Simple ZPL2 Documentation

Release 0.1.0

Joe Sacher

May 27, 2017

Contents

Contents:

CHAPTER 1

Simple ZPL2

Simple Project to help in building ZPL2 strings for printing barcodes with Zebra or compatible label printers.

- Free software: MIT license
- Documentation: https://simple-zpl2.readthedocs.io.

Features

- Methods for adding ZPL2 entries in the label data
- Error handling for data entered into methods, to maintain valid ZPL data
- Using web service to render ZPL2 label as PNG for quick development
- Simple class to print to network based ZPL label printer

Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

Stable release

To install Simple ZPL2, run this command in your terminal:

```
$ pip install simple_zpl2
```

This is the preferred method to install Simple ZPL2, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

From sources

The sources for Simple ZPL2 can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/sacherjj/simple_zpl2
```

Or download the tarball:

```
$ curl  -OL https://github.com/sacherjj/simple_zpl2/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use Simple ZPL2 in a project:

```
from simple_zpl2 import Formatter

# Each label is built with a Formatter object
# The below code would generally be built into a method to create the object or both␣
↪create and print.
zpl = Formatter()
zpl.add_field_origin(20, 20)
zpl.add_barcode_qr(1, 2, zpl.QR_ERROR_CORRECTION_STANDARD)
zpl.add_field_data('This is data inside a QR code.  This is a barcode often read by␣
↪cell phones.')
```

You can view the zpl encoded text:

```
print(zpl.zpl_text)
```

Using a web service to render the label as PNG:

```
from PIL import Image
import io

# Get PNG byte array
png = zpl.render_png(label_width=2, label_height=1)
# render fake file from bytes
fake_file = io.BytesIO(png)
img = Image.open(fake_file)
# Open image with the default image viewer on your system
img.show()
```

Print label to network based label printer:

```
from simple_zpl2 import NetworkPrinter

```

```
prn = NetworkPrinter('192.168.40.1')
prn.print_zpl(zpl)
```

CHAPTER 4

API Documentation

Formatter

class simple_zpl2.formatter.**Formatter**

Bases: object

Builds ZPL II label data based on methods called and data passed.

Note: Dots to real measurements based on printer dpi:

- 150 dpi: 6 dots = 1 mm, 152 dots = 1 in,
- 200 dpi: 8 dots = 1 mm, 203 dots = 1 in,
- 300 dpi: 12 dots = 1 mm, 300 dots = 1 in,
- 600 dpi: 24 dots = 1mm, 600 dots = 1 in

END = '^XZ'

Ending block in ZPL2 document. Added automatically

JUSTIFICATION_AUTO = 2

JUSTIFICATION_LEFT = 0

JUSTIFICATION_RIGHT = 1

ORIENTATION_180 = 'I'

ORIENTATION_270 = 'B'

ORIENTATION_90 = 'R'

ORIENTATION_NORMAL = 'N'

QR_ERROR_CORRECTION_HIGH = 'Q'

QR_ERROR_CORRECTION_LOW = 'H'

QR_ERROR_CORRECTION_STANDARD = 'M'

QR_ERROR_CORRECTION_ULTRA_HIGH = 'H'

START = '^XA'
Starting block in ZPL2 document. Added automatically

TEXT_JUSTIFICATION_CENTER = 'C'

TEXT_JUSTIFICATION_JUSTIFIED = 'J'

TEXT_JUSTIFICATION_LEFT = 'L'

TEXT_JUSTIFICATION_RIGHT = 'R'

add_barcode_ansi_codabar(**args*, ***kwargs*)
ANSI Codabar Bar Code (^BK)

Characters to encode (0-9)

Parameters

- **orientation** – 'N' - normal, 'R' - rotate 90, 'I' - inverted, 'B' - rotate 270
- **height** – bar code height in dots (1 to 32000)
- **print_text** – print text of data ('Y', 'N')
- **text_above** – print text above barcode ('Y', 'N')
- **start_character** – 'A', 'B', 'C', 'D'
- **stop_character** – 'A', 'B', 'C', 'D'

add_barcode_aztec(**args*, ***kwargs*)
Aztec Barcode (^B0 [zero] or ^BO [letter])

