python-picaso-lcd Documentation

Release 0.1.0-dev

Christian Fässler, Danilo Bargen

Sep 27, 2017

Contents

This is a Python library for PICASO based LCDs by 4D Systems connected via serial port.

CHAPTER 1

Usage

Connect the LCD to your computer using a serial connection (e.g. USB). Then create a new `picaso_lcd.Display` instance, with the serial port as argument:

```
import picaso_lcd
disp = picaso_lcd.Display('/dev/ttyUSB0')
```

(If you're using Windows, the port string would probably be something like `COM3`.)

The different functionality groups are provided in their own namespaces. For example the text related functions are grouped in `disp.text`, while the graphics related functions are grouped in `disp.gfx`. General purpose methods are not in a sub-namespace and can be accessed directly.

Example:

```
import picaso_lcd
disp = picaso_lcd.Display('/dev/ttyUSB0')

disp.cls()
disp.text.put_string('Welcome\nThis is a nice library.')
disp.text.move_cursor(0, 7)
disp.text.put_character('!')
```

This will result in the following text being written on the display:

```
Welcome!
This is a nice library.
```

For more information, please refer to the API Docs.

CHAPTER 2

API Documentation

picaso_lcd.display

This module contains all necessary code to control the LCD display. All the classes named `DisplayFOO` can also be accessed via a *`Display`* instance at `display.foo`.

class `picaso_lcd.display.`**`Display`**(*port*, *baudrate=9600*, *read_timeout=10*, *write_timeout=10*)

This class represents a 4D Systems serial LCD. It's the main class of this project.

Parameters

- **`port`** (*`str or unicode`*) – serial port to which the display is connected
- **`baudrate`** (*`int`*) – default 9600 in SPE2 rev 1.1
- **`read_timeout`** (*`int or None`*) – Serial read timeout. This may be `None` (blocking), `0` (non-blocking) or an integer > 0 (seconds).
- **`write_timeout`** (*`int or None`*) – Serial write timeout. This may be `None` (blocking), `0` (non-blocking) or an integer > 0 (seconds).

Return type Display instance

`gfx_polyline`(*lines*, *color*, *closed=False*, *filled=False*)

A polyline could be closed or filled, where filled is always closed.

`set_contrast`(*contrast*)

Set the contrast. Note that this has no effect on most LCDs.

`set_orientation`(*value*)

Set display orientation 0 = Landscape 1 = Landscape reverse 2 = portrait 3 = portrait reverse

Returns previous orientation

`write_cmd`(*cmd*, *return_bytes=0*)

Write list of words to the serial port.

Values are always converted into a word (16bit value, consisting of two bytes: high byte, low byte) even if they would fit into a single byte.

The communication protocol is based on exchanging words. Only a few special commands use single byte values, in this case use write_raw_cmd instead.

Parameters

- **cmd** (*list of int*) – The list of command words (16 bit) to send.
- **return_bytes** (*int*) – Number of return bytes. Default 0.

Returns List of response bytes if there are any, else None.

Return type list or none

write_raw_cmd(*cmd*, *return_bytes=0*)
Write list of bytes directly to the serial port.

Parameters

- **cmd** (*list of int*) – List containing numeric bytes.
- **return_bytes** (*int*) – Number of return bytes. Default 0.

Returns List of response bytes if there are any, else None.

Return type list or none

class picaso_lcd.display.**DisplayText**(*display*)
Text/String related functions. Can be accessed directly from a *Display* instance using `display.text.<method>`.

Parameters **display** (Display) – The display instance.

get_character_height(*character*)
Get the height of a character in pixels.

The *Character Height* command is used to calculate the height in pixel units for a character, based on the currently selected font. The font can be proportional or mono-spaced. If the total height of the character exceeds 255 pixel units, the function will return the 'wrapped' (modulo 8) value.

TODO: Handle and hide this strange modulo stuff!

Parameters **character** (*str*) – The ASCII character for which to calculate the height.

Returns The height of the character. If the total height of the character exceeds 255 pixel units, the function will return the 'wrapped' (modulo 8) value.

Return type int

get_character_width(*character*)
Get the width of a character in pixels.

The *Character Width* command is used to calculate the width in pixel units for a character, based on the currently selected font. The font can be proportional or mono-spaced. If the total width of the character exceeds 255 pixel units, the function will return the 'wrapped' (modulo 8) value.

TODO: Handle and hide this strange modulo stuff!

Parameters **character** (*str*) – The ASCII character for which to calculate the width.

Returns The width of the character. If the total width of the character exceeds 255 pixel units, the function will return the 'wrapped' (modulo 8) value.

Return type int

move_cursor(*line*, *column*)
Move cursor to specified position.

