python-escpos Documentation

Release 2.0.1.dev0+ngc13a071.d20161207

Manuel F Martinez and others

December 07, 2016

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.

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

CHAPTER 3

Documentation and Usage

The basic usage is:

```
from escpos import *

""" Seiko Epson Corp. Receipt Printer M129 Definitions (EPSON TM-T88IV) """
Epson = escpos.Escpos(0x04b8,0x0202,0)
Epson.text("Hello World\n")
Epson.image("logo.gif")
Epson.barcode('1324354657687','EAN13',64,2,'','')
Epson.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

Content

5.1 Dependencies

5.1.1 Fedora

Fortunately everything is on Fedora repositories.

```
# yum install python-imaging pyserial pyusb python-qrcode
```

5.1.2 Ubuntu

Ultimately, this instructions also apply to Raspbian, in case you are interested to install python-escpos on your Raspberry with Raspbian.

Install the packages available on distro repositories.

```
# apt-get install python-imaging pyserial
```

The packages which are not available at Ubuntu repositories need to be installed manually.

pyusb

This is the python binding to libusb-1.0

- Get the latest tarball from sourceforge
- Build and install it

```
# tar zxvf pyusb-1.*.tar.gz
# cd pyusb-1.*
# python setup.py build
# sudo python setup.py install
```

python-qrcode

This is the python module to generate QR Codes

- Checkout the latest code from github
- Build and install it

```
# git clone https://github.com/lincolnloop/python-qrcode
# cd python-qrcode
# python setup.py build
# sudo python setup.py install
```

5.2 Installation

5.2.1 System preparation

1. Install the required dependencies
2. Get the *Product ID* and *Vendor ID* from the lsusb command `# lsusb Bus 002 Device 001: ID 1a2b:1a2b Device name`
3. 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.
4. Restart udev `# sudo service udev restart` In some new systems it is done with `# sudo udevadm control --reload`

5.2.2 Install

- Clone python-escpos from github
- Change directory to python-escpos and install the package

```
# cd python-escpos
# python setup.py build
# sudo python setup.py install
```

- Enjoy !!!

5.3 Methods

Note: **TODO** Merge this page with the API-description. (Make the API-description more pretty and then replace this with the API-description.)

5.3.1 Escpos class

Escpos inherits its methods to the printers. the following methods are defined:

image("image_name.ext")

Prints an image. Its adjust the size in order to print it.

- `image_name.ext` is the complete file name and location of any image type (jpg, gif, png, bmp)

Raises `ImageSizeError` exception.

qr("text")

Prints a QR code. The size has been adjusted to Version 4, so it can be enough small to be printed but also enough big to be read by a smart phone.

- `text` Any text that needs to be QR encoded. It could be a slogan, salutation, url, etc.

barcode("code", "barcode_type", width, height, "position", "font")

Prints a barcode.

- `code` is an alphanumeric code to be printed as bar code
- `barcode_type` must be one of the following type of codes for function type A:
  - UPC-A
  - UPC-E
  - EAN13
  - EAN8
  - CODE39
  - ITF
  - NW7

  And for function type B:
  - Any type above
  - CODE93
  - CODE128
  - GS1-128
  - GS1 DataBar Omnidirectional
  - GS1 DataBar Truncated
  - GS1 DataBar Limited
  - GS1 DataBar Expanded
- `width` is a numeric value in the range between (1,255) *Default:* 64
- `height` is a numeric value in the range between (2,6) *Default:* 3
- `position` is where to place the code around the bars, could be one of the following values:
  - ABOVE
  - BELOW
  - BOTH

  - OFF > *Default:* BELOW
- `font` is one of the 2 type of fonts, values could be:
  - A
  - B > *Default:* A
- `fuction_type` chooses between ESCPOS function type A or B. A is default, B has more barcode options. Choose which one based upon your printer support and require barcode.
- A
- B > *Default* A
- Raises `BarcodeTypeError`, `BarcodeSizeError`, `BarcodeCodeError` exceptions.

text("text")

Prints raw text. Raises `TextError` exception.

set("align", "font", "type", width, height, invert, smooth, flip)

Set text properties. * `align` set horizontal position for text, the possible values are:

- CENTER
- LEFT
- RIGHT > > *Default:* left
- `font` type could be `A` or `B`. *Default:* A
- `type` type could be `B` (Bold), `U` (Underline) or `normal`. *Default:* normal
- `width` is a numeric value, 1 is for regular size, and 2 is twice the standard size. *Default*: 1
- `height` is a numeric value, 1 is for regular size and 2 is twice the standard size. *Default*: 1
- `invert` is a boolean value, True enables white on black printing. *Default*: False
- `smooth` is a boolean value, True enables text smoothing. *Default*: False
- `flip` is a boolean value, True enables upside-down text. *Default*: False

cut("mode")

Cut paper. * `mode` set a full or partial cut. *Default:* full

Partial cut is not implemented in all printers.

cashdraw(pin)

Sends a pulse to the cash drawer in the specified pin.

- `pin` is a numeric value which defines the pin to be used to send the pulse, it could be 2 or 5. Raises `CashDrawerError()`

hw("operation")

Hardware operations.

- `operation` is any of the following options:
  - INIT
  - SELECT
  - RESET

control("align")

Carrier feed and tabs. * `align` is a string which takes any of the following values:

- LF *for Line Feed*
- FF *for Form Feed*
- CR *for Carriage Return*
- HT *for Horizontal Tab*
- VT *for Vertical Tab*

5.4 Printers

Note: **TODO** Merge this page into the API-description.

There 3 different type of printers:

5.4.1 USB(idVendor, idProduct, interface, in_ep, out_ep)

Based on pyusb and libusb-1.0

- `idVendor` is the Vendor ID
- `idProduct` is the Product ID
- `interface` is the USB device interface (default = 0)
- `in_ep` is the input end point (default = 0x82)
- `out_ep` is the output end point (default = 0x01)

5.4.2 Serial("devfile", baudrate, bytesize, timeout)

Based on pyserial, default values are based on the defaults set by DIP_SWITCH_1 on the documentation(hardware side).

- `devfile` is an alphanumeric device file name under /dev filesystem (default = /ev/ttyS0)
- `baudrate` is the Baud rate for serial transmission (default = 9600)
- `bytesize` sets the serial buffer size (default = 8)

- `timeout` defines Read/Write timeout (default = 1)

5.4.3 Network("host", port)

Based on socket * `host` is an alphanumeric host name, could be either DNS host name or IP address. * `port` to write to (default = 9100)

5.4.4 File("file_name")

Printcap printers * `file_name` is the full path to the device file name

5.5 Raspberry Pi

This instructions were tested on Raspbian.

Unless you have done any distro with libusb-1.0 on the Raspberry Pi, the following instructions should works fine on your raspberry distro.

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*.

5.5.1 Dependencies

First, install the packages available on Raspbian.