Parameters

- **orientation** –
  - 'N' - normal
  - 'R' - rotate 90
  - 'I' - inverted
  - 'B' - rotate 270
- **magnification** – 1 to 10
- **ecic** –
  - 'Y' - data contains ECICs
  - 'N' - does not contain ECICs
- **ec_symbol_size** –
  - 0 - default error correction
  - 01-99 - error correction percentage
  - 101-104 - 1-4 layer compact symbol
  - 201-232 - 1-32 layer full-range symbol
  - 300 - simple Aztec "Rune"
- **menu_symbol** –

- 'Y' - a menu or barcode reader initialization sybmol
- 'N' - not menu symbol

- **number_of_symbols** – Structured append 1-26 sybmols
- **structured_id_append** – up to 24 character ID data

add_barcode_codablock(**args*, ***kwargs*)
CODABLOCK Bar Code (^BB)

Parameters

- **orientation** –
  - 'N' - normal
  - 'R' - rotate 90
  - 'I' - inverted
  - 'B' - rotate 270
- **height** – height of individual dots (2 to 32000)
- **security_level** – ('Y', 'N') only 'N' if mode is 'A'
- **characters_per_row** – 2-62
- **row_count** – mode A: 1-22, mode E,F: 2-4
- **mode** –
  - 'A' - Code 39
  - 'F' - Code 128
  - 'E' - Code 128 with FNC1

add_barcode_code_11(**args*, ***kwargs*)
Code 11 Bar Code (^B1)

Characters to encode (0-9 and -)

Parameters

- **orientation** –
  - 'N' - normal
  - 'R' - rotate 90
  - 'I' - inverted
  - 'B' - rotate 270
- **check_digit** –
  - 'Y' - 1 digit
  - 'N' - 2 digits
- **height** – bar code height in dots (1 to 32000)
- **print_text** – print text of data ('Y', 'N')
- **text_above** – print text above barcode ('Y', 'N')

add_barcode_code_128(**args*, ***kwargs*)
Code 128 Barcode (^BC)

Parameters

- **orientation** –
  - 'N' - normal
  - 'R' - rotate 90
  - 'I' - inverted
  - 'B' - rotate 270
- **height** – bar code height in dots (1 to 32000)
- **print_text** – print text of data ('Y', 'N')
- **text_above** – print text above barcode ('Y', 'N')
- **check_digit** – Add Mod10 check digit to Mod103 ('Y', 'N')

add_barcode_code_39(**args*, ***kwargs*)
Code 39 Bar Code (^B3)

Characters to encode (0-9, A-Z, -, ., $, /, +, %, ' ') If Scanner supports extended ASCII, must encode ^FD with +$ and -$ surrounding

Parameters

- **orientation** – 'N' - normal, 'R' - rotate 90, 'I' - inverted, 'B' - rotate 270
- **check_digit** – calculate and print Mod 43 check digit ('Y', 'N')
- **height** – bar code height in dots (1 to 32000)
- **print_text** – print text of data ('Y', 'N')
- **text_above** – print text above barcode ('Y', 'N')

add_barcode_code_49(**args*, ***kwargs*)
Code 49 Bar Code (^B4)

Parameters

- **orientation** – 'N' - normal, 'R' - rotate 90, 'I' - inverted, 'B' - rotate 270
- **height_multiplier** – 1 to height of label (recommending much more than 1)
- **print_text** – print text of data ('Y', 'N')
- **text_above** – print text above barcode ('Y', 'N')
- **starting_mode** – 0 - Regular Alphanumeric Mode 1 - Multiple Read Alphanumeric 2 - Regular Numeric Mode 3 - Group Alphanumeric Mode 4 - Regular Alphanumeric Shift 1 5 - Regular Alphanumeric Shift 2 A - Automatic Mode. The printer determines the starting mode by analyzing the field data.

add_barcode_code_93(**args*, ***kwargs*)
Code 93 Bar Code (^BA)

Parameters

- **orientation** – 'N' - normal, 'R' - rotate 90, 'I' - inverted, 'B' - rotate 270
- **height** – bar code height in dots (1 to 32000)
- **print_text** – print text of data ('Y', 'N')
- **text_above** – print text above barcode ('Y', 'N')

