python-escpos Documentation

Release 3.0a5

Manuel F Martinez and others

Jun 19, 2019

User Documentation

1 Description 3

2 Dependencies 5

3 Documentation and Usage 7

4 Contributing 9

5 Disclaimer 11

6 Content 13

Python Module Index 63

Index 65

CHAPTER 1

Description

Python ESC/POS is a library which lets the user have access to all those printers handled by ESC/POS commands, as defined by Epson, from a Python application.

The library tries to implement the functions provided by the ESC/POS-commandset and supports sending text, images, barcodes and qr-codes to the printer.

Text can be aligned/justified and fonts can be changed by size, type and weight.

Also, this module handles some hardware functionalities like cutting paper, control characters, printer reset and similar functions.

Since supported commands differ from printer to printer the software tries to automatically apply the right settings for the printer that you set. These settings are handled by escpos-printer-db which is also used in escpos-php.

CHAPTER 2

Dependencies

This library makes use of:

- pyusb for USB-printers
- Pillow for image printing
- qrcode for the generation of QR-codes
- pyserial for serial printers
- viivakoodi for the generation of barcodes

CHAPTER 3

Documentation and Usage

The basic usage is:

```
from escpos.printer import Usb

""" Seiko Epson Corp. Receipt Printer (EPSON TM-T88III) """
p = Usb(0x04b8, 0x0202, 0, profile="TM-T88III")
p.text("Hello World\n")
p.image("logo.gif")
p.barcode('1324354657687', 'EAN13', 64, 2, '', '')
p.cut()
```

Another example based on the Network printer class:

```
from escpos.printer import Network

kitchen = Network("192.168.1.100") #Printer IP Address
kitchen.text("Hello World\n")
kitchen.barcode('1324354657687', 'EAN13', 64, 2, '', '')
kitchen.cut()
```

The full project-documentation is available on Read the Docs.

CHAPTER 4

Contributing

This project is open for any contribution! Please see CONTRIBUTING.rst for more information.

CHAPTER 5

Disclaimer

None of the vendors cited in this project agree or endorse any of the patterns or implementations. Its names are used only to maintain context.

CHAPTER 6

Content

6.1 Installation

Last Reviewed 2016-07-23

6.1.1 Installation with PIP

Installation should be rather straight-forward. python-escpos is on PyPi, so you can simply enter:

```
pip install python-escpos
```

This should install all necessary dependencies. Apart from that python-escpos should also be available as a Debian package. If you want to always benefit from the newest stable releases you should probably install from PyPi.

6.1.2 Setup udev for USB-Printers

1. Get the *Product ID* and *Vendor ID* from the lsusb command `# lsusb Bus 002 Device 001: ID 1a2b:1a2b Device name`
2. Create a udev rule to let users belonging to *dialout* group use the printer. You can create the file `/etc/udev/rules.d/99-escpos.rules` and add the following: `SUBSYSTEM=="usb", ATTRS{idVendor}=="1a2b", ATTRS{idProduct}=="1a2b", MODE="0664", GROUP="dialout"` Replace *idVendor* and *idProduct* hex numbers with the ones that you got from the previous step. Note that you can either, add yourself to "dialout" group, or use another group you already belongs instead "dialout" and set it in the `GROUP` parameter in the above rule.
3. Restart udev `# sudo service udev restart` In some new systems it is done with `# sudo udevadm control --reload`

6.1.3 Enabling tab-completion in CLI

python-escpos has a CLI with tab-completion. This is realised with `argcomplete`. In order for this to work you have to enable tab-completion, which is described in the manual of argcomplete.

If you only want to enable it for python-escpos, or global activation does not work, try this:

```
eval "$(register-python-argcomplete python-escpos)"
```

6.2 Methods

Last Reviewed 2017-01-25

6.2.1 Escpos class

The core part of this libraries API is the Escpos class. You use it by instantiating a *printer* which is a child of Escpos. The following methods are available:

class escpos.escpos.**Escpos**(*profile=None*, *magic_encode_args=None*, ***kwargs*)
ESC/POS Printer object

This class is the abstract base class for an esc/pos-printer. The printer implementations are children of this class.

image(*img_source*, *high_density_vertical=True*, *high_density_horizontal=True*, *impl=u'bitImageRaster'*, *fragment_height=960*, *center=False*)
Print an image

You can select whether the printer should print in high density or not. The default value is high density. When printing in low density, the image will be stretched.

Esc/Pos supplies several commands for printing. This function supports three of them. Please try to vary the implementations if you have any problems. For example the printer *IT80-002* will have trouble aligning images that are not printed in Column-mode.

The available printing implementations are:

- *bitImageRaster*: prints with the *GS v 0*-command
- *graphics*: prints with the *GS (L*-command
- *bitImageColumn*: prints with the *ESC* *-command

Parameters

- **img_source** – PIL image or filename to load: *jpg*, *gif*, *png* or *bmp*
- **high_density_vertical** – print in high density in vertical direction *default:* True
- **high_density_horizontal** – print in high density in horizontal direction *default:* True
- **impl** – choose image printing mode between *bitImageRaster*, *graphics* or *bitImageColumn*
- **fragment_height** – Images larger than this will be split into multiple fragments *default:* 960
- **center** – Center image horizontally *default:* False

qr(*content*, *ec=0*, *size=3*, *model=2*, *native=False*, *center=False*, *impl=u'bitImageRaster'*)
Print QR Code for the provided string

Parameters

- **content** – The content of the code. Numeric data will be more efficiently compacted.
- **ec** – Error-correction level to use. One of QR_ECLEVEL_L (default), QR_ECLEVEL_M, QR_ECLEVEL_Q or QR_ECLEVEL_H. Higher error correction results in a less compact code.
- **size** – Pixel size to use. Must be 1-16 (default 3)
- **model** – QR code model to use. Must be one of QR_MODEL_1, QR_MODEL_2 (default) or QR_MICRO (not supported by all printers).
- **native** – True to render the code on the printer, False to render the code as an image and send it to the printer (Default)
- **center** – Centers the code *default:* False

charcode(*code=u'AUTO'*)
Set Character Code Table

Sets the control sequence from `CHARCODE` in *escpos.constants* as active. It will be sent with the next text sequence. If you set the variable code to `AUTO` it will try to automatically guess the right codepage. (This is the standard behaviour.)

Parameters **code** – Name of CharCode

Raises *CharCodeError*

static check_barcode(*bc*, *code*)
This method checks if the barcode is in the proper format. The validation concerns the barcode length and the set of characters, but won't compute/validate any checksum. The full set of requirement for each barcode type is available in the ESC/POS documentation.

As an example, using EAN13, the barcode *12345678901* will be correct, because it can be rendered by the printer. But it does not suit the EAN13 standard, because the checksum digit is missing. Adding a wrong checksum in the end will also be considered correct, but adding a letter won't (EAN13 is numeric only).

Todo: Add a method to compute the checksum for the different standards

---

Todo: For fixed-length standards with mandatory checksum (EAN, UPC), compute and add the checksum automatically if missing.

Parameters

- **bc** – barcode format, see :py:func‘~escpos.Escpos.barcode‘
- **code** – alphanumeric data to be printed as bar code, see :py:func‘~escpos.Escpos.barcode‘

Returns bool

barcode(*code*, *bc*, *height=64*, *width=3*, *pos=u'BELOW'*, *font=u'A'*, *align_ct=True*, *function_type=None*, *check=True*)
Print Barcode

This method allows to print barcodes. The rendering of the barcode is done by the printer and therefore has to be supported by the unit. By default, this method will check whether your barcode text is correct, that is the characters and lengths are supported by ESCPOS. Call the method with *check=False* to disable the check, but note that uncorrect barcodes may lead to unexpected printer behaviour. There are two forms of the barcode function. Type A is default but has fewer barcodes, while type B has some more to choose from.

Use the parameters *height* and *width* for adjusting of the barcode size. Please take notice that the barcode will not be printed if it is outside of the printable area. (Which should be impossible with this method, so this information is probably more useful for debugging purposes.)

Todo: On TM-T88II width from 1 to 6 is accepted. Try to acquire command reference and correct the code.

Todo: Supplying pos does not have an effect for every barcode type. Check and document for which types this is true.

If you do not want to center the barcode you can call the method with *align_ct=False*, which will disable automatic centering. Please note that when you use center alignment, then the alignment of text will be changed automatically to centered. You have to manually restore the alignment if necessary.

Todo: If further barcode-types are needed they could be rendered transparently as an image. (This could also be of help if the printer does not support types that others do.)

Parameters

- **code** – alphanumeric data to be printed as bar code
- **bc** – barcode format, possible values are for type A are:
  - UPC-A
  - UPC-E
  - EAN13
  - EAN8
  - CODE39
  - ITF
  - NW7

  Possible values for type B:
  - All types from function type A
  - CODE93
  - CODE128
  - GS1-128
  - GS1 DataBar Omnidirectional
  - GS1 DataBar Truncated
  - GS1 DataBar Limited

  - GS1 DataBar Expanded

  If none is specified, the method raises *BarcodeTypeError*.

- **height** (*int*) – barcode height, has to be between 1 and 255 *default*: 64
- **width** (*int*) – barcode width, has to be between 2 and 6 *default*: 3
- **pos** – where to place the text relative to the barcode, *default*: BELOW
  - ABOVE
  - BELOW
  - BOTH
  - OFF
- **font** – select font (see ESC/POS-documentation, the device often has two fonts), *default*: A
  - A
  - B
- **align_ct** (*bool*) – If this parameter is True the barcode will be centered. Otherwise no alignment command will be issued.
- **function_type** – Choose between ESCPOS function type A or B, depending on printer support and desired barcode. If not given, the printer will attempt to automatically choose the correct function based on the current profile. *default*: A
- **check** – If this parameter is True, the barcode format will be checked to ensure it meets the bc requirements as defigned in the esc/pos documentation. See py:func:*~escpos.Escpos.check_barcode* for more information. *default*: True.

Raises *BarcodeSizeError*, *BarcodeTypeError*, *BarcodeCodeError*

text(*txt*)

Print alpha-numeric text

The text has to be encoded in the currently selected codepage. The input text has to be encoded in unicode.

Parameters **txt** – text to be printed

Raises *TextError*

textln(*txt=u''*)

Print alpha-numeric text with a newline

The text has to be encoded in the currently selected codepage. The input text has to be encoded in unicode.

Parameters **txt** – text to be printed with a newline

Raises *TextError*

ln(*count=1*)

Print a newline or more

Parameters **count** – number of newlines to print

Raises `ValueError` if count < 0

block_text(*txt*, *font=None*, *columns=None*)

Text is printed wrapped to specified columns

Text has to be encoded in unicode.

