5.0.

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

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 reset the device & switching it off prior to exiting the python process.

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 (value)

Sets the LCD contrast

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 UC1701X LCD 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.

REFERENCES

- http://elinux.org/Rpi_Low-level_peripherals#General_Purpose_Input.2FOutput_.28GPIO.29
- <http://binerry.de/post/25787954149/pcd8544-library-for-raspberry-pi>
- <http://www.avdweb.nl/arduino/hardware-interfacing/nokia-5110-lcd.html>
- <http://www.raspberrypi.org/phpBB3/viewtopic.php?f=32&t=9814&start=100>
- <https://projects.drogon.net/raspberry-pi/wiringpi/pins/>
- http://www.henningkarlsen.com/electronics/t_imageconverter_mono.php
- <https://vimeo.com/41393421>
- <http://fritzing.org>
- <http://www.sitronix.com.tw/sitronix/product.nsf/Doc/ST7735?OpenDocument>
- <http://learn.adafruit.com/1-8-tft-display>
- <http://www.raspberrypi.org/phpBB3/viewtopic.php?t=28696&p=262909>
- http://elinux.org/images/1/19/Passing_Time_With_SPI_Framebuffer_Driver.pdf
- <http://www.flickr.com/photos/ngreatorex/7672743302/>
- <https://github.com/notro/fbtf>
- <https://github.com/rm-hull/st7735fb>
- <http://www.areinhardt.de/news/raspberry-pi-tft-display/>
- <http://www.whence.com/rpi/>
- <http://harizanov.com/product/1-8-tft-display-for-raspberry-pi/>
- <https://www.circuits.dk/everything-about-raspberry-gpio/>

CONTRIBUTING

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

9.1 GitHub

The source code is available to clone at: <https://github.com/rm-hull/luma.lcd.git>

9.2 Contributors

- Thijs Triemstra (@thijstriemstra)
- Dougie Lawson (@dougiewatson)
- WsMithril (@WsMithril)
- Peter Martin (@pe7er)
- Saumyakanta Sahoo (@somu1795)
- Philip Howard (@Gadgetoid)
- Ricardo Amendoeira (@ric2b)
- Kevin Stone (@kevinastone)
- Drone (@dhrone)
- Matthew Lovell (@mattblovell)
- Maciej Sokolowski (@matemaciek)

CHANGELOG

Version	Description	Date
2.9.0	<ul style="list-style-type: none">• Add ST7789 Colour LCD display driver	2021/03/14
2.8.0	<ul style="list-style-type: none">• Add ILI9486 Colour LCD display driver	2020/12/14
2.7.1	<ul style="list-style-type: none">• Fix mutable default parameter bug when using multiple displays	2020/11/15
2.7.0	<ul style="list-style-type: none">• Improved performance for ST7739 and ILI9341 displays	2020/11/04
2.6.0	<ul style="list-style-type: none">• Drop support for Python 3.5, only 3.6 or newer is supported now• Pin luma.core to 1.x.y line only, in anticipation of performance improvements in upcoming major release	2020/10/25
2.5.0	<ul style="list-style-type: none">• Add HD44780 LCD display driver	2020/09/24
2.4.0	<ul style="list-style-type: none">• Drop support for Python 2.7, only 3.5 or newer is supported now	2020/07/04
2.3.0	<ul style="list-style-type: none">• Add PWM backlight control	2020/01/08

continues on next page

Table 1 – continued from previous page

Version	Description	Date
2.2.0	<ul style="list-style-type: none"> • Add ILI9341 Colour LCD display driver 	2019/11/25
2.1.0	<ul style="list-style-type: none"> • Rework namespace handling for luma sub-projects 	2019/06/16
2.0.0	<ul style="list-style-type: none"> • BREAKING CHANGES: Removal of <code>luma.lcd.aux.backlight</code> class • Device classes now incorporate backlight capability 	2019/06/02
1.1.1	<ul style="list-style-type: none"> • Add support for 160x80 display size for ST7735 • Minor documentation updates 	2019/03/30
1.1.0	<ul style="list-style-type: none"> • Add ST7567 Monochrome LCD display driver (courtesy of @Gadgetoid) • Change HT1621 tests • Update dependencies 	2018/09/07
1.0.3	<ul style="list-style-type: none"> • Changed version number to inside <code>luma/lcd/__init__.py</code> 	2017/11/23
1.0.2	<ul style="list-style-type: none"> • Documentation and dependencies updates 	2017/10/30
1.0.1	<ul style="list-style-type: none"> • Update dependencies 	2017/09/14
1.0.0	<ul style="list-style-type: none"> • Stable version • Remove deprecated methods 	2017/09/09
0.5.0	<ul style="list-style-type: none"> • Add UC1701X Monochrome LCD display driver 	2017/06/11
0.4.1	<ul style="list-style-type: none"> • <code>luma.core</code> 0.9.0 or newer is required now 	2017/04/22