- **check_digit** – print check digit ('Y', 'N')

add_barcode_data_matrix(**args*, ***kwargs*)

Data Matrix Bar Code (^BX)

Parameters

- **orientation** –
  - 'N' - normal
  - 'R' - rotate 90
  - 'I' - inverted
  - 'B' - rotate 270
- **height** – height of individual symbol elements 1-width of label
- **quality** – amount of data added for error correction 0, 50, 80, 100, 140, 200
- **columns** –
  - columns to encode 9-49
  - odd values only for quality 0-140
  - even values for quality 200
- **rows** – rows to encode 9-49
- **format_id** –
  - 1 = field data is numeric + space (0..9,") – No &
  - 2 = field data is uppercase alphanumeric + space (A..Z,") – No &"
  - 3 = field data is uppercase alphanumeric + space, period, comma, dash, and slash (0..9,A..Z,".-/")
  - 4 = field data is upper-case alphanumeric + space (0..9,A..Z,") – no &"
  - 5 = field data is full 128 ASCII 7-bit set
  - 6 = field data is full 256 ISO 8-bit set
- **escape_sequence** – any character
- **aspect_ratio** –
  - 1 = square
  - 2 = rectangular

Effects of ^BY on ^BX

w = module width (no effect)

r = ratio (no effect)

h = height of symbol

If the dimensions of individual symbol elements are not specified in the ^BY command, the height of symbol value is divided by the required rows/columns, rounded, limited to a minimum value of one, and used as the dimensions of individual symbol elements.

Field Data (^FD) for ^BX

Quality 000 to 140

- The & and || can be used to insert carriage returns, line feeds, and the backslash, similar to the PDF417. Other characters in the control character range can be inserted only by using ^FH. Field data is limited to 596 characters for quality 0 to 140. Excess field data causes no symbol to print; if ^CV is active, INVALID-L prints. The field data must correspond to a user-specified format ID or no symbol prints; if ^CV is active, INVALID-C prints.
- The maximum field sizes for quality 0 to 140 symbols are shown in the tktable in the g parameter.

Quality 200

- If more than 3072 bytes are supplied as field data, it is truncated to 3072 bytes. This limits the maximum size of a numeric Data Matrix symbol to less than the 3116 numeric characters that the specification would allow. The maximum alphanumeric capacity is 2335 and the maximum 8-bit byte capacity is 1556.
- If ^FH is used, field hexadecimal processing takes place before the escape sequence processing described below.
- The underscore is the default escape sequence control character for quality 200 field data. A different escape sequence control character can be selected by using parameter g in the ^BX command.

The information that follows applies to firmware version: V60.13.0.12, V60.13.0.12Z, V60.13.0.12B, V60.13.0.12ZB, or later. The input string escape sequences can be embedded in quality 200 field data using the ASCII 95 underscore character (_) or the character entered in parameter g:

- _X is the shift character for control characters (e.g., _@=NUL,_G=BEL,_0 is PAD)
- _1 to _3 for FNC characters 1 to 3 (explicit FNC4, upper shift, is not allowed)
- FNC2 (Structured Append) must be followed by nine digits, composed of three-digit numbers with values between 1 and 254, that represent the symbol sequence and file identifier (for example, symbol 3 of 7 with file ID 1001 is represented by _2214001001)
- 5NNN is code page NNN where NNN is a three-digit code page value (for example, Code Page 9 is represented by _5009)
- _dNNN creates ASCII decimal value NNN for a code word (must be three digits)
- _ in data is encoded by __ (two underscores) The information that follows applies to all other versions of firmware. The input string escape sequences can be embedded in quality 200 field data using the ASCII 7E tilde character (~) or the character entered in parameter g:
- ~X is the shift character for control characters (e.g., ~@=NUL,~G=BEL,~0 is PAD)
- ~1 to ~3 for FNC characters 1 to 3 (explicit FNC4, upper shift, is not allowed)
- FNC2 (Structured Append) must be followed by nine digits, composed of three-digit numbers with values between 1 and 254, that represent the symbol sequence and file identifier (for example, symbol 3 of 7 with file ID 1001 is represented by ~2214001001)
- 5NNN is code page NNN where NNN is a three-digit code page value (for example, Code Page 9 is represented by ~5009)
- ~dNNN creates ASCII decimal value NNN for a code word (must be three digits)
- ~ in data is encoded by a ~ (tilde)

add_barcode_default(**args*, ***kwargs*)
Set defaults for bar codes (^BY)

Parameters

- **module_width** – 1 to 10 dots (default 2)
- **wide_narrow_ratio** – 2.0 to 3.0 is 0.1 increments (default 3.0)
- **height** – bar code height in dots (default 10)

add_barcode_ean_13(**args*, ***kwargs*)
EAN-13 Bar Code (^BE)

Following Field data is limited to exactly 12 characters.