```
# apt-get install python-imaging python-serial python-setuptools
```

PyUSB

PyUSB 1.0 is not available on Ubuntu, so you have to download and install it manually

1. Download the latest tarball from Sourceforge
2. Decompress the zip file
3. Install the library

```
# wget ...
# unzip pyusb*.zip
# cd pyusb*
# python setup.py build
# sudo python setup.py install
```

python-qrcode

1. Checkout the code from github
2. Install the library

```
    # git clone https://github.com/lincolnloop/python-qrcode
    # cd python-qrcode
    # python setup.py build
    # sudo python setup.py install
```

5.5.2 Installation

If you have installed pyusb for libusb-1.0 then you need to:

1. Download the latest file
2. Decompress the file
3. Install the library

```
# git clone https://github.com/manpaz/python-escpos.git
# cd python-escpos
# python setup.py build
# sudo python setup.py install
```

Now you can attach your printer and and test it with the example code in the project's home

5.6 TODO

5.6.1 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.

5.6.2 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 check barcode codes. Alternatively or as an addition write explanations about each barcode-type. Research whether the check digits can be computed autmatically.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/python-escpos/checkouts/v2.0.0/src/escpos/escpos.py:docstring of escpos.escpos.Escpos.barcode, line 8.)

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/checkouts/v2.0.0/src/escpos/escpos.py:docstring of escpos.escpos.Escpos.barcode, line 15.)

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/checkouts/v2.0.0/src/escpos/escpos.py:docstring of escpos.escpos.Escpos.barcode, line 16.)

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/checkouts/v2.0.0/src/escpos/escpos.py:docstring of escpos.escpos.Escpos.barcode, line 23.)

Todo

Check this function on TM-T88II.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/python-escpos/checkouts/v2.0.0/src/escpos/escpos.py:docstring of escpos.escpos.Escpos.cut, line 7.)

5.7 Usage

5.7.1 Define your printer

USB printer

Before start creating your Python ESC/POS printer instance, you must see at your system for the printer parameters. This is done with the 'lsusb' command.

First run the command to look for the "Vendor ID" and "Product ID", then 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 yields the "Interface" number that must be handy to have and the second yields the "Output Endpoint" address.

USB Printer initialization

```
Epson = 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 with your instance. So, assuming that we have another printer where in_ep is on 0x81 and out_ep=0x02, then the printer definition should looks like:

Generic USB Printer initialization

```
Generic = 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

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

Serial printer

Must 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 hooked up.

Serial printer initialization

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

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 the initialize passing the device node name.

```
Epson = 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.

5.7.2 Define your instance

The following example demonstrate how to initialize the Epson TM-TI88IV on USB interface

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

5.7.3 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 programically.

The default configuration file is named `config.yaml`. It's in 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 search to the `load()` method.

To load the configured pritner, 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 should be one of the printers defined in Printers.

The rest of the parameters are whatever you want to pass to the printer.