continues on next page

Table 1 – continued from previous page

Version	Description	Date
0.4.0	<ul style="list-style-type: none"> • Add HT1621 seven-segment driver 	2017/04/22
0.3.3	<ul style="list-style-type: none"> • Add deprecation warning for <code>bcm_LIGHT</code> 	2017/03/14
0.3.4	<ul style="list-style-type: none"> • Add support for 128x128 display size for ST7735 • Implement horizontal and vertical offsets (for ST7735) • Make backlight configurable as active high or active low 	2017/04/17
0.3.3	<ul style="list-style-type: none"> • Add deprecation warning for <code>bcm_LIGHT</code> 	2017/03/14
0.3.2	<ul style="list-style-type: none"> • Raise <code>error.UnsupportedPlatform</code> if <code>RPi.GPIO</code> is not available 	2017/03/08
0.3.0	<ul style="list-style-type: none"> • Add ST7735 Color TFT LCD display driver • Removed width and height parameters from device constructors • BREAKING CHANGES: Move backlight class to different package 	2017/03/05
0.2.3	<ul style="list-style-type: none"> • Allow PCD8544 driver constructor to accept any args 	2017/03/02
0.2.2	<ul style="list-style-type: none"> • Restrict exported Python symbols from <code>luma.lcd.device</code> 	2017/03/02
0.2.1	<ul style="list-style-type: none"> • Bugfix: Backlight didn't switch off properly • Add tests 	2017/01/23

continues on next page

Table 1 – continued from previous page

Version	Description	Date
0.2.0	<ul style="list-style-type: none">• BREAKING CHANGES: Package rename to luma. lcd	2017/01/13
0.0.1	<ul style="list-style-type: none">• Bit-bang version using wiringPi	2013/01/28

THE MIT LICENSE (MIT)

Copyright (c) 2013-2021 Richard Hull & Contributors

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

PYTHON MODULE INDEX

I

`luma.lcd`, [35](#)

`luma.lcd.device`, [35](#)

C

capabilities() (*luma.lcd.device.hd44780 method*), 36
 capabilities() (*luma.lcd.device.ht1621 method*), 38
 capabilities() (*luma.lcd.device.ili9341 method*), 39
 capabilities() (*luma.lcd.device.ili9486 method*), 41
 capabilities() (*luma.lcd.device.pcd8544 method*), 42
 capabilities() (*luma.lcd.device.st7567 method*), 43
 capabilities() (*luma.lcd.device.st7735 method*), 44
 capabilities() (*luma.lcd.device.st7789 method*), 45
 capabilities() (*luma.lcd.device.uc1701x method*), 46
 cleanup() (*luma.lcd.device.hd44780 method*), 36
 cleanup() (*luma.lcd.device.ht1621 method*), 38
 cleanup() (*luma.lcd.device.ili9341 method*), 39
 cleanup() (*luma.lcd.device.ili9486 method*), 41
 cleanup() (*luma.lcd.device.pcd8544 method*), 42
 cleanup() (*luma.lcd.device.st7567 method*), 43
 cleanup() (*luma.lcd.device.st7735 method*), 44
 cleanup() (*luma.lcd.device.st7789 method*), 45
 cleanup() (*luma.lcd.device.uc1701x method*), 46
 clear() (*luma.lcd.device.hd44780 method*), 36
 clear() (*luma.lcd.device.ht1621 method*), 38
 clear() (*luma.lcd.device.ili9341 method*), 39
 clear() (*luma.lcd.device.ili9486 method*), 41
 clear() (*luma.lcd.device.pcd8544 method*), 42
 clear() (*luma.lcd.device.st7567 method*), 43
 clear() (*luma.lcd.device.st7735 method*), 44
 clear() (*luma.lcd.device.st7789 method*), 45
 clear() (*luma.lcd.device.uc1701x method*), 46
 command() (*luma.lcd.device.hd44780 method*), 37
 command() (*luma.lcd.device.ht1621 method*), 38
 command() (*luma.lcd.device.ili9341 method*), 40
 command() (*luma.lcd.device.ili9486 method*), 41
 command() (*luma.lcd.device.pcd8544 method*), 42

command() (*luma.lcd.device.st7567 method*), 43
 command() (*luma.lcd.device.st7735 method*), 44
 command() (*luma.lcd.device.st7789 method*), 45
 command() (*luma.lcd.device.uc1701x method*), 47
 contrast() (*luma.lcd.device.hd44780 method*), 37
 contrast() (*luma.lcd.device.ht1621 method*), 38
 contrast() (*luma.lcd.device.ili9341 method*), 40
 contrast() (*luma.lcd.device.ili9486 method*), 41
 contrast() (*luma.lcd.device.pcd8544 method*), 42
 contrast() (*luma.lcd.device.st7567 method*), 43
 contrast() (*luma.lcd.device.st7735 method*), 44
 contrast() (*luma.lcd.device.st7789 method*), 45
 contrast() (*luma.lcd.device.uc1701x method*), 47