This command moves the text cursor to a screen position set by line and column parameters. The line and column position is calculated, based on the size and scaling factor for the currently selected font. When text is outputted to screen it will be displayed from this position. The text position could also be set with *Move Origin* command if required to set the text position to an exact pixel location. Note that lines and columns start from 0, so line 0, column 0 is the top left corner of the display.

Parameters

- **line** (*int*) – Line number (0..n)
- **column** (*int*) – Column number (0..n)

Returns None

put_character(*char*)
Write a single character to the display.

Parameters **char** (*str*) – The character to print. Must be a printable ASCII character.

Returns None

put_string(*string*)
Write a string to the display. Maximum string length is 511 chars.

Parameters **string** (*str*) – The string to print. Must consist of printable ASCII characters.

set_attributes(*bold=False*, *italic=False*, *inverse=False*, *underlined=False*)
Control text attributes like bold, italic, underlined etc in a single command.

The Text Attributes command controls the following functions:

•Text Bold

•Text Italic

•Text Inverse

•Text Underlined

Note: The *set_y_gap()* command is required to be at least 2 for the underline (Text Underlined attribute) to be visible. Please refer to the *set_y_gap()* command for further information.

Parameters

- **bold** (*bool*) – Text bold attribute.
- **italic** (*bool*) – Text italic attribute.
- **inverse** (*bool*) – Text inverse attribute.
- **underlined** (*bool*) – Text underlined attribute.

Returns Dictionary of previous attribute values. Note that this does not consider values set using the dedicated `set_bold`, `set_italic` etc functions.

Return type dict of bool

set_bg_color(*color*)
Set the text background color.

The *Text Background Color* command sets the text background color, and reports back the previous background color.

Parameters **color** (*int*) – The color to be set as background color.

Returns The previous background color.

Return type int

set_bold(*mode*)
Enable or disable bold mode.

The *Text Bold* command sets the Bold attribute for the text and report back the previous bold status.

Parameters **mode** (*int*) – 1 for ON, 0 for OFF.

Returns Previous mode.

Return type int

set_fg_color(*color*)
Set the text foreground color.

The *Text Foreground Color* command sets the text foreground color, and reports back the previous foreground color.

Parameters **color** (*int*) – The color to be set as foreground color.

Returns The previous foreground color.

Return type int

set_font(*font*)
Set the used font.

The *Set Font* command sets the required font using its ID, and report back the previous font ID used.

Parameters **font** (*int*) – The font ID to use. 0 - Font1 (System Font) 1 - Font2 3 - Font3 (Default Font)

Returns The previous font ID used.

Return type int

set_gap(*pixelcount*)
Set both the x- and the y-gap between characters.

This is a shortcut function that calls both *set_x_gap()* and *set_y_gap()*. The return value is a tuple containing the previous pixelcount values.

Parameters **pixelcount** (*int*) – Gap size in pixels, 0 to 32 (default 0).

Returns Tuple `(previous_x_gap, previous_y_gap)`

Return type tuple(int, int)

set_height(*multiplier*)
Set the text height.

The *Text Height* command sets the text height multiplier between 1 and 16, and returns the previous multiplier.

Parameters **multiplier** (*int*) – Height multiplier, 1 to 16 (default 1).

Returns Previous multiplier.

Return type int

set_inverse(*mode*)
Enable or disable inverse mode.

The *Text Inverse* command sets the inverse attribute for the text and report back the previous inverse status.

Parameters **mode** (*int*) – 1 for ON, 0 for OFF.

Returns Previous mode.

Return type int

set_italic(*mode*)
Enable or disable italic mode.

The *Text Italic* command sets the italic attribute for the text and report back the previous italic status.

Parameters **mode** (*int*) – 1 for ON, 0 for OFF.

Returns Previous mode.

Return type int

set_opacity(*mode*)
Enable or disable opacity.

The *Text Opacity* command selects whether or not the 'background' pixels are drawn, and returns the previous text opacity status. (Default mode is OPAQUE with BLACK background.)

Parameters **mode** (*int*) – 1 for ON (opaque), 0 for OFF (transparent).

Returns Previous mode.

Return type int

set_size(*multiplier*)
Set the text size.

This is a shortcut functions that calls both *set_width()* and *set_height()*. The return value is a tuple containing previous width- and height- multipliers.

Parameters **multiplier** (*int*) – Size multiplier, 1 to 16 (default 1).

Returns Tuple `(previous_width, previous_height)`

Return type tuple(int, int)

set_underline(*mode*)
Enable or disable text underline.

The *Text Underline* command sets the text to underlined, and returns the previous text underline status.

Note: The *set_y_gap()* command is required to be at least 2 for the underline to be visible. Please refer to the *set_y_gap()* command for further information.

Parameters **mode** (*int*) – 1 for ON, 0 for OFF.