Parameters

- **orientation** – 'N' - normal, 'R' - rotate 90, 'I' - inverted, 'B' - rotate 270
- **height** – bar code height in dots (1 to 32000)
- **print_text** – print text of data ('Y', 'N')
- **text_above** – print text above barcode ('Y', 'N')

add_barcode_ean_8(**args*, ***kwargs*)
EAN 8 Bar Code (^B8)

Parameters

- **orientation** – 'N' - normal, 'R' - rotate 90, 'I' - inverted, 'B' - rotate 270
- **height** – bar code height in dots (1 to 32000)
- **print_text** – print text of data ('Y', 'N')
- **text_above** – print text above barcode ('Y', 'N')

add_barcode_gs1_batabar(**args*, ***kwargs*)
GS1 Databar Bar Code (^BR)

Parameters

- **orientation** – 'N' - normal, 'R' - rotate 90, 'I' - inverted, 'B' - rotate 270
- **symbology_type** –

  1 = GS1 DataBar Omnidirectional 2 = GS1 DataBar Truncated 3 = GS1 DataBar Stacked 4 = GS1 DataBar Stacked Omnidirectional 5 = GS1 DataBar Limited 6 = GS1 DataBar Expanded 7 = UPC-A 8 = UPC-E 9 = EAN-13

  10 = EAN-8 11 = UCC/EAN-128 and CC-A/B 12 = UCC/EAN-128 and CC-C
- **magnification** – 1-10
- **separator_height** – 1 or 2
- **height** – bar code height 1-32000 dots
- **width** – 2 - 22 (even numbers only)

add_barcode_industrial_2_of_5(**args*, ***kwargs*)
Industrial 2 of 5 Bar Code (^BI)

Characters to encode (0-9)

Parameters

- **orientation** – 'N' - normal, 'R' - rotate 90, 'I' - inverted, 'B' - rotate 270
- **height** – bar code height in dots (1 to 32000)
- **print_text** – print text of data ('Y', 'N')
- **text_above** – print text above barcode ('Y', 'N')

add_barcode_interleaved_2_of_5(**args*, ***kwargs*)
Interleaved 2 of 5 Bar Code (^B2)

Characters to encode (0-9)

Parameters

- **orientation** –
  - 'N' - normal
  - 'R' - rotate 90
  - 'I' - inverted
  - 'B' - rotate 270
- **height** – bar code height in dots (1 to 32000)
- **print_text** – print text of data ('Y', 'N')
- **text_above** – print text above barcode ('Y', 'N')
- **check_digit** – calculate and print Mod 10 check digit ('Y', 'N')

add_barcode_logmars(**args*, ***kwargs*)
LOGMARS Bar Code (^BL)

Parameters

- **orientation** –
  - 'N' - normal
  - 'R' - rotate 90
  - 'I' - inverted
  - 'B' - rotate 270
- **height** – bar code height in dots (1 to 32000)
- **text_above** – print text above barcode ('Y', 'N')

add_barcode_micropdf_417(**args*, ***kwargs*)
MicroPDF417 Bar Code (^BF)

Parameters

- **orientation** – 'N' - normal, 'R' - rotate 90, 'I' - inverted, 'B' - rotate 270
- **height** – bar code height in dots (1 to 9999)
- **mode** – 0-33

To encode data into a MicroPDF417 bar code, complete these steps:

1. Determine the type of data to be encoded (for example, ASCII characters, numbers, 8-bit data, or a combination). 2. Determine the maximum amount of data to be encoded within the bar code (for example, number of ASCII characters, quantity of numbers, or quantity of 8-bit data characters). 3. Determine the percentage of check digits that are used within the

bar code. The higher the percentage of check digits that are used, the more resistant the bar code is to damage — however, the size of the bar code increases. 4. Use Table 10 with the information gathered from the questions above to select the mode of the bar code.