Parameters

- **txt** – text to be printed
- **font** – font to be used, can be `a` or `b`
- **columns** – amount of columns

Returns None

set(*align=u'left'*, *font=u'a'*, *bold=False*, *underline=0*, *width=1*, *height=1*, *density=9*, *invert=False*, *smooth=False*, *flip=False*, *double_width=False*, *double_height=False*, *custom_size=False*)
Set text properties by sending them to the printer

Parameters

- **align** (*str*) – horizontal position for text, possible values are:
  - 'center'
  - 'left'
  - 'right'

  default: 'left'
- **font** (*str*) – font given as an index, a name, or one of the special values 'a' or 'b', referring to fonts 0 and 1.
- **bold** (*bool*) – text in bold, *default*: False
- **underline** (*bool*) – underline mode for text, decimal range 0-2, *default*: 0
- **double_height** (*bool*) – doubles the height of the text
- **double_width** (*bool*) – doubles the width of the text
- **custom_size** (*bool*) – uses custom size specified by width and height parameters. Cannot be used with double_width or double_height.
- **width** (*int*) – text width multiplier when custom_size is used, decimal range 1-8, *default*: 1
- **height** (*int*) – text height multiplier when custom_size is used, decimal range 1-8, *default*: 1
- **density** (*int*) – print density, value from 0-8, if something else is supplied the density remains unchanged
- **invert** (*bool*) – True enables white on black printing, *default*: False
- **smooth** (*bool*) – True enables text smoothing. Effective on 4x4 size text and larger, *default*: False
- **flip** (*bool*) – True enables upside-down printing, *default*: False

line_spacing(*spacing=None*, *divisor=180*)
Set line character spacing.

If no spacing is given, we reset it to the default.

There are different commands for setting the line spacing, using a different denominator:

'+" line_spacing/360 of an inch, 0 <= line_spacing <= 255 '3' line_spacing/180 of an inch, 0 <= line_spacing <= 255 'A' line_spacing/60 of an inch, 0 <= line_spacing <= 85

Some printers may not support all of them. The most commonly available command (using a divisor of 180) is chosen.

cut(*mode=u'FULL'*, *feed=True*)
Cut paper.

Without any arguments the paper will be cut completely. With 'mode=PART' a partial cut will be attempted. Note however, that not all models can do a partial cut. See the documentation of your printer for details.

Todo: Check this function on TM-T88II.

Parameters

- **mode** – set to 'PART' for a partial cut. default: 'FULL'
- **feed** – print and feed before cutting. default: true

Raises **ValueError** – if mode not in ('FULL', 'PART')

cashdraw(*pin*)
Send pulse to kick the cash drawer

Kick cash drawer on pin 2 or pin 5 according to default parameter. For non default parameter send a decimal sequence i.e. [27,112,48] or [27,112,0,25,255]

Parameters **pin** – pin number, 2 or 5 or list of decimals

Raises *CashDrawerError*

linedisplay_select(*select_display=False*)
Selects the line display or the printer

This method is used for line displays that are daisy-chained between your computer and printer. If you set *select_display* to true, only the display is selected and if you set it to false, only the printer is selected.

Parameters **select_display** (*bool*) – whether the display should be selected or the printer

linedisplay_clear()
Clears the line display and resets the cursor

This method is used for line displays that are daisy-chained between your computer and printer.

linedisplay(*text*)
Display text on a line display connected to your printer

You should connect a line display to your printer. You can do this by daisy-chaining the display between your computer and printer.

Parameters **text** – Text to display

hw(*hw*)
Hardware operations

Parameters **hw** – hardware action, may be:

- INIT
- SELECT
- RESET

print_and_feed(*n=1*)
Print data in print buffer and feed *n* lines

if n not in range (0, 255) then ValueError will be raised

Parameters **n** – number of n to feed. 0 <= n <= 255. default: 1

Raises **ValueError** – if not 0 <= n <= 255

control(*ctl*, *count=5*, *tab_size=8*)
Feed control sequences

Parameters

- **ctl** – string for the following control sequences:
  - LF *for Line Feed*
  - FF *for Form Feed*
  - CR *for Carriage Return*
  - HT *for Horizontal Tab*
  - VT *for Vertical Tab*
- **count** – integer between 1 and 32, controls the horizontal tab count. Defaults to 5.
- **tab_size** – integer between 1 and 255, controls the horizontal tab size in characters. Defaults to 8

Raises *TabPosError*

panel_buttons(*enable=True*)
Controls the panel buttons on the printer (e.g. FEED)

When enable is set to False the panel buttons on the printer will be disabled. Calling the method with enable=True or without argument will enable the panel buttons.

If panel buttons are enabled, the function of the panel button, such as feeding, will be executed upon pressing the button. If the panel buttons are disabled, pressing them will not have any effect.

This command is effective until the printer is initialized, reset or power-cycled. The default is enabled panel buttons.

Some panel buttons will always work, especially when printer is opened. See for more information the manual of your printer and the escpos-command-reference.

Parameters **enable** – controls the panel buttons

Return type None

query_status(*mode*)
Queries the printer for its status, and returns an array of integers containing it.

Parameters **mode** – Integer that sets the status mode queried to the printer. - RT_STATUS_ONLINE: Printer status. - RT_STATUS_PAPER: Paper sensor.

Return type array(integer)

is_online()
Queries the online status of the printer.

Returns When online, returns `True`; `False` otherwise.

Return type bool

paper_status()
Queries the paper status of the printer.

Returns 2 if there is plenty of paper, 1 if the paper has arrived to the near-end sensor and 0 if there is no paper.

Returns 2: Paper is adequate. 1: Paper ending. 0: No paper.

Return type int

6.3 Printers

Last Reviewed 2017-01-25

As of now there are 5 different type of printer implementations.

6.3.1 USB

The USB-class uses pyusb and libusb to communicate with USB-based printers. Note that this driver is not suited for USB-to-Serial-adapters and similiar devices, but only for those implementing native USB.

class escpos.printer.**Usb**(*idVendor*, *idProduct*, *usb_args=None*, *timeout=0*, *in_ep=130*, *out_ep=1*, **args*, ***kwargs*)

USB printer

This class describes a printer that natively speaks USB.

inheritance:

__init__(*idVendor*, *idProduct*, *usb_args=None*, *timeout=0*, *in_ep=130*, *out_ep=1*, **args*, ***kwargs*)

Parameters

- **idVendor** – Vendor ID
- **idProduct** – Product ID
- **usb_args** – Optional USB arguments (e.g. custom_match)
- **timeout** – Is the time limit of the USB operation. Default without timeout.
- **in_ep** – Input end point
- **out_ep** – Output end point

open(*usb_args*)

Search device on USB tree and set it as escpos device.

Parameters **usb_args** – USB arguments

close()

Release USB interface

6.3.2 Serial

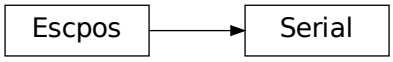
This driver uses pyserial in order to communicate with serial devices. If you are using an USB-based adapter to connect to the serial port, then you should also use this driver. The configuration is often based on DIP-switches that you can set on your printer. For the hardware-configuration please refer to your printer's manual.

class escpos.printer.**Serial**(*devfile=u'/dev/ttyS0'*, *baudrate=9600*, *bytesize=8*, *timeout=1*, *parity='N'*, *stopbits=1*, *xonxoff=False*, *dsrdtr=True*, **args*, ***kwargs*)

Serial printer

This class describes a printer that is connected by serial interface.

inheritance:



__init__(*devfile=u'/dev/ttyS0'*, *baudrate=9600*, *bytesize=8*, *timeout=1*, *parity='N'*, *stopbits=1*, *xonxoff=False*, *dsrdtr=True*, **args*, ***kwargs*)

Parameters

- **devfile** – Device file under dev filesystem
- **baudrate** – Baud rate for serial transmission
- **bytesize** – Serial buffer size
- **timeout** – Read/Write timeout
- **parity** – Parity checking
- **stopbits** – Number of stop bits
- **xonxoff** – Software flow control
- **dsrdtr** – Hardware flow control (False to enable RTS/CTS)

open()

Setup serial port and set is as escpos device

close()

Close Serial interface

6.3.3 Network

This driver is based on the socket class.

class escpos.printer.**Network**(*host*, *port=9100*, *timeout=60*, **args*, ***kwargs*)

Network printer

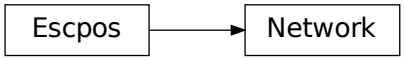
This class is used to attach to a networked printer. You can also use this in order to attach to a printer that is forwarded with `socat`.

If you have a local printer on parallel port `/dev/usb/lp0` then you could start `socat` with:

```
socat -u TCP4-LISTEN:4242,reuseaddr,fork OPEN:/dev/usb/lp0
```

Then you should be able to attach to port `4242` with this class. Otherwise the normal usecase would be to have a printer with ethernet interface. This type of printer should work the same with this class. For the address of the printer check its manuals.

inheritance:



__init__(*host*, *port=9100*, *timeout=60*, **args*, ***kwargs*)

Parameters

- **host** – Printer's hostname or IP address
- **port** – Port to write to
- **timeout** – timeout in seconds for the socket-library

open()
Open TCP socket with `socket`-library and set it as escpos device

close()
Close TCP connection

Troubleshooting

Problems with a network-attached printer can have numerous causes. Make sure that your device has a proper IP address. Often you can check the IP address by triggering the self-test of the device. As a next step try to send text manually to the device. You could use for example:

```
echo "OK\n" | nc IPADDRESS 9100
# the port number is often 9100
```

As a last resort try to reset the interface of the printer. This should be described in its manual.

6.3.4 File

This printer "prints" just into a file-handle. Especially on *nix-systems this comes very handy.

class escpos.printer.**File**(*devfile=u'/dev/usb/lp0'*, *auto_flush=True*, **args*, ***kwargs*)
Generic file printer

This class is used for parallel port printer or other printers that are directly attached to the filesystem. Note that you should stay away from using USB-to-Parallel-Adapter since they are unreliable and produce arbitrary errors.

inheritance:

__init__(*devfile=u'/dev/usb/lp0'*, *auto_flush=True*, **args*, ***kwargs*)

Parameters

- **devfile** – Device file under dev filesystem
- **auto_flush** – automatically call flush after every call of _raw()

open()
Open system file

flush()
Flush printing content

close()
Close system file

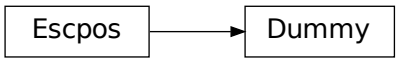
6.3.5 Dummy

The Dummy-printer is mainly for testing- and debugging-purposes. It stores all of the "output" as raw ESC/POS in a string and returns that.

class escpos.printer.**Dummy**(**args*, ***kwargs*)
Dummy printer

This class is used for saving commands to a variable, for use in situations where there is no need to send commands to an actual printer. This includes generating print jobs for later use, or testing output.

inheritance:



output
Get the data that was sent to this printer

clear()
Clear the buffer of the printer

This method can be called if you send the contents to a physical printer and want to use the Dummy printer for new output.