Returns Previous mode.

Return type int

set_width(*multiplier*)
Set the text width.

The *Text Width* command sets the text width multiplier between 1 and 16, and returns the previous multiplier.

Parameters **multiplier** (*int*) – Width multiplier, 1 to 16 (Default 1).

Returns Previous multiplier.

Return type int

set_x_gap(*pixelcount*)
Set the horizontal gap between characters.

The *Text X-gap* command sets the pixel gap between characters (x-axis), where the gap is in pixel units, and the response is the previous pixelcount value.

Parameters **pixelcount** (*int*) – Gap size in pixels, 0 to 32 (default 0).

Returns Previous pixelcount value.

Return type int

set_y_gap(*pixelcount*)
Set the vertical gap between characters.

The *Text Y-gap* command sets the pixel gap between characters (y-axis), where the gap is in pixel units, and the response is the previous pixelcount value.

This command is required to be used if setting text to have an Underline using the *set_underline()* command, or *set_attributes()* command with the suitable bits set. See these command for further information.

Parameters **pixelcount** (*int*) – Gap size in pixels, 0 to 32 (default 0).

Returns Previous pixelcount value.

Return type int

class picaso_lcd.display.**DisplayTouch**(*display*)
Touchscreen related functions. Can be accessed directly from a *Display* instance using `display.touch.<method>`.

Parameters **display** (Display) – The display instance.

get_status(*mode*)
Poll the touch screen.

Returns various Touch Screen parameters to caller, based on the touch detect region on the screen set by the *set_detect_region()* command.

Request modes

- mode = 0: Get status
- mode = 1: Get X coordinates
- mode = 2: Get Y coordinates

Response values

- **mode = 0: The various states of the touch screen. Possible values:** 0 = INVALID / NOTOUCH, 1 = PRESS, 2 = RELEASE, 3 = MOVING
- mode = 1: The X coordinates of the touch
- mode = 2: The Y coordinates of the touch

Parameters **mode** (*int*) – The status mode (0, 1 or 2). See method docstring for more information.

Returns A value dependent on the request mode.

Return type int

set_detect_region(*x1*, *y1*, *x2*, *y2*)
Set the touch detect region.

Specifies a new touch detect region on the screen. This setting will filter out any touch activity outside the region and only touch activity within that region will be reported by the status poll *get_status()* command.

Parameters

- **x1** – X coordinate of top left corner of the region
- **y1** – Y coordinate of top left corner of the region
- **x1** – X coordinate of bottom right corner of the region
- **y1** – Y coordinate of bottom right corner of the region

set_mode(*mode*)

Set touch screen related parameters.

mode = 0: Enables and initialises Touch Screen hardware. mode = 1: Disables the Touch Screen. mode = 2: This will reset the current active region to default which is the full screen area.

Note: Touch Screen task runs in the background and disabling it when not in use will free up extra resources for 4DGL CPU cycles.

Parameters **mode** (*int*) – The touch mode (0, 1 or 2). See method docstring for more information.

picaso_lcd.utils

This module contains different utility functions, e.g. for handling of binary data or colors.

picaso_lcd.utils.**dbyte_to_int**(*high_byte*, *low_byte*)

Convert a double byte `(high byte, low byte)` to a single integer.

If you're already dealing with a 2-tuple, you can simply convert it to an integer by using arg unpacking:

```
>>> dbyte = (0x1, 0x2)
>>> dbyte_to_int(*dbyte)
```

Parameters

- **high_byte** (*int*) – The high byte.
- **low_byte** (*int*) – The low byte.

Returns Single integer.

Return type int

picaso_lcd.utils.**int_to_dbyte**(*value*)

Convert a single integer to a double byte: `(high byte, low byte)`.

The value (which must be < 2**16) is split up into a two-byte structure: (high byte, low byte).

Parameters **value** (*int < 2**16*) – The value to be converted.

Returns 2-Tuple with high byte and low byte.

Raises ValueError

Return type (int, int)

picaso_lcd.utils.**to_16bit_color**(*red*, *green*, *blue*)

Convert rgb color to 16 bit color.

Color scheme is 16bit (565). Greater values are truncated.

Parameters

- **red** (*int*) – The red value. Should be smaller than 32, larger values are truncated.

- **green** (*int*) – The green value. Should be smaller than 64, larger values are truncated.
- **blue** (*int*) – The blue value. Should be smaller than 32, larger values are truncated.

Returns 16 bit color value.

Return type int

picaso_lcd.exceptions

Custom exceptions used by the library.

exception picaso_lcd.exceptions.**CommunicationError**
Communication with device failed (e.g. a serial read / write timeout).

exception picaso_lcd.exceptions.**PicasoError**
Something went wrong while processing the command.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index