MO - mode DC - Number of Data Columns DR - Number of Data Rows EC - % of CWS for EC MX - Max Alpha Characters MD - Max Digits

MO DC DR EC MX MD 0 1 11 64 6 8 1 1 14 50 12 17 2 1 17 41 18 26 3 1 20 40 22 32 4 1 24 33 30 44 5 1 28 29 38 55 6 2 8 50 14 20 7 2 11 41 24 35 8 2 14 32 36 52 9 2 17 29 46 67

10 2 20 28 56 82 11 2 23 28 64 93 12 2 26 29 72 105 13 3 6 67 10 14 14 3 8 58 18 26 15 3 10 53 26 38 16 3 12 50 34 49 17 3 15 47 46 67 18 3 20 43 66 96 19 3 26 41 90 132 20 3 32 40 114 167 21 3 38 39 138 202 22 3 44 38 162 237 23 4 6 50 22 32 24 4 8 44 34 49 25 4 10 40 46 67 26 4 12 38 58 85 27 4 15 35 76 111 28 4 20 33 106 155 29 4 26 31 142 208 30 4 32 30 178 261 31 4 38 29 214 313 32 4 44 28 250 366 33 4 4 50 14 20

add_barcode_msi (**args*, ***kwargs*)
MSI Bar Code (^BM)

Characters to encode (0-9)

Parameters

- **orientation** – 'N' - normal, 'R' - rotate 90, 'I' - inverted, 'B' - rotate 270
- **check_digit** – A - no check digits B - 1 Mod 10 C - 2 Mod 10 D - 1 Mod 11 and 1 Mod 10
- **height** – bar code height in dots (1 to 32000)
- **print_text** – print text of data ('Y', 'N')
- **text_above** – print text above barcode ('Y', 'N')
- **insert_check_digit** – Add check digit to text line ('Y', 'N')

add_barcode_pdf417 (**args*, ***kwargs*)
PDF417 Bar Code (^B7)

Parameters

- **orientation** – 'N' - normal, 'R' - rotate 90, 'I' - inverted, 'B' - rotate 270
- **height** – height of individual dots, recommends larger than 1
- **security_level** – 0 - error detection only, 1-8 correction level
- **data_column_count** – number of code word columns (1-30)
- **row_count** – number of rows to encode (3-90)
- **truncate** – truncate right row indicators and stop pattern ('Y', 'N')

add_barcode_planet_code (**args*, ***kwargs*)
Planet Code Bar Code (^B5)

Parameters

- **orientation** – 'N' - normal, 'R' - rotate 90, 'I' - inverted, 'B' - rotate 270
- **height** – bar code height in dots (1 to 9999)
- **print_text** – print text of data ('Y', 'N')
- **text_above** – print text above barcode ('Y', 'N')

add_barcode_plessey(**args*, ***kwargs*)
Plessey Bar Code (^BP)

Characters to encode (0-9 A-F)

Parameters

- **orientation** – 'N' - normal, 'R' - rotate 90, 'I' - inverted, 'B' - rotate 270
- **check_digit** – print check digit ('Y', 'N')
- **height** – bar code height in dots (1 to 32000)
- **print_text** – print text of data ('Y', 'N')
- **text_above** – print text above barcode ('Y', 'N')

add_barcode_postal(**args*, ***kwargs*)
Postal Bar Code (^BZ)

Characters (0-9)

Parameters

- **orientation** –
  - 'N' - normal
  - 'R' - rotate 90
  - 'I' - inverted
  - 'B' - rotate 270
- **height** – bar code height in dots (1 to 32000)
- **print_text** – print text of data ('Y', 'N')
- **text_above** – print text above barcode ('Y', 'N')
- **code_type** –
  - 0 = Postnet bar code
  - 1 = Plant Bar Code
  - 2 = Reserved
  - 3 = USPS Intelligent Mail bar code

add_barcode_qr(**args*, ***kwargs*)
QR Barcode (^BQ)