6.4 Raspberry Pi

Last Reviewed 2017-01-05

This instructions were tested on Raspbian Jessie.

Warning: You should **never** directly connect an printer with RS232-interface (serial port) directly to a Raspberry PI or similar interface (e.g. those simple USB-sticks without encasing). Those interfaces are based on 5V- or 3,3V-logic (the latter in the case of Raspberry PI). Classical RS232 uses 12V-logic and would **thus destroy your interface**. Connect both systems with an appropriate *level shifter*.

6.4.1 Dependencies

First, install the packages available on Raspbian.

```
sudo apt-get install python3 python3-setuptools python3-pip libjpeg8-dev
```

6.4.2 Installation

You can install by using pip3.

```
sudo pip3 install --upgrade pip
sudo pip3 install python-escpos
```

6.4.3 Run

You need sudo and python3 to run your program.

```
sudo python3 your-program.py
```

Now you can attach your printer and and test it with the example code in the project's set of examples. You can find that in the project-repository.

For more details on this check the *installation-manual*.

6.5 TODO

6.5.1 Introduction

python-escpos is the initial idea, from here we can start to build a robust library to get most of the ESC/POS printers working with this library.

Eventually, this library must be able to cover almost all the defined models detailed in the ESC/POS Command Specification Manual.

6.5.2 Details

What things are planned to work on?

Testing

- Test on many printers as possible (USB, Serial, Network)
- automate testing

Design

- Add all those sequences which are not common, but part of the ESC/POS Command Specifications.
  - Port to Python 3
  - Windows compatibility (hidapi instead libusb?)
  - PDF417 support
- use something similar to the *capabilities* in escpos-php

Todos in the codebase

Todo: Add a method to compute the checksum for the different standards

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/python-escpos/envs/v3.0a5/lib/python2.7/site-packages/python_escpos-3.0a5-py2.7.egg/escpos/escpos.py:docstring of escpos.escpos.Escpos.check_barcode, line 9.)

Todo: For fixed-length standards with mandatory checksum (EAN, UPC), compute and add the checksum automatically if missing.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/python-escpos/envs/v3.0a5/lib/python2.7/site-packages/python_escpos-3.0a5-py2.7.egg/escpos/escpos.py:docstring of escpos.escpos.Escpos.check_barcode, line 11.)

Todo: On TM-T88II width from 1 to 6 is accepted. Try to acquire command reference and correct the code.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/python-escpos/envs/v3.0a5/lib/python2.7/site-packages/python_escpos-3.0a5-py2.7.egg/escpos/escpos.py:docstring of escpos.escpos.Escpos.barcode, line 14.)

Todo: Supplying pos does not have an effect for every barcode type. Check and document for which types this is true.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/python-escpos/envs/v3.0a5/lib/python2.7/site-packages/python_escpos-3.0a5-py2.7.egg/escpos/escpos.py:docstring of escpos.escpos.Escpos.barcode, line 15.)

Todo: If further barcode-types are needed they could be rendered transparently as an image. (This could also be of help if the printer does not support types that others do.)

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/python-escpos/envs/v3.0a5/lib/python2.7/site-packages/python_escpos-3.0a5-py2.7.egg/escpos/escpos.py:docstring of escpos.escpos.Escpos.barcode, line 22.)

Todo: Check this function on TM-T88II.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/python-escpos/envs/v3.0a5/lib/python2.7/site-packages/python_escpos-3.0a5-py2.7.egg/escpos/escpos.py:docstring of escpos.escpos.Escpos.cut, line 7.)

Todo: On TM-T88II width from 1 to 6 is accepted. Try to acquire command reference and correct the code.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/python-escpos/envs/v3.0a5/lib/python2.7/site-packages/python_escpos-3.0a5-py2.7.egg/escpos/escpos.py:docstring of escpos.escpos.Escpos.barcode, line 14.)

Todo: Supplying pos does not have an effect for every barcode type. Check and document for which types this is true.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/python-escpos/envs/v3.0a5/lib/python2.7/site-packages/python_escpos-3.0a5-py2.7.egg/escpos/escpos.py:docstring of escpos.escpos.Escpos.barcode, line 15.)

Todo: If further barcode-types are needed they could be rendered transparently as an image. (This could also be of help if the printer does not support types that others do.)

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/python-escpos/envs/v3.0a5/lib/python2.7/site-packages/python_escpos-3.0a5-py2.7.egg/escpos/escpos.py:docstring of escpos.escpos.Escpos.barcode, line 22.)

Todo: Add a method to compute the checksum for the different standards

---

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/python-escpos/envs/v3.0a5/lib/python2.7/site-packages/python_escpos-3.0a5-py2.7.egg/escpos/escpos.py:docstring of escpos.escpos.Escpos.check_barcode, line 9.)

Todo: For fixed-length standards with mandatory checksum (EAN, UPC), compute and add the checksum automatically if missing.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/python-escpos/envs/v3.0a5/lib/python2.7/site-packages/python_escpos-3.0a5-py2.7.egg/escpos/escpos.py:docstring of escpos.escpos.Escpos.check_barcode, line 11.)

Todo: On TM-T88II width from 1 to 6 is accepted. Try to acquire command reference and correct the code.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/python-escpos/envs/v3.0a5/lib/python2.7/site-packages/python_escpos-3.0a5-py2.7.egg/escpos/escpos.py:docstring of

escpos.escpos.Escpos.barcode, line 14.)

Todo: Supplying pos does not have an effect for every barcode type. Check and document for which types this is true.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/python-escpos/envs/v3.0a5/lib/python2.7/site-packages/python_escpos-3.0a5-py2.7.egg/escpos/escpos.py:docstring of escpos.escpos.Escpos.barcode, line 15.)

Todo: If further barcode-types are needed they could be rendered transparently as an image. (This could also be of help if the printer does not support types that others do.)

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/python-escpos/envs/v3.0a5/lib/python2.7/site-packages/python_escpos-3.0a5-py2.7.egg/escpos/escpos.py:docstring of escpos.escpos.Escpos.barcode, line 22.)

Todo: Check this function on TM-T88II.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/python-escpos/envs/v3.0a5/lib/python2.7/site-packages/python_escpos-3.0a5-py2.7.egg/escpos/escpos.py:docstring of escpos.escpos.Escpos.cut, line 7.)

6.6 Usage

Last Reviewed 2017-06-10

6.6.1 Define your printer

USB printer

Before creating your Python ESC/POS printer instance, consult the system to obtain the printer parameters. This is done with the 'lsusb' command.

Run the command and look for the "Vendor ID" and "Product ID" and write down the values. These values are displayed just before the name of the device with the following format:

```
xxxx:xxxx
```

Example:

```
# lsusb
Bus 002 Device 001: ID 04b8:0202 Epson ...
```

Write down the the values in question, then issue the following command so you can get the "Interface" number and "End Point"

```
# lsusb -vvv -d xxxx:xxxx | grep iInterface
    iInterface              0
# lsusb -vvv -d xxxx:xxxx | grep bEndpointAddress | grep OUT
      bEndpointAddress     0x01  EP 1 OUT
```

The first command will yield the "Interface" number that must be handy to have and the second yields the "Output Endpoint" address.

USB Printer initialization

```
p = printer.Usb(0x04b8,0x0202)
```

By default the "Interface" number is "0" and the "Output Endpoint" address is "0x01". If you have other values then you can define them on your instance. So, assuming that we have another printer where in_ep is on 0x81 and out_ep=0x02, then the printer definition should look like:

Generic USB Printer initialization

```
p = printer.Usb(0x1a2b,0x1a2b,0,0x81,0x02)
```

Network printer

You only need the IP of your printer, either because it is getting its IP by DHCP or you set it manually.

Network Printer initialization

```
p = printer.Network("192.168.1.99")
```

Serial printer

Most of the default values set by the DIP switches for the serial printers, have been set as default on the serial printer class, so the only thing you need to know is which serial port the printer is connected to.

Serial printer initialization

```
p = printer.Serial("/dev/tty0")

# on a Windows OS serial devices are typically accessible as COM
p = printer.Serial("COM1")
```

Other printers

Some printers under */dev* can't be used or initialized with any of the methods described above. Usually, those are printers used by printcap, however, if you know the device name, you could try to initialize by passing the device node name.

```
p = printer.File("/dev/usb/lp1")
```

The default is "/dev/usb/lp0", so if the printer is located on that node, then you don't necessary need to pass the node name.

6.6.2 Define your instance

The following example demonstrates how to initialize the Epson TM-TI88IV on a USB interface.

```
from escpos import *
""" Seiko Epson Corp. Receipt Printer M129 Definitions (EPSON TM-T88IV) """
p = printer.Usb(0x04b8,0x0202)
# Print text
p.text("Hello World\n")
# Print image
p.image("logo.gif")
# Print QR Code
p.qr("You can readme from your smartphone")
# Print barcode
p.barcode('1324354657687','EAN13',64,2,'','')
# Cut paper
p.cut()
```

6.6.3 Configuration File

You can create a configuration file for python-escpos. This will allow you to use the CLI, and skip some setup when using the library programmatically.

The default configuration file is named `config.yaml` and uses the YAML format. For windows it is probably at:

```
%appdata%/python-escpos/config.yaml
```

And for linux:

```
$HOME/.config/python-escpos/config.yaml
```

If you aren't sure, run:

```
from escpos import config
c = config.Config()
c.load()
```

If it can't find the configuration file in the default location, it will tell you where it's looking. You can always pass a path, or a list of paths, to the `load()` method.

To load the configured printer, run:

```
from escpos import config
c = config.Config()
printer = c.printer()
```

The printer section

The `printer` configuration section defines a default printer to create.

The only required paramter is `type`. The value of this has to be one of the printers defined in *Printers*.

The rest of the given parameters will be passed on to the initialization of the printer class. Use these to overwrite the default values as specified in *Printers*. This implies that the parameters have to match the parameter-names of the respective printer class.

An example file printer:

```
printer:
        type: File
        devfile: /dev/someprinter
```

And for a network printer:

```
printer:
        type: Network
        host: 127.0.0.1
        port: 9000
```

An USB-printer could be defined by:

```
printer:
        type: Usb
        idVendor: 0x1234
        idProduct: 0x5678
        in_ep: 0x66
        out_ep: 0x01
```

6.6.4 Printing text right

Python-escpos is designed to accept unicode. So make sure that you use `u'strings'` or import `unicode_literals` from `__future__` if you are on Python 2. On Python 3 you should be fine.

For normal usage you can simply pass your text to the printers `text()`-function. It will automatically guess the right codepage and then send the encoded data to the printer. If this feature does not work, please try to isolate the error and then create an issue on the Github project page.

If you want or need to you can manually set the codepage. For this please use the `charcode()`-function. You can set any key-value that is in `CHARCODE`. If something is wrong, an `CharCodeError` will be raised. After you have manually set the codepage the printer won't change it anymore. You can revert to normal behaviour by setting charcode to `AUTO`.