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
```

5.7.4 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

5.7.5 How to update your code for USB printers

Old code

```
Epson = escpos.Escpos(0x04b8,0x0202,0)
```

New code

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

Nothe that "0" which is the interface number is no longer needed.

5.8 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.

5.8.1 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

This is not yet consequently done in every piece of code, but please try to ensure that your code honors PEP8. The checks by Landscape and QuantifiedCode that run on every PR 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 a released version of the package.

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!

5.9 Changelog

5.9.1 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

5.9.2 2016-01-24 - Version 1.0.9

- fix constant definition for PC1252
- move documentation to Sphinx

5.9.3 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

5.9.4 2015-08-22 - Version 1.0.7

- Issue #57: Fixed transparent images

5.9.5 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>

5.9.6 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>

5.9.7 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

5.9.8 2014-02-23 - Version 1.0.3

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

5.9.9 2013-12-30 - Version 1.0.2

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

5.9.10 2013-03-14 - Version 1.0.1

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

5.9.11 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

5.10 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 Bashlinux

license GNU GPL v3

class escpos.escpos.**Escpos**(*columns=32*)

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

codepage = None

image(*img_source*, *high_density_vertical=True*, *high_density_horizontal=True*, *impl=u'bitImageRaster'*)

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*

qr(*content*, *ec=0*, *size=3*, *model=2*, *native=False*)

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)

charcode(*code*)

Set Character Code Table

Sends the control sequence from *escpos.constants* to the printer with `escpos.printer.'implementation'._raw()`.

Parameters **code** – Name of CharCode

Raises *CharCodeError*

barcode(*code*, *bc*, *height=64*, *width=3*, *pos=u'BELOW'*, *font=u'A'*, *align_ct=True*, *function_type=u'A'*)

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. Currently you have to check manually whether your barcode text is correct. 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.

Todo

Add a method to check barcode codes. Alternatively or as an addition write explanations about each barcode-type. Research whether the check digits can be computed autmatically.

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. *default*: A

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*

block_text(*txt*, *columns=None*)
Text is printed wrapped to specified columns

Text has to be encoded in unicode.

Parameters

- **txt** – text to be printed
- **columns** – amount of columns

Returns None

set(*align=u'left'*, *font=u'a'*, *text_type=u'normal'*, *width=1*, *height=1*, *density=9*, *invert=False*, *smooth=False*, *flip=False*)
Set text properties by sending them to the printer

Parameters

- **align** – horizontal position for text, possible values are:
  - CENTER
  - LEFT
  - RIGHT

  default: LEFT
- **font** – font type, possible values are A or B, *default*: A
- **text_type** – text type, possible values are:
  - B for bold

  - U for underlined
  - U2 for underlined, version 2
  - BU for bold and underlined
  - BU2 for bold and underlined, version 2
  - NORMAL for normal text

  default: NORMAL
- **width** – text width multiplier, decimal range 1-8, *default*: 1
- **height** – text height multiplier, decimal range 1-8, *default*: 1
- **density** – 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** – True enables text smoothing. Effective on 4x4 size text and larger, *default*: False
- **flip** – True enables upside-down printing, *default*: False

cut(*mode=u''*)
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

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

Kick cash drawer on pin 2 or pin 5 according to parameter.

Parameters **pin** – pin number, 2 or 5

Raises *CashDrawerError*

hw(*hw*)
Hardware operations

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

- INIT
- SELECT
- RESET

control(*ctl*, *pos=4*)
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*

- **pos** – integer between 1 and 16, controls the horizontal tab position

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

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 EsposIO-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')
    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

5.11 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 Bashlinux

license GNU GPL v3

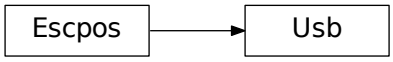
class `escpos.printer.`**`Usb`**(*idVendor*, *idProduct*, *interface=0*, *in_ep=130*, *out_ep=1*, **args*, ***kwargs*)

Bases: *escpos.escpos.Escpos*

USB printer

This class describes a printer that natively speaks USB.

inheritance:



`open`()

Search device on USB tree and set it as escpos device

`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

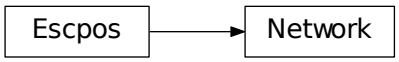
This class is used to attach to a networked printer. You can also use this in order to attach to a printer that is forwarded with `socat`.

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

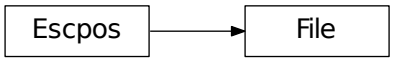
close()
Close TCP connection

class escpos.printer.**File**(*devfile=u'/dev/usb/lp0'*, **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

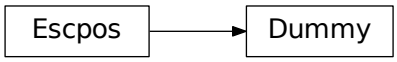
class escpos.printer.**Dummy**(**args*, ***kwargs*)

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

close()

5.12 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 Bashlinux

license GNU GPL v3

5.13 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*
- *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 Bashlinux

license GNU GPL v3

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.

No data for the barcode has been supplied in *escpos.escpos.Escpos.barcode()*. 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.**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 in the range 0 to 16.

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*.

5.14 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.

5.15 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 GNU GPL v3

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

5.16 Indices and tables

- genindex
- modindex
- search

e

T

U

W