Parameters

- **model** – 1 - original, 2 - enhanced
- **magnification** – 1 to 10
- **error_correction** – 'H' - ultra-high, 'Q' - high', 'M' - standard, 'L' - low
- **mask_value** – 0-7 defaults 7

QR Switches (formatted into the ^FD field data) There are 4 switch fields that are allowed, some with associated parameters and some without. Two of these fields are always present, one is optional, and one's presence depends on the value of another. The switches are always placed in a fixed order. The four switches, in order are:

Mixed mode <D>iijjxx,Optional (note that this switch ends with a comma ",") Error correction level <H, Q, M, L>Mandatory Data input <A, M>,Mandatory (note that this switch ends with a comma ",") Character Mode <N, A, Bdddd, K>Conditional (present if data input is M)

Mixed mode (Optional) = D - allows mixing of different types of character modes in one code. ii = code No. – a 2 digit number in the range 01 to 16 Value = subtracted from the Nth number of the divided code (must be two digits). jj = No. of divisions – a 2 digit number in the range 02 to 16 Number of divisions (must be two digits). xx = parity data – a 2 digit hexadecimal character in the range 00 to FF Parity data value is obtained by calculating at the input data (the original input data before divided byte-by-byte through the EX-OR operation). , = the mixed mode switch, when present, is terminated with a comma

Error correction level (Required) = H, Q, M, or L H = ultra-high reliability level Q = high reliability level M = standard level (default) L = high density level

Data input (Required) = A or M followed by a comma A = Automatic Input (default). Character Mode is not specified. Data character string JIS8 unit, Shift JIS. When the input mode is Automatic Input, the binary codes of 0x80 to 0x9F and 0xE0 to 0xFF cannot be set. M = Manual Input. Character Mode must be specified. Two types of data input mode exist: Automatic (A) and Manual (M). If A is specified, the character mode does not need to be specified. If M is specified, the character mode must be specified. Character Mode (Required when data input = M) = N, A, Bxxxx, or K N = numeric: digits 0 – 9 A = alphanumeric: digits 0 – 9, upper case letters A – Z, space, and $%*+-./:) (45 characters) Bxxxx = 8-bit byte mode. The 'xxxx' is the number of characters and must be exactly 4 decimal digits. This handles the 8-bit Latin/Kana character set in accordance with JIS X 0201 (character values 0x00 to 0xFF). K = Kanji — handles only Kanji characters in accordance with the Shift JIS system based on JIS X 0208. This means that all parameters after the character mode K should be 16-bit characters. If there are any 8-bit characters (such as ASCII code), an error occurs. The data to be encoded follows immediately after the last switch.

Considerations for ^FD When Using the QR Code:

QR Switches (formatted into the ^FD field data)

mixed mode <D> D = allows mixing of different types of character modes in one code. code No. <01 16> Value = subtracted from the Nth number of the divided code (must be two digits).

No. of divisions <02 16> Number of divisions (must be two digits).

parity data <1 byte> Parity data value is obtained by calculating at the input data (the original input data before divided byte-by-byte through the EX-OR operation).

error correction level <H, Q, M, L> H = ultra-high reliability level Q = high reliability level M = standard level (default) L = high density level

character Mode <N, A, B, K> N = numeric A = alphanumeric Bxxxx = 8-bit byte mode. This handles the 8-bit Latin/Kana character set in accordance with JIS X 0201 (character values 0x00 to 0xFF). xxxx = number of data characters is represented by two bytes of BCD code. K = Kanji — handles only Kanji characters in accordance with the Shift JIS system based on JIS X 0208. This means that all parameters after the character mode K should be 16-bit characters. If there are any 8-bit characters (such as ASCII code), an error occurs.

data character string <Data> Follows character mode or it is the last switch in the ^FD statement.

data input <A, M> A = Automatic Input (default). Data character string JIS8 unit, Shift JIS. When the input mode is Automatic Input, the binary codes of 0x80 to 0x9F and 0xE0 to 0xFF cannot be set. M = Manual Input Two types of data input mode exist: Automatic (A)

and Manual (M). If A is specified, the character mode does not need to be specified. If M is specified, the character mode must be specified.