6.6.5 Advanced Usage: Print from binary blob

Imagine you have a file with ESC/POS-commands in binary form. This could be useful for testing capabilities of your printer with a known working combination of commands. You can print this data with the following code, using the standard methods of python-escpos. (This is an advantage of the fact that *_raw()* accepts binary strings.)

```
from escpos import printer
p = printer.Serial()  # adapt this to your printer model

file = open("binary-blob.bin", "rb")  # read in the file containing your commands in␣
↪binary-mode
data = file.read()
file.close()

p._raw(data)
```

That's all, the printer should then print your data. You can also use this technique to let others reproduce an issue that you have found. (Just "print" your commands to a File-printer on your local filesystem.) However, please keep in mind, that often it is easier and better to just supply the code that you are using.

Here you can download an example, that will print a set of common barcodes:

- `barcode.bin` by @mike42

6.6.6 Advanced Usage: change capabilities-profile

Packaged together with the escpos-code is a capabilities-file. This file in JSON-format describes the capabilities of different printers. It is developed and hosted in escpos-printer-db.

Certain applications like the usage of cx_freeze might change the packaging structure. This leads to the capabilities-profile not being found. In this case you can use the environment-variable *ESCPOS_CAPABILITIES_FILE*. The following code is an example.

```
# use packaged capabilities-profile
python-escpos cut

# use capabilities-profile that you have put in /usr/python-escpos
export ESCPOS_CAPABILITIES_FILE=/usr/python-escpos/capabilities.json
python-escpos cut

# use packaged file again
unset ESCPOS_CAPABILITIES_FILE
python-escpos cut
```

6.6.7 Hint: preprocess printing

Printing images directly to the printer is rather slow. One factor that slows down the process is the transmission over e.g. serial port.

Apart from configuring your printer to use the maximum baudrate (in the case of serial-printers), there is not much that you can do. However you could use the *escpos.printer.Dummy*-printer to preprocess your image. This is probably best explained by an example:

```
from escpos.printer import Serial, Dummy

p = Serial()
d = Dummy()

# create ESC/POS for the print job, this should go really fast
d.text("This is my image:\n")
d.image("funny_cat.png")
d.cut()

# send code to printer
p._raw(d.output)
```

This way you could also store the code in a file and print it later. You could then for example print the code from another process than your main-program and thus reduce the waiting time. (Of course this will not make the printer print faster.)

6.7 Printing Barcodes

Last Reviewed 2016-07-31

Most ESC/POS-printers implement barcode-printing. The barcode-commandset is implemented in the barcode-method. For a list of compatible barcodes you should check the manual of your printer. As a rule of thumb: even older

Epson-models support most 1D-barcodes. To be sure just try some implementations and have a look at the notices below.

6.7.1 barcode-method

The barcode-method is rather low-level and orients itself on the implementation of ESC/POS. In the future this class could be supplemented by a high-level class that helps the user generating the payload.

`Escpos.`**`barcode`**(*code*, *bc*, *height=64*, *width=3*, *pos=u'BELOW'*, *font=u'A'*, *align_ct=True*, *function_type=None*, *check=True*)

Print Barcode

This method allows to print barcodes. The rendering of the barcode is done by the printer and therefore has to be supported by the unit. By default, this method will check whether your barcode text is correct, that is the characters and lengths are supported by ESCPOS. Call the method with *check=False* to disable the check, but note that uncorrect barcodes may lead to unexpected printer behaviour. There are two forms of the barcode function. Type A is default but has fewer barcodes, while type B has some more to choose from.

Use the parameters *height* and *width* for adjusting of the barcode size. Please take notice that the barcode will not be printed if it is outside of the printable area. (Which should be impossible with this method, so this information is probably more useful for debugging purposes.)

Todo: On TM-T88II width from 1 to 6 is accepted. Try to acquire command reference and correct the code.

Todo: Supplying pos does not have an effect for every barcode type. Check and document for which types this is true.

If you do not want to center the barcode you can call the method with *align_ct=False*, which will disable automatic centering. Please note that when you use center alignment, then the alignment of text will be changed automatically to centered. You have to manually restore the alignment if necessary.

Todo: If further barcode-types are needed they could be rendered transparently as an image. (This could also be of help if the printer does not support types that others do.)

Parameters

- **`code`** – alphanumeric data to be printed as bar code
- **`bc`** – barcode format, possible values are for type A are:
  - UPC-A
  - UPC-E
  - EAN13
  - EAN8
  - CODE39
  - ITF
  - NW7

  Possible values for type B:

  - All types from function type A
  - CODE93
  - CODE128
  - GS1-128
  - GS1 DataBar Omnidirectional
  - GS1 DataBar Truncated
  - GS1 DataBar Limited
  - GS1 DataBar Expanded

  If none is specified, the method raises *BarcodeTypeError*.
- **height** (*int*) – barcode height, has to be between 1 and 255 *default*: 64
- **width** (*int*) – barcode width, has to be between 2 and 6 *default*: 3
- **pos** – where to place the text relative to the barcode, *default*: BELOW
  - ABOVE
  - BELOW
  - BOTH
  - OFF
- **font** – select font (see ESC/POS-documentation, the device often has two fonts), *default*: A
  - A
  - B
- **align_ct** (*bool*) – If this parameter is True the barcode will be centered. Otherwise no alignment command will be issued.
- **function_type** – Choose between ESCPOS function type A or B, depending on printer support and desired barcode. If not given, the printer will attempt to automatically choose the correct function based on the current profile. *default*: A
- **check** – If this parameter is True, the barcode format will be checked to ensure it meets the bc requirements as defigned in the esc/pos documentation. See py:func:*~escpos.Escpos.check_barcode* for more information. *default*: True.

Raises *BarcodeSizeError*, *BarcodeTypeError*, *BarcodeCodeError*

6.7.2 CODE128

Code128 barcodes need a certain format. For now the user has to make sure that the payload is correct. For alphanumeric CODE128 you have to preface your payload with *{B*.

```
from escpos.printer import Dummy, Serial
p = Serial()
# print CODE128 012ABCDabcd
p.barcode("{B012ABCDabcd", "CODE128", function_type="B")
```

A very good description on CODE128 is also on Wikipedia.

6.8 Contributing

This project is open to any kind of contribution. You can help with improving the documentation, adding fixes to the code, providing test cases in code or as a description or just spreading the word. Please feel free to create an issue or pull request. In order to reduce the amount of work for everyone please try to adhere to good practice.

The pull requests and issues will be prefilled with templates. Please fill in your information where applicable.

This project uses semantic versioning and tries to adhere to the proposed rules as well as possible.

6.8.1 Author-list

This project keeps a list of authors. This can be auto-generated by calling *./doc/generate-authors.sh*. When contributing the first time, please include a commit with the output of this script in place. Otherwise the integration-check will fail.

When you change your username or mail-address, please also update the *.mailmap* and the authors-list. You can find a good documentation on the mapping-feature in the documentation of git-shortlog.

6.8.2 Style-Guide

When writing code please try to stick to these rules.

Python 2 and 3

We have rewritten the code in order to maintain compatibility with both Python 2 and Python 3. In order to ensure that we do not miss any accidental degradation, please add these imports to the top of every file of code:

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
from __future__ import unicode_literals
```

Furthermore please be aware of the differences between Python 2 and 3. For example this guide is helpful. Special care has to be taken when dealing with strings and byte-strings. Please note that the `_raw()`-method only accepts byte-strings. Often you can achieve compatibility quite easily with a tool from the *six*-package.

PEP8

The entire codebase adheres to the rules of PEP8. These rules are enforced by running *flake8* in the integration-checks. Please adhere to these rules as your contribution can only be merged if the check succeeds. You can use flake8 or similar tools locally in order to check your code. Apart from that the travis-log and the check by Landscape will provide you with hints.

GIT

The master-branch contains code that has been released to PyPi. A release is marked with a tag corresponding to the version. Issues are closed when they have been resolved in the development-branch.

When you have a change to make, begin by creating a new branch from the HEAD of *python-escpos/development*. Name your branch to indicate what you are trying to achieve. Good branch names might be *improve/text-handling*, *feature/enable-color-printing*.

Please try to group your commits into logical units. If you need to tidy up your branch, you can make use of a git feature called an 'interactive rebase' before making a pull request. A small, self-contained change-set is easier to review, and improves the chance of your code being merged. Please also make sure that before creating your PR, your branch is rebased on a recent commit or you merged a recent commit into your branch. This way you can ensure that your PR is without merge conflicts.

Docstrings

This project tries to have a good documentation. Please add a docstring to every method and class. Have a look at existing methods and classes for the style. We use basically standard rst-docstrings for Sphinx.

Test

Try to write tests whenever possible. Our goal for the future is 100% coverage. We are currently using *nose* but might change in the future. You can copy the structure from other testcases. Please remember to adapt the docstrings.

Further reading

For further best practices and hints on contributing please see the contribution-guide. Should there be any contradictions between this guide and the linked one, please stick to this text. Aside from that feel free to create an issue or write an email if anything is unclear.

Thank you for your contribution!

6.9 Changelog

6.9.1 2019-06-16 - Version 3.0a5 - "Lightly Seared On The Reality Grill"

This release is the sixth alpha release of the new version 3.0. Please be aware that the API is subject to change until v3.0 is released.

changes

- allow arbitrary USB arguments in USB-class
- add Win32Raw-Printer on Windows-platforms
- add and improve Windows support of USB-class
- use pyyaml safe_load()
- improve doc
- implement _read method of Network printer class

contributors

- Patrick Kanzler
- Gerard Marull-Paretas
- Ramon Poca

- akeonly
- Omer Akram
- Justin Vieira

6.9.2 2018-05-15 - Version 3.0a4 - "Kakistocrat"

This release is the fifth alpha release of the new version 3.0. Please be aware that the API will still change until v3.0 is released.

changes

- raise exception when TypeError occurs in cashdraw (#268)
- Feature/clear content in dummy printer (#271)
- fix is_online() (#282)
- improve documentation
- Modified submodule to always pull from master branch (#283)
- parameter for implementation of nonnative qrcode (#289)
- improve platform independence (#296)

contributors

- Christoph Heuel
- Patrick Kanzler
- kennedy
- primax79
- reck31
- Thijs Triemstra

6.9.3 2017-10-08 - Version 3.0a3 - "Just Testing"

This release is the fourth alpha release of the new version 3.0. Please be aware that the API will still change until v3.0 is released.

changes

- minor changes in documentation, tests and examples
- pickle capabilities for faster startup
- first implementation of centering images and QR
- check barcodes based on regex

contributors

- Patrick Kanzler
- Lucy Linder
- Romain Porte
- Sergio Pulgarin

6.9.4 2017-08-04 - Version 3.0a2 - "It's My Party And I'll Sing If I Want To"

This release is the third alpha release of the new version 3.0. Please be aware that the API will still change until v3.0 is released.

changes

- refactor of the set-method
- preliminary support of POS "line display" printing
- improvement of tests
- added ImageWidthError
- list authors in repository
- add support for software-based barcode-rendering
- fix SerialException when trying to close device on __del__
- added the DLE EOT querying command for USB and Serial
- ensure QR codes have a large enough border
- make feed for cut optional
- fix the behavior of horizontal tabs
- added test script for hard an soft barcodes
- implemented paper sensor querying command
- added weather forecast example script
- added a method for simpler newlines

contributors

- csoft2k
- Patrick Kanzler
- mrwunderbar666
- Romain Porte
- Ahmed Tahri

6.9.5 2017-03-29 - Version 3.0a1 - "Headcrash"

This release is the second alpha release of the new version 3.0. Please be aware that the API will still change until v3.0 is released.

changes

- automatically upload releases to GitHub
- add environment variable ESCPOS_CAPABILITIES_FILE
- automatically handle cases where full cut or partial cut is not available
- add print_and_feed

contributors

- Sam Cheng
- Patrick Kanzler
- Dmytro Katyukha

6.9.6 2017-01-31 - Version 3.0a - "Grey Area"

This release is the first alpha release of the new version 3.0. Please be aware that the API will still change until v3.0 is released.

changes

- change the project's license to MIT in accordance with the contributors (see python-escpos/python-escpos#171)
- feature: add "capabilities" which are shared with escpos-php, capabilities are stored in escpos-printer-db
- feature: the driver tries now to guess the appropriate codepage and sets it automatically (called "magic encode")
- as an alternative you can force the codepage with the old API
- updated and improved documentation
- changed constructor of main class due to introduction of capablities
- changed interface of method *blocktext*, changed behavior of multiple methods, for details refer to the documentation on python-escpos.readthedocs.io
- add support for custom cash drawer sequence
- enforce flake8 on the src-files, test py36 and py37 on travis

contributors

- Michael Billington
- Michael Elsdörfer
- Patrick Kanzler (with code by Frédéric Van der Essen)
- Asuki Kono

- Benito López
- Curtis // mashedkeyboard
- Thijs Triemstra
- ysuolmai

6.9.7 2016-08-26 - Version 2.2.0 - "Fate Amenable To Change"

changes

- fix improper API-use in qrcode()
- change setup.py shebang to make it compatible with virtualenvs.
- add constants for sheet mode and colors
- support changing the linespacing

contributors

- Michael Elsdörfer
- Patrick Kanzler

6.9.8 2016-08-10 - Version 2.1.3 - "Ethics Gradient"

changes

- configure readthedocs and travis
- update doc with hint on image preprocessing
- add fix for printing large images (by splitting them into multiple images)

contributors

- Patrick Kanzler

6.9.9 2016-08-02 - Version 2.1.2 - "Death and Gravity"

changes

- fix File-printer: flush after every call of _raw()
- fix lists in documentation
- fix CODE128: by adding the control character to the barcode-selection-sequence the barcode became unusable

contributors

- Patrick Kanzler

6.9.10 2016-08-02 - Version 2.1.1 - "Contents May Differ"

changes

- rename variable interface in USB-class to timeout
- add support for hypothesis and move pypy3 to the allowed failures (pypy3 is not supported by hypothesis)

contributors

- Patrick Kanzler
- Renato Lorenzi

6.9.11 2016-07-23 - Version 2.1.0 - "But Who's Counting?"

changes

- packaging: configured the coverage-analysis codecov.io
- GitHub: improved issues-template
- documentation: add troubleshooting tip to network-interface
- the module, cli and documentation is now aware of the version of python-escpos
- the cli does now support basic tabcompletion

contributors

- Patrick Kanzler

6.9.12 2016-06-24 - Version 2.0.0 - "Attitude Adjuster"

This version is based on the original version of python-escpos by Manuel F Martinez. However, many contributions have greatly improved the old codebase. Since this version does not completely match the interface of the version published on PyPi and has many improvements, it will be released as version 2.0.0.

changes

- refactor complete code in order to be compatible with Python 2 and 3
- modernize packaging
- add testing and CI
- merge various forks into codebase, fixing multiple issues with barcode-, QR-printing, cashdraw and structure
- improve the documentation
- extend support of barcode-codes to type B
- add function to disable panel-buttons
- the text-functions are now intended for unicode, the driver will automatically encode the string based on the selected codepage

- the image-functions are now much more flexible
- added a CLI
- restructured the constants

contributors

- Thomas van den Berg
- Michael Billington
- Nate Bookham
- Davis Goglin
- Christoph Heuel
- Patrick Kanzler
- Qian LinFeng

6.9.13 2016-01-24 - Version 1.0.9

- fix constant definition for PC1252
- move documentation to Sphinx

6.9.14 2015-10-27 - Version 1.0.8

- **Merge pull request #59 from zouppen/master**
  - Support for images vertically longer than 256 pixels
  - Sent by Joel Lehtonen <joel.lehtonen@koodilehto.fi>
- Updated README

6.9.15 2015-08-22 - Version 1.0.7

- Issue #57: Fixed transparent images

6.9.16 2015-07-06 - Version 1.0.6

- **Merge pull request #53 from ldos/master**
  - Extended params for serial printers
  - Sent by ldos <cafeteria.ldosalzira@gmail.com>

6.9.17 2015-04-21 - Version 1.0.5

- **Merge pull request #45 from Krispy2009/master**
  - Raising the right error when wrong charcode is used
  - Sent by Kristi <Krispy2009@gmail.com>

6.9.18 2014-05-20 - Version 1.0.4

- Issue #20: Added Density support (Sent by thomas.erbacher@ragapack.de)
- Added charcode tables
- Fixed Horizontal Tab
- Fixed code tabulators

6.9.19 2014-02-23 - Version 1.0.3

- Issue #18: Added quad-area characters (Sent by syncman1x@gmail.com)
- Added exception for PIL import

6.9.20 2013-12-30 - Version 1.0.2

- Issue #5: Fixed vertical tab
- Issue #9: Fixed identation inconsistence

6.9.21 2013-03-14 - Version 1.0.1

- Issue #8: Fixed set font
- Added QR support

6.9.22 2012-11-15 - Version 1.0

- Issue #2: Added ethernet support
- Issue #3: Added compatibility with libusb-1.0.1
- Issue #4: Fixed typo in escpos.py

6.10 Esc/Pos

Module *escpos.escpos*

Main class

This module contains the abstract base class *Escpos*.

author Manuel F Martinez and others

organization Bashlinux and python-escpos

copyright Copyright (c) 2012-2017 Bashlinux and python-escpos

license MIT

class escpos.escpos.**Escpos**(*profile=None*, *magic_encode_args=None*, ***kwargs*)

Bases: object

ESC/POS Printer object

This class is the abstract base class for an esc/pos-printer. The printer implementations are children of this class.

device = None

image(*img_source*, *high_density_vertical=True*, *high_density_horizontal=True*, *impl=u'bitImageRaster'*, *fragment_height=960*, *center=False*)
Print an image

You can select whether the printer should print in high density or not. The default value is high density. When printing in low density, the image will be stretched.

Esc/Pos supplies several commands for printing. This function supports three of them. Please try to vary the implementations if you have any problems. For example the printer *IT80-002* will have trouble aligning images that are not printed in Column-mode.

The available printing implementations are:

- *bitImageRaster*: prints with the *GS v 0*-command
- *graphics*: prints with the *GS (L*-command
- *bitImageColumn*: prints with the *ESC* *-command

Parameters

- **img_source** – PIL image or filename to load: *jpg*, *gif*, *png* or *bmp*
- **high_density_vertical** – print in high density in vertical direction *default:* True
- **high_density_horizontal** – print in high density in horizontal direction *default:* True
- **impl** – choose image printing mode between *bitImageRaster*, *graphics* or *bitImageColumn*
- **fragment_height** – Images larger than this will be split into multiple fragments *default:* 960
- **center** – Center image horizontally *default:* False