D

data() (*luma.lcd.device.hd44780 method*), 37
 data() (*luma.lcd.device.ht1621 method*), 38
 data() (*luma.lcd.device.ili9341 method*), 40
 data() (*luma.lcd.device.ili9486 method*), 41
 data() (*luma.lcd.device.pcd8544 method*), 42
 data() (*luma.lcd.device.st7567 method*), 43
 data() (*luma.lcd.device.st7735 method*), 45
 data() (*luma.lcd.device.st7789 method*), 45
 data() (*luma.lcd.device.uc1701x method*), 47
 display() (*luma.lcd.device.hd44780 method*), 37
 display() (*luma.lcd.device.ht1621 method*), 38
 display() (*luma.lcd.device.ili9341 method*), 40
 display() (*luma.lcd.device.ili9486 method*), 41
 display() (*luma.lcd.device.pcd8544 method*), 42
 display() (*luma.lcd.device.st7567 method*), 43
 display() (*luma.lcd.device.st7735 method*), 45
 display() (*luma.lcd.device.st7789 method*), 46
 display() (*luma.lcd.device.uc1701x method*), 47

G

get_font() (*luma.lcd.device.hd44780 method*), 37

H

hd44780 (*class in luma.lcd.device*), 35
 hide() (*luma.lcd.device.hd44780 method*), 37
 hide() (*luma.lcd.device.ht1621 method*), 38
 hide() (*luma.lcd.device.ili9341 method*), 40

`hide()` (*luma.lcd.device.ili9486 method*), 41
`hide()` (*luma.lcd.device.pcd8544 method*), 42
`hide()` (*luma.lcd.device.st7567 method*), 43
`hide()` (*luma.lcd.device.st7735 method*), 45
`hide()` (*luma.lcd.device.st7789 method*), 46
`hide()` (*luma.lcd.device.uc1701x method*), 47
`ht1621` (*class in luma.lcd.device*), 37

I

`ili9341` (*class in luma.lcd.device*), 39
`ili9486` (*class in luma.lcd.device*), 40
`init_framebuffer()` (*luma.lcd.device.hd44780 method*), 37
`init_framebuffer()` (*luma.lcd.device.ili9341 method*), 40
`init_framebuffer()` (*luma.lcd.device.ili9486 method*), 41
`init_framebuffer()` (*luma.lcd.device.st7735 method*), 45

L

`luma.lcd`
 module, 35
`luma.lcd.device`
 module, 35

M

module
 luma.lcd, 35
 luma.lcd.device, 35

P

`pcd8544` (*class in luma.lcd.device*), 41
`preprocess()` (*luma.lcd.device.hd44780 method*), 37
`preprocess()` (*luma.lcd.device.ht1621 method*), 38
`preprocess()` (*luma.lcd.device.ili9341 method*), 40
`preprocess()` (*luma.lcd.device.ili9486 method*), 41
`preprocess()` (*luma.lcd.device.pcd8544 method*), 42
`preprocess()` (*luma.lcd.device.st7567 method*), 43
`preprocess()` (*luma.lcd.device.st7735 method*), 45
`preprocess()` (*luma.lcd.device.st7789 method*), 46
`preprocess()` (*luma.lcd.device.uc1701x method*), 47

S

`set_window()` (*luma.lcd.device.st7789 method*), 46
`show()` (*luma.lcd.device.hd44780 method*), 37
`show()` (*luma.lcd.device.ht1621 method*), 39
`show()` (*luma.lcd.device.ili9341 method*), 40
`show()` (*luma.lcd.device.ili9486 method*), 41
`show()` (*luma.lcd.device.pcd8544 method*), 42
`show()` (*luma.lcd.device.st7567 method*), 43
`show()` (*luma.lcd.device.st7735 method*), 45
`show()` (*luma.lcd.device.st7789 method*), 46
`show()` (*luma.lcd.device.uc1701x method*), 47

`st7567` (*class in luma.lcd.device*), 42
`st7735` (*class in luma.lcd.device*), 44
`st7789` (*class in luma.lcd.device*), 45

T

`text()` (*luma.lcd.device.hd44780 property*), 37

U

`uc1701x` (*class in luma.lcd.device*), 46