^FD Field Data (Normal Mode) Automatic Data Input (A) with Switches ^FD <error correction level>A, <data character string> ^FS

Manual Data Input (M) with Switches ^FD <error correction level>M, <character mode><data character string> ^FS

^FD Field Data (Mixed Mode – requires more switches) Automatic Data Input (A) with Switches ^FD <D><code No.> <No. of divisions> <parity data>, <error correction level> A, <data character string>, <data character string>, < : >, <data character string n**> ^FS

Manual Data Input (M) with Switches ^FD <code No.> <No. of divisions> <parity data>, <error correction level> M, <character mode 1> <data character string 1>, <character mode 2> <data character string 2>, < : > < : >, <character mode n> <data character string n**> ^FS

n** up to 200 in mixed mode

add_barcode_standard_2_of_5 (**args*, ***kwargs*)
Standard 2 of 5 Bar Code (^BJ)

Characters to encode (0-9)

Parameters

- **orientation** – 'N' - normal, 'R' - rotate 90, 'I' - inverted, 'B' - rotate 270
- **height** – bar code height in dots (1 to 32000)
- **print_text** – print text of data ('Y', 'N')
- **text_above** – print text above barcode ('Y', 'N')

add_barcode_tlc39 (**args*, ***kwargs*)
TLC39 Bar Code (^BT)

Parameters

- **orientation** –
  - 'N' - normal
  - 'R' - rotate 90
  - 'I' - inverted
  - 'B' - rotate 270
- **code_39_width** – width of the Code 39 bar code 1-10
- **code_39_ratio** – wide to narrow bar width ratio of Code 39 bar code 2.0-3.0 by 0.1
- **code_39_height** – height of the Code 39 bar code 1-9999
- **micropdf417_height** – height of MicroPDF417 bar code 1-255
- **micropdf417_width** – width of MicroPDF417 bar code 1-10

Note: ECI Number.

If the seventh character is not a comma, only Code 39 prints. This means if more than 6 digits are present, Code 39 prints for the first six digits (and no Micro-PDF symbol is printed).

•Must be 6 digits.

•Firmware generates invalid character error if the firmware sees anything but 6 digits.

•This number is not padded.

Serial number.

The serial number can contain up to 25 characters and is variable length. The serial number is stored in the Micro-PDF symbol. If a comma follows the serial number, then additional data is used below.

•If present, must be alphanumeric (letters and numbers, no punctuation). This value is used if a comma follows the ECI number.

Additional data.

If present, it is used for things such as a country code. Data cannot exceed 150 bytes. This includes serial number commas.

•Additional data is stored in the Micro-PDF symbol and appended after the serial number. A comma must exist between each maximum of 25 characters in the additional fields.

•Additional data fields can contain up to 25 alphanumeric characters per field.

---

add_barcode_upc_a (**args*, ***kwargs*)
UPC-A Bar Code (^BU)

Parameters

- **orientation** –
  - 'N' - normal
  - 'R' - rotate 90
  - 'I' - inverted
  - 'B' - rotate 270
- **height** – bar code height in dots (1 to 9999)
- **print_text** – print text of data ('Y', 'N')
- **text_above** – print text above barcode ('Y', 'N')
- **check_digit** – print check digit ('Y', 'N')

add_barcode_upc_e (**args*, ***kwargs*)
UPC-E Bar Code (^B9)

Parameters

- **orientation** – 'N' - normal, 'R' - rotate 90, 'I' - inverted, 'B' - rotate 270
- **height** – bar code height in dots (1 to 32000)
- **print_text** – print text of data ('Y', 'N')
- **text_above** – print text above barcode ('Y', 'N')
- **check_digit** – print check digit ('Y', 'N')

add_barcode_upc_ean_extensions (**args*, ***kwargs*)
UPC/EAN Extensions Bar Code (^BS)

Parameters

- **orientation** – 'N' - normal, 'R' - rotate 90, 'I' - inverted, 'B' - rotate 270

- **height** – bar code height in dots (1 to 32000)
- **print_text** – print text of data ('Y', 'N')
- **text_above** – print text above barcode ('Y', 'N')

add_barcode_ups_maxicode (**args*, ***kwargs*)