qr(*content*, *ec=0*, *size=3*, *model=2*, *native=False*, *center=False*, *impl=u'bitImageRaster'*)
Print QR Code for the provided string

Parameters

- **content** – The content of the code. Numeric data will be more efficiently compacted.
- **ec** – Error-correction level to use. One of QR_ECLEVEL_L (default), QR_ECLEVEL_M, QR_ECLEVEL_Q or QR_ECLEVEL_H. Higher error correction results in a less compact code.
- **size** – Pixel size to use. Must be 1-16 (default 3)
- **model** – QR code model to use. Must be one of QR_MODEL_1, QR_MODEL_2 (default) or QR_MICRO (not supported by all printers).
- **native** – True to render the code on the printer, False to render the code as an image and send it to the printer (Default)
- **center** – Centers the code *default:* False

charcode(*code=u'AUTO'*)
Set Character Code Table

Sets the control sequence from CHARCODE in *escpos.constants* as active. It will be sent with the next text sequence. If you set the variable code to AUTO it will try to automatically guess the right codepage. (This is the standard behaviour.)

Parameters **code** – Name of CharCode

Raises *CharCodeError*

static check_barcode(*bc*, *code*)

This method checks if the barcode is in the proper format. The validation concerns the barcode length and the set of characters, but won't compute/validate any checksum. The full set of requirement for each barcode type is available in the ESC/POS documentation.

As an example, using EAN13, the barcode *12345678901* will be correct, because it can be rendered by the printer. But it does not suit the EAN13 standard, because the checksum digit is missing. Adding a wrong checksum in the end will also be considered correct, but adding a letter won't (EAN13 is numeric only).

Todo: Add a method to compute the checksum for the different standards

---

Todo: For fixed-length standards with mandatory checksum (EAN, UPC), compute and add the checksum automatically if missing.

Parameters

- **bc** – barcode format, see :py:func'~escpos.Escpos.barcode'
- **code** – alphanumeric data to be printed as bar code, see :py:func'~escpos.Escpos.barcode'

Returns bool

barcode(*code*, *bc*, *height=64*, *width=3*, *pos=u'BELOW'*, *font=u'A'*, *align_ct=True*, *function_type=None*, *check=True*)

Print Barcode

This method allows to print barcodes. The rendering of the barcode is done by the printer and therefore has to be supported by the unit. By default, this method will check whether your barcode text is correct, that is the characters and lengths are supported by ESCPOS. Call the method with *check=False* to disable the check, but note that uncorrect barcodes may lead to unexpected printer behaviour. There are two forms of the barcode function. Type A is default but has fewer barcodes, while type B has some more to choose from.

Use the parameters *height* and *width* for adjusting of the barcode size. Please take notice that the barcode will not be printed if it is outside of the printable area. (Which should be impossible with this method, so this information is probably more useful for debugging purposes.)

Todo: On TM-T88II width from 1 to 6 is accepted. Try to acquire command reference and correct the code.

Todo: Supplying pos does not have an effect for every barcode type. Check and document for which types this is true.

If you do not want to center the barcode you can call the method with *align_ct=False*, which will disable automatic centering. Please note that when you use center alignment, then the alignment of text will be changed automatically to centered. You have to manually restore the alignment if necessary.

Todo: If further barcode-types are needed they could be rendered transparently as an image. (This could also be of help if the printer does not support types that others do.)

Parameters

- **code** – alphanumeric data to be printed as bar code
- **bc** – barcode format, possible values are for type A are:
  - UPC-A
  - UPC-E
  - EAN13
  - EAN8
  - CODE39
  - ITF
  - NW7

  Possible values for type B:
  - All types from function type A
  - CODE93
  - CODE128
  - GS1-128
  - GS1 DataBar Omnidirectional
  - GS1 DataBar Truncated
  - GS1 DataBar Limited
  - GS1 DataBar Expanded

  If none is specified, the method raises *BarcodeTypeError*.
- **height** (*int*) – barcode height, has to be between 1 and 255 *default*: 64
- **width** (*int*) – barcode width, has to be between 2 and 6 *default*: 3
- **pos** – where to place the text relative to the barcode, *default*: BELOW
  - ABOVE
  - BELOW
  - BOTH
  - OFF
- **font** – select font (see ESC/POS-documentation, the device often has two fonts), *default*: A
  - A
  - B

- **align_ct** (*bool*) – If this parameter is True the barcode will be centered. Otherwise no alignment command will be issued.
- **function_type** – Choose between ESCPOS function type A or B, depending on printer support and desired barcode. If not given, the printer will attempt to automatically choose the correct function based on the current profile. *default*: A
- **check** – If this parameter is True, the barcode format will be checked to ensure it meets the bc requirements as defigned in the esc/pos documentation. See py:func:*~escpos.Escpos.check_barcode* for more information. *default*: True.

Raises *BarcodeSizeError*, *BarcodeTypeError*, *BarcodeCodeError*

soft_barcode(*barcode_type*, *data*, *impl=u'bitImageColumn'*, *module_height=5*, *module_width=0.2*, *text_distance=1*)

text(*txt*)

Print alpha-numeric text

The text has to be encoded in the currently selected codepage. The input text has to be encoded in unicode.

Parameters **txt** – text to be printed

Raises *TextError*

textln(*txt=u''*)

Print alpha-numeric text with a newline

The text has to be encoded in the currently selected codepage. The input text has to be encoded in unicode.

Parameters **txt** – text to be printed with a newline

Raises *TextError*

ln(*count=1*)

Print a newline or more

Parameters **count** – number of newlines to print

Raises `ValueError` if count < 0

block_text(*txt*, *font=None*, *columns=None*)

Text is printed wrapped to specified columns

Text has to be encoded in unicode.

Parameters

- **txt** – text to be printed
- **font** – font to be used, can be `a` or `b`
- **columns** – amount of columns

Returns None

set(*align=u'left'*, *font=u'a'*, *bold=False*, *underline=0*, *width=1*, *height=1*, *density=9*, *invert=False*, *smooth=False*, *flip=False*, *double_width=False*, *double_height=False*, *custom_size=False*)

Set text properties by sending them to the printer

Parameters

- **align** (*str*) – horizontal position for text, possible values are:
  - 'center'
  - 'left'

– 'right'

default: 'left'

- **font** (*str*) – font given as an index, a name, or one of the special values 'a' or 'b', referring to fonts 0 and 1.
- **bold** (*bool*) – text in bold, *default*: False
- **underline** (*bool*) – underline mode for text, decimal range 0-2, *default*: 0
- **double_height** (*bool*) – doubles the height of the text
- **double_width** (*bool*) – doubles the width of the text
- **custom_size** (*bool*) – uses custom size specified by width and height parameters. Cannot be used with double_width or double_height.
- **width** (*int*) – text width multiplier when custom_size is used, decimal range 1-8, *default*: 1
- **height** (*int*) – text height multiplier when custom_size is used, decimal range 1-8, *default*: 1
- **density** (*int*) – print density, value from 0-8, if something else is supplied the density remains unchanged
- **invert** (*bool*) – True enables white on black printing, *default*: False
- **smooth** (*bool*) – True enables text smoothing. Effective on 4x4 size text and larger, *default*: False
- **flip** (*bool*) – True enables upside-down printing, *default*: False

line_spacing(*spacing=None*, *divisor=180*)

Set line character spacing.

If no spacing is given, we reset it to the default.

There are different commands for setting the line spacing, using a different denominator:

'+" line_spacing/360 of an inch, 0 <= line_spacing <= 255 '3' line_spacing/180 of an inch, 0 <= line_spacing <= 255 'A' line_spacing/60 of an inch, 0 <= line_spacing <= 85

Some printers may not support all of them. The most commonly available command (using a divisor of 180) is chosen.

cut(*mode=u'FULL'*, *feed=True*)

Cut paper.

Without any arguments the paper will be cut completely. With 'mode=PART' a partial cut will be attempted. Note however, that not all models can do a partial cut. See the documentation of your printer for details.

Todo: Check this function on TM-T88II.

Parameters

- **mode** – set to 'PART' for a partial cut. default: 'FULL'
- **feed** – print and feed before cutting. default: true

Raises **ValueError** – if mode not in ('FULL', 'PART')

cashdraw(*pin*)
Send pulse to kick the cash drawer

Kick cash drawer on pin 2 or pin 5 according to default parameter. For non default parameter send a decimal sequence i.e. [27,112,48] or [27,112,0,25,255]

Parameters **pin** – pin number, 2 or 5 or list of decimals

Raises *CashDrawerError*

linedisplay_select(*select_display=False*)
Selects the line display or the printer

This method is used for line displays that are daisy-chained between your computer and printer. If you set *select_display* to true, only the display is selected and if you set it to false, only the printer is selected.

Parameters **select_display** (*bool*) – whether the display should be selected or the printer

linedisplay_clear()
Clears the line display and resets the cursor

This method is used for line displays that are daisy-chained between your computer and printer.

linedisplay(*text*)
Display text on a line display connected to your printer

You should connect a line display to your printer. You can do this by daisy-chaining the display between your computer and printer.

Parameters **text** – Text to display

hw(*hw*)
Hardware operations

Parameters **hw** – hardware action, may be:

- INIT
- SELECT
- RESET

print_and_feed(*n=1*)
Print data in print buffer and feed *n* lines

if n not in range (0, 255) then ValueError will be raised

Parameters **n** – number of n to feed. 0 <= n <= 255. default: 1

Raises **ValueError** – if not 0 <= n <= 255

control(*ctl*, *count=5*, *tab_size=8*)
Feed control sequences

Parameters

- **ctl** – string for the following control sequences:
  - LF *for Line Feed*
  - FF *for Form Feed*
  - CR *for Carriage Return*
  - HT *for Horizontal Tab*
  - VT *for Vertical Tab*

- **count** – integer between 1 and 32, controls the horizontal tab count. Defaults to 5.
- **tab_size** – integer between 1 and 255, controls the horizontal tab size in characters. Defaults to 8

Raises *TabPosError*

panel_buttons(*enable=True*)
Controls the panel buttons on the printer (e.g. FEED)

When enable is set to False the panel buttons on the printer will be disabled. Calling the method with enable=True or without argument will enable the panel buttons.

If panel buttons are enabled, the function of the panel button, such as feeding, will be executed upon pressing the button. If the panel buttons are disabled, pressing them will not have any effect.

This command is effective until the printer is initialized, reset or power-cycled. The default is enabled panel buttons.

Some panel buttons will always work, especially when printer is opened. See for more information the manual of your printer and the escpos-command-reference.

Parameters **enable** – controls the panel buttons

Return type None

query_status(*mode*)
Queries the printer for its status, and returns an array of integers containing it.

Parameters **mode** – Integer that sets the status mode queried to the printer. - RT_STATUS_ONLINE: Printer status. - RT_STATUS_PAPER: Paper sensor.

Return type array(integer)

is_online()
Queries the online status of the printer.

Returns When online, returns `True`; `False` otherwise.

Return type bool

paper_status()
Queries the paper status of the printer.

Returns 2 if there is plenty of paper, 1 if the paper has arrived to the near-end sensor and 0 if there is no paper.

Returns 2: Paper is adequate. 1: Paper ending. 0: No paper.

Return type int

class escpos.escpos.**EscposIO**(*printer*, *autocut=True*, *autoclose=True*, ***kwargs*)
Bases: `object`

ESC/POS Printer IO object

Allows the class to be used together with the *with*-statement. You have to define a printer instance and assign it to the EscposIO class. This example explains the usage:

```
with EscposIO(printer.Serial('/dev/ttyUSB0')) as p:
    p.set(font='a', height=2, align='center', text_type='bold')
    p.printer.set(align='left')
    p.printer.image('logo.gif')
    p.writelines('Big line\n', font='b')
```

(continues on next page)

(continued from previous page)

```
    p.writelines('')
    p.writelines('BIG TEXT', width=2)
```

After the *with*-statement the printer automatically cuts the paper if *autocut* is *True*.

set(***kwargs*)
Set the printer-parameters

Controls which parameters will be passed to *Escpos.set()*. For more information on the parameters see the *set()*-methods documentation. These parameters can also be passed with this class' constructor or the *writelines()*-method.

Parameters **kwargs** – keyword-parameters that will be passed to *Escpos.set()*

writelines(*text*, ***kwargs*)

close()
called upon closing the *with*-statement

6.11 Printer implementations

Module *escpos.printer*

This module contains the implementations of abstract base class Escpos.

author Manuel F Martinez and others

organization Bashlinux and python-escpos

copyright Copyright (c) 2012-2017 Bashlinux and python-escpos

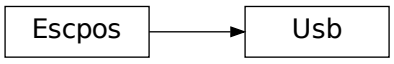
license MIT

class escpos.printer.**Usb**(*idVendor*, *idProduct*, *usb_args=None*, *timeout=0*, *in_ep=130*, *out_ep=1*, **args*, ***kwargs*)
Bases: *escpos.escpos.Escpos*

USB printer

This class describes a printer that natively speaks USB.

inheritance:



open(*usb_args*)
Search device on USB tree and set it as escpos device.

Parameters **usb_args** – USB arguments

close()
Release USB interface

class escpos.printer.**Serial**(*devfile=u'/dev/ttyS0'*, *baudrate=9600*, *bytesize=8*, *timeout=1*, *parity='N'*, *stopbits=1*, *xonxoff=False*, *dsrdtr=True*, **args*, ***kwargs*)

Bases: *escpos.escpos.Escpos*

Serial printer

This class describes a printer that is connected by serial interface.

inheritance:

open()

Setup serial port and set is as escpos device

close()

Close Serial interface

class escpos.printer.**Network**(*host*, *port=9100*, *timeout=60*, **args*, ***kwargs*)

Bases: *escpos.escpos.Escpos*

Network printer

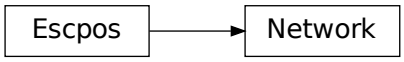
This class is used to attach to a networked printer. You can also use this in order to attach to a printer that is forwarded with `socat`.

If you have a local printer on parallel port `/dev/usb/lp0` then you could start `socat` with:

```
socat -u TCP4-LISTEN:4242,reuseaddr,fork OPEN:/dev/usb/lp0
```

Then you should be able to attach to port `4242` with this class. Otherwise the normal usecase would be to have a printer with ethernet interface. This type of printer should work the same with this class. For the address of the printer check its manuals.

inheritance:



open()

Open TCP socket with `socket`-library and set it as escpos device

close()

Close TCP connection

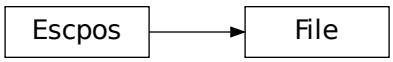
class escpos.printer.**File**(*devfile=u'/dev/usb/lp0'*, *auto_flush=True*, **args*, ***kwargs*)

Bases: *escpos.escpos.Escpos*

Generic file printer

This class is used for parallel port printer or other printers that are directly attached to the filesystem. Note that you should stay away from using USB-to-Parallel-Adapter since they are unreliable and produce arbitrary errors.

inheritance:



open()
Open system file

flush()
Flush printing content

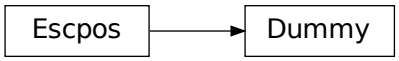
close()
Close system file

class escpos.printer.**Dummy**(**args*, ***kwargs*)
Bases: *escpos.escpos.Escpos*

Dummy printer

This class is used for saving commands to a variable, for use in situations where there is no need to send commands to an actual printer. This includes generating print jobs for later use, or testing output.

inheritance:



output
Get the data that was sent to this printer

clear()
Clear the buffer of the printer

This method can be called if you send the contents to a physical printer and want to use the Dummy printer for new output.

close()

6.12 Constants

Module *escpos.constants*

Set of ESC/POS Commands (Constants)

This module contains constants that are described in the esc/pos-documentation. Since there is no definitive and unified specification for all esc/pos-like printers the constants could later be moved to *capabilities* as in escpos-php by @mike42.

author Manuel F Martinez and others

organization Bashlinux and python-escpos

copyright Copyright (c) 2012-2017 Bashlinux and python-escpos

license MIT

escpos.constants.**CD_KICK_DEC_SEQUENCE**(*esc*, *p*, *m*, *t1=50*, *t2=50*)

escpos.constants.**SET_FONT**(*n*)

6.13 Exceptions

Module *escpos.exceptions*

ESC/POS Exceptions classes

Result/Exit codes:

- *0* = success
- *10* = No Barcode type defined *BarcodeTypeError*
- *20* = Barcode size values are out of range *BarcodeSizeError*
- *30* = Barcode text not supplied *BarcodeCodeError*
- *40* = Image height is too large *ImageSizeError*
- *41* = Image width is too large *ImageWidthError*
- *50* = No string supplied to be printed *TextError*
- *60* = Invalid pin to send Cash Drawer pulse *CashDrawerError*
- *70* = Invalid number of tab positions *TabPosError*
- *80* = Invalid char code *CharCodeError*
- *90* = USB device not found *USBNotFoundError*
- *100* = Set variable out of range *SetVariableError*
- *200* = Configuration not found *ConfigNotFoundError*
- *210* = Configuration syntax error *ConfigSyntaxError*
- *220* = Configuration section not found *ConfigSectionMissingError*

author Manuel F Martinez and others

organization Bashlinux and python-escpos

copyright Copyright (c) 2012-2017 Bashlinux and python-escpos

license MIT

exception escpos.exceptions.**Error**(*msg*, *status=None*)

Bases: exceptions.Exception

Base class for ESC/POS errors

exception escpos.exceptions.**BarcodeTypeError**(*msg=u''*)
Bases: *escpos.exceptions.Error*

No Barcode type defined.

This exception indicates that no known barcode-type has been entered. The barcode-type has to be one of those specified in *escpos.escpos.Escpos.barcode()*. The returned error code is *10*.

exception escpos.exceptions.**BarcodeSizeError**(*msg=u''*)
Bases: *escpos.exceptions.Error*

Barcode size is out of range.

This exception indicates that the values for the barcode size are out of range. The size of the barcode has to be in the range that is specified in *escpos.escpos.Escpos.barcode()*. The resulting returncode is *20*.

exception escpos.exceptions.**BarcodeCodeError**(*msg=u''*)
Bases: *escpos.exceptions.Error*

No Barcode code was supplied, or it is incorrect.

No data for the barcode has been supplied in *escpos.escpos.Escpos.barcode()* or the the *check* parameter was True and the check failed. The returncode for this exception is *30*.

exception escpos.exceptions.**ImageSizeError**(*msg=u''*)
Bases: *escpos.exceptions.Error*

Image height is longer than 255px and can't be printed.

The returncode for this exception is *40*.

exception escpos.exceptions.**ImageWidthError**(*msg=u''*)
Bases: *escpos.exceptions.Error*

Image width is too large.

The return code for this exception is *41*.

exception escpos.exceptions.**TextError**(*msg=u''*)
Bases: *escpos.exceptions.Error*

Text string must be supplied to the *text()* method.

This exception is raised when an empty string is passed to *escpos.escpos.Escpos.text()*. The returncode for this exception is *50*.

exception escpos.exceptions.**CashDrawerError**(*msg=u''*)
Bases: *escpos.exceptions.Error*

Valid pin must be set in order to send pulse.

A valid pin number has to be passed onto the method *escpos.escpos.Escpos.cashdraw()*. The returncode for this exception is *60*.

exception escpos.exceptions.**TabPosError**(*msg=u''*)
Bases: *escpos.exceptions.Error*

Valid tab positions must be set by using from 1 to 32 tabs, and between 1 and 255 tab size values. Both values multiplied must not exceed 255, since it is the maximum tab value.

This exception is raised by *escpos.escpos.Escpos.control()*. The returncode for this exception is *70*.

exception escpos.exceptions.**CharCodeError**(*msg=u''*)
Bases: *escpos.exceptions.Error*

Valid char code must be set.

The supplied charcode-name in *escpos.escpos.Escpos.charcode()* is unknown. Ths returncode for this exception is *80*.

exception escpos.exceptions.**USBNotFoundError**(*msg=u''*)
Bases: *escpos.exceptions.Error*

Device wasn't found (probably not plugged in)

The USB device seems to be not plugged in. Ths returncode for this exception is *90*.

exception escpos.exceptions.**SetVariableError**(*msg=u''*)
Bases: *escpos.exceptions.Error*

A set method variable was out of range

Check set variables against minimum and maximum values Ths returncode for this exception is *100*.

exception escpos.exceptions.**ConfigNotFoundError**(*msg=u''*)
Bases: *escpos.exceptions.Error*

The configuration file was not found

The default or passed configuration file could not be read Ths returncode for this exception is *200*.

exception escpos.exceptions.**ConfigSyntaxError**(*msg=u''*)
Bases: *escpos.exceptions.Error*

The configuration file is invalid

The syntax is incorrect Ths returncode for this exception is *210*.

exception escpos.exceptions.**ConfigSectionMissingError**(*msg=u''*)
Bases: *escpos.exceptions.Error*

The configuration file is missing a section

The part of the config asked for doesn't exist in the loaded configuration Ths returncode for this exception is *220*.

6.14 Capabilities

Module *escpos.capabilities*

exception escpos.capabilities.**NotSupported**
Bases: exceptions.Exception

Raised if a requested feature is not supported by the printer profile.

args

message

class escpos.capabilities.**BaseProfile**
Bases: object

This respresents a printer profile.

A printer profile knows about the number of columns, supported features, colors and more.

profile_data = {}

get_font(*font*)
Return the escpos index for *font*. Makes sure that the requested *font* is valid.

get_columns(*font*)
Return the number of columns for the given font.

supports(*feature*)
Return true/false for the given feature.

get_code_pages()
Return the support code pages as a `{name: index}` dict.

escpos.capabilities.**get_profile**(*name=None*, ***kwargs*)
Get the profile by name; if no name is given, return the default profile.

escpos.capabilities.**get_profile_class**(*name*)
For the given profile name, load the data from the external database, then generate dynamically a class.

escpos.capabilities.**clean**(*s*)

class escpos.capabilities.**Profile**(*columns=None*, *features=None*)
Bases: escpos.capabilities.DefaultProfile

For users, who want to provide their profile

get_code_pages()
Return the support code pages as a `{name: index}` dict.

get_font(*font*)
Return the escpos index for *font*. Makes sure that the requested *font* is valid.

profile_data = {'codePages': {'0': 'CP437', '1': 'CP932', '11': 'CP851', '12': 'C

supports(*feature*)
Return true/false for the given feature.

get_columns(*font*)
Return the number of columns for the given font.

6.15 Config

Module *escpos.config*

ESC/POS configuration manager.

This module contains the implentations of abstract base class *Config*.

class escpos.config.**Config**
Bases: object

Configuration handler class.

This class loads configuration from a default or specificed directory. It can create your defined printer and return it to you.

load(*config_path=None*)
Load and parse the configuration file using pyyaml

Parameters **config_path** – An optional file path, file handle, or byte string for the configuration file.

printer()
Returns a printer that was defined in the config, or throws an exception.

This method loads the default config if one hasn't beeen already loaded.

6.16 Image helper

Module *escpos.image*

Image format handling class

This module contains the image format handler *EscposImage*.

author Michael Billington

organization python-escpos

copyright Copyright (c) 2016 Michael Billington <michael.billington@gmail.com>

license MIT

class escpos.image.**EscposImage**(*img_source*)
Bases: object

Load images in, and output ESC/POS formats.

The class is designed to efficiently delegate image processing to PIL, rather than spend CPU cycles looping over pixels.

width
Width of image in pixels

width_bytes
Width of image if you use 8 pixels per byte and 0-pad at the end.

height
Height of image in pixels

to_column_format(*high_density_vertical=True*)
Extract slices of an image as equal-sized blobs of column-format data.

Parameters **high_density_vertical** – Printed line height in dots

to_raster_format()
Convert image to raster-format binary

split(*fragment_height*)
Split an image into multiple fragments after fragment_height pixels

Parameters **fragment_height** – height of fragment

Returns list of PIL objects

center(*max_width*)
In-place image centering

Param Maximum width in order to deduce x offset for centering

Returns None

6.17 CLI

Module *escpos.cli*

CLI

This module acts as a command line interface for python-escpos. It mirrors closely the available ESCPOS commands while adding a couple extra ones for convience.

It requires you to have a configuration file. See documentation for details.

escpos.cli.**str_to_bool**(*string*)
Used as a type in argparse so that we get back a proper bool instead of always True

escpos.cli.**main**()
Handles loading of configuration and creating and processing of command line arguments. Called when run from a CLI.

escpos.cli.**demo**(*printer*, ***kwargs*)
Prints specificed demos. Called when CLI is passed *demo*. This function uses the DEMO_FUNCTIONS dictionary.

Parameters

- **printer** – A printer from escpos.printer
- **kwargs** – A dict with a key for each function you want to test. It's in this format since it usually comes from argparse.

6.18 Magic Encode

Module *escpos.magicencode*

Magic Encode

This module tries to convert an UTF-8 string to an encoded string for the printer. It uses trial and error in order to guess the right codepage. The code is based on the encoding-code in py-xml-escpos by @fvdsn.

author Patrick Kanzler

organization python-escpos

copyright Copyright (c) 2016 Patrick Kanzler and Frédéric van der Essen

license MIT

class escpos.magicencode.**Encoder**(*codepage_map*)
Bases: object

Takes a list of available code spaces. Picks the right one for a given character.

Note: To determine the code page, it needs to do the conversion, and thus already knows what the final byte in the target encoding would be. Nevertheless, the API of this class doesn't return the byte.

The caller use to do the character conversion itself.

$ python -m timeit -s "{u'ö':'a'}.get(u'ö')" 100000000 loops, best of 3: 0.0133 usec per loop

$ python -m timeit -s "u'ö'.encode('latin1')" 100000000 loops, best of 3: 0.0141 usec per loop

get_sequence(*encoding*)

get_encoding_name(*encoding*)
Given an encoding provided by the user, will return a canonical encoding name; and also validate that the encoding is supported.

TODO: Support encoding aliases: pc437 instead of cp437.

can_encode(*encoding*, *char*)
Determine if a character is encodeable in the given code page.

Parameters

- **encoding** – The name of the encoding.
- **char** – The character to attempt to encode.

encode(*text*, *encoding*, *defaultchar=u'?'*)
Encode text under the given encoding

Parameters

- **text** – Text to encode
- **encoding** – Encoding name to use (must be defined in capabilities)
- **defaultchar** – Fallback for non-encodable characters

find_suitable_encoding(*char*)
The order of our search is a specific one:

1. code pages that we already tried before; there is a good chance they might work again, reducing the search space, and by re-using already used encodings we might also reduce the number of codepage change instructiosn we have to send. Still, any performance gains will presumably be fairly minor.
2. code pages in lower ESCPOS slots first. Presumably, they are more likely to be supported, so if a printer profile is missing or incomplete, we might increase our change that the code page we pick for this character is actually supported.

escpos.magicencode.**split_writable_text**(*encoder*, *text*, *encoding*)
Splits off as many characters from the begnning of text as are writable with "encoding". Returns a 2-tuple (writable, rest).

class escpos.magicencode.**MagicEncode**(*driver*, *encoding=None*, *disabled=False*, *defaultsymbol=u'?'*, *encoder=None*)
Bases: object

A helper that helps us to automatically switch to the right code page to encode any given Unicode character.

This will consider the printers supported codepages, according to the printer profile, and if a character cannot be encoded with the current profile, it will attempt to find a suitable one.

If the printer does not support a suitable code page, it can insert an error character.

force_encoding(*encoding*)
Sets a fixed encoding. The change is emitted right away.

From now one, this buffer will switch the code page anymore. However, it will still keep track of the current code page.

write(*text*)
Write the text, automatically switching encodings.

write_with_encoding(*encoding*, *text*)

6.19 Codepages

Module *escpos.codepages*

class escpos.codepages.**CodePageManager**(*data*)
Holds information about all the code pages (as defined in escpos-printer-db).

get_all()

static get_encoding_name(*encoding*)

get_encoding(*encoding*)

6.20 Katakana

Module *escpos.katakana*

Helpers to encode Japanese characters.

I doubt that this currently works correctly.

escpos.katakana.**encode_katakana**(*text*)
I don't think this quite works yet.

6.21 Indices and tables

- genindex
- modindex
- search

Python Module Index

e

escpos.capabilities, 56
escpos.cli, 58
escpos.codepages, 60
escpos.config, 57
escpos.constants, 53
escpos.escpos, 43
escpos.exceptions, 54
escpos.image, 58
escpos.katakana, 61
escpos.magicencode, 59
escpos.printer, 51

Index

A

args (*escpos.capabilities.NotSupported attribute*), 56

B

barcode() (*escpos.escpos.Escpos method*), 45
BarcodeCodeError, 55
BarcodeSizeError, 55
BarcodeTypeError, 54
BaseProfile (*class in escpos.capabilities*), 56
block_text() (*escpos.escpos.Escpos method*), 47

C

can_encode() (*escpos.magicencode.Encoder method*), 59
cashdraw() (*escpos.escpos.Escpos method*), 48
CashDrawerError, 55
CD_KICK_DEC_SEQUENCE() (*in module escpos.constants*), 54
center() (*escpos.image.EscposImage method*), 58
charcode() (*escpos.escpos.Escpos method*), 44
CharCodeError, 55
check_barcode() (*escpos.escpos.Escpos static method*), 45
clean() (*in module escpos.capabilities*), 57
clear() (*escpos.printer.Dummy method*), 53
close() (*escpos.escpos.EscposIO method*), 51
close() (*escpos.printer.Dummy method*), 53
close() (*escpos.printer.File method*), 53
close() (*escpos.printer.Network method*), 52
close() (*escpos.printer.Serial method*), 52
close() (*escpos.printer.Usb method*), 51
CodePageManager (*class in escpos.codepages*), 60
Config (*class in escpos.config*), 57
ConfigNotFoundError, 56
ConfigSectionMissingError, 56
ConfigSyntaxError, 56
control() (*escpos.escpos.Escpos method*), 49
cut() (*escpos.escpos.Escpos method*), 48

D

demo() (*in module escpos.cli*), 59
device (*escpos.escpos.Escpos attribute*), 44
Dummy (*class in escpos.printer*), 53

E

encode() (*escpos.magicencode.Encoder method*), 60
encode_katakana() (*in module escpos.katakana*), 61
Encoder (*class in escpos.magicencode*), 59
Error, 54
Escpos (*class in escpos.escpos*), 43
escpos.capabilities (*module*), 56
escpos.cli (*module*), 58
escpos.codepages (*module*), 60
escpos.config (*module*), 57
escpos.constants (*module*), 53
escpos.escpos (*module*), 43
escpos.exceptions (*module*), 54
escpos.image (*module*), 58
escpos.katakana (*module*), 61
escpos.magicencode (*module*), 59
escpos.printer (*module*), 51
EscposImage (*class in escpos.image*), 58
EscposIO (*class in escpos.escpos*), 50

F

File (*class in escpos.printer*), 52
find_suitable_encoding() (*escpos.magicencode.Encoder method*), 60
flush() (*escpos.printer.File method*), 53
force_encoding() (*escpos.magicencode.MagicEncode method*), 60

G

get_all() (*escpos.codepages.CodePageManager method*), 60

get_code_pages() (*escpos.capabilities.BaseProfile method*), 57
get_code_pages() (*escpos.capabilities.Profile method*), 57
get_columns() (*escpos.capabilities.BaseProfile method*), 56
get_columns() (*escpos.capabilities.Profile method*), 57
get_encoding() (*escpos.codepages.CodePageManager method*), 60
get_encoding_name() (*escpos.codepages.CodePageManager static method*), 60
get_encoding_name() (*escpos.magicencode.Encoder method*), 59
get_font() (*escpos.capabilities.BaseProfile method*), 56
get_font() (*escpos.capabilities.Profile method*), 57
get_profile() (*in module escpos.capabilities*), 57
get_profile_class() (*in module escpos.capabilities*), 57
get_sequence() (*escpos.magicencode.Encoder method*), 59

H

height (*escpos.image.EscposImage attribute*), 58
hw() (*escpos.escpos.Escpos method*), 49

I

image() (*escpos.escpos.Escpos method*), 44
ImageSizeError, 55
ImageWidthError, 55
is_online() (*escpos.escpos.Escpos method*), 50

L

line_spacing() (*escpos.escpos.Escpos method*), 48
linedisplay() (*escpos.escpos.Escpos method*), 49
linedisplay_clear() (*escpos.escpos.Escpos method*), 49
linedisplay_select() (*escpos.escpos.Escpos method*), 49
ln() (*escpos.escpos.Escpos method*), 47
load() (*escpos.config.Config method*), 57

M

MagicEncode (*class in escpos.magicencode*), 60
main() (*in module escpos.cli*), 59
message (*escpos.capabilities.NotSupported attribute*), 56

N

Network (*class in escpos.printer*), 52
NotSupported, 56

O

open() (*escpos.printer.File method*), 53
open() (*escpos.printer.Network method*), 52
open() (*escpos.printer.Serial method*), 52
open() (*escpos.printer.Usb method*), 51
output (*escpos.printer.Dummy attribute*), 53

P

panel_buttons() (*escpos.escpos.Escpos method*), 50
paper_status() (*escpos.escpos.Escpos method*), 50
print_and_feed() (*escpos.escpos.Escpos method*), 49
printer() (*escpos.config.Config method*), 57
Profile (*class in escpos.capabilities*), 57
profile_data (*escpos.capabilities.BaseProfile attribute*), 56
profile_data (*escpos.capabilities.Profile attribute*), 57

Q

qr() (*escpos.escpos.Escpos method*), 44
query_status() (*escpos.escpos.Escpos method*), 50

S

Serial (*class in escpos.printer*), 51
set() (*escpos.escpos.Escpos method*), 47
set() (*escpos.escpos.EscposIO method*), 51
SET_FONT() (*in module escpos.constants*), 54
SetVariableError, 56
soft_barcode() (*escpos.escpos.Escpos method*), 47
split() (*escpos.image.EscposImage method*), 58
split_writable_text() (*in module escpos.magicencode*), 60
str_to_bool() (*in module escpos.cli*), 59
supports() (*escpos.capabilities.BaseProfile method*), 57
supports() (*escpos.capabilities.Profile method*), 57

T

TabPosError, 55
text() (*escpos.escpos.Escpos method*), 47
TextError, 55
textln() (*escpos.escpos.Escpos method*), 47
to_column_format() (*escpos.image.EscposImage method*), 58
to_raster_format() (*escpos.image.EscposImage method*), 58

U

Usb (*class in escpos.printer*), 51

USBNotFoundError, 56

W

width (*escpos.image.EscposImage attribute*), 58
width_bytes (*escpos.image.EscposImage attribute*), 58
write() (*escpos.magicencode.MagicEncode method*), 60
write_with_encoding() (*escpos.magicencode.MagicEncode method*), 60
writelines() (*escpos.escpos.EscposIO method*), 51