UPS MaxiCode Bar Code

Parameters

- **mode** –
  - 2 - structured carrier message: numeric postal code (U.S.)
  - 3 - structured carrier message: alphanumeric postal code (non-U.S.)
  - 4 - standard symbol, secretary
  - 5 - full EEC
  - 6 - reader program, secretary
- **symbol_number** – 1-8
- **symbol_count** – 1-8

Considerations for ^FD when Using ^BD The ^FD statement is divided into two parts: a high priority message (hpm) and a low priority message (lpm). There are two types of high priority messages. One is for a U.S. Style Postal Code; the other is for a non-U.S. Style Postal Code. The syntax for either of these high priority messages must be exactly as shown or an error message is generated. Format: ^FD <hpm><lpm>

<hpm> = high priority message (applicable only in Modes 2 and 3) Values: 0 to 9, except where noted U.S. Style Postal Code (Mode 2)

<hpm> = aaabbbccccdddd aaa = three-digit class of service bbb = three-digit country zip code ccccc = five-digit zip code dddd = four-digit zip code extension (if none exists, four zeros (0000) must be entered)

non-U.S. Style Postal Code (Mode 3) <hpm> = aaabbbcccccc aaa = three-digit class of service bbb = three-digit country zip code ccccc = six-digit zip code (A through Z or 0 to 9)

<lpm> = low priority message (only applicable in Modes 2 and 3) GS is used to separate fields in a message (0x1D). RS is used to separate format types (0x1E). EOT is the end of transmission characters.

Message Header []>RS Transportation Data Format Header01GS96 Tracking Number*<tracking number> SCAC*GS<SCAC> UPS Shipper NumberGS<shipper number> Julian Day of Pick-upGS<day of pickup> Shipment ID NumberGS<shipment ID number> Package n/xGS<n/x> Package WeightGS<weight> Address ValidationGS<validation> Ship to Street AddressGS<street address> Ship to CityGS<city> Ship to StateGS<state> RSRS End of MessageEOT (* Mandatory Data for UPS)

Comments • The formatting of <hpm> and <lpm> apply only when using Modes 2 and 3. Mode 4, for example, takes whatever data is defined in the ^FD command and places it in the symbol. • UPS requires that certain data be present in a defined manner. When formatting MaxiCode data for UPS, always use uppercase characters. When filling in the fields in the <lpm> for UPS, follow the data size and types specified in Guide to Bar Coding with UPS. • If you do not choose a mode, the default is Mode 2. If you use non-U.S. Postal Codes, you probably get an error message (invalid character or message too short). When using non-U.S. codes, use Mode 3. • ZPL II doesn't automatically change your mode based on the

zip code format. • When using special characters, such as GS, RS, or EOT, use the ^FH command to tell ZPL II to use the hexadecimal value following the underscore character (_).

add_comment(**args*, ***kwargs*)
Comment Block (^FX)

Parameters **comment_text** – Text to insert as comment

add_field_block(**args*, ***kwargs*)
Field Block (^FB)

Parameters

- **width** – width of text 0 to label width
- **max_lines** – max number of lines in block, 1 to 9999
- **dots_between_lines** – dots between line adjustment -9999 to 9999
- **text_justification** –
  - 'L' - Left
  - 'C' - center
  - 'R' - right
  - 'J' - justified
- **hanging_indent** – 0 to 9999

add_field_data(**args*, ***kwargs*)

Field Data for Text or Barcode (^FD with 1-many ^FS)

param data_list if list or tuple, multiple data blocks with '^FS' separator otherwise, single field with value

param replace_newlines If true, replaces

with &

add_field_data_ansi_codabar(*data*, *start_char=None*, *stop_char=None*)

add_field_data_code_11(*data*)

add_field_data_code_39(*data*, *extended_ascii=False*)
Add field data for code 39 barcode

Parameters

- **data** – Data to encode
- **extended_ascii** – Boolean for extended ascii support

add_field_data_code_93(*data*, *extended_ascii=False*)

Parameters

- **data** –
- **extended_ascii** –

Returns

add_field_data_ean_13(*data*)

add_field_data_ean_8(*data*)
Verify and Add Field Data for EAN 8

Parameters **data** – data for barcode (must be numeric)

add_field_data_industrial_2_of_5(*data*)

add_field_data_interleaved_2_of_5(*data*)

add_field_data_logmars(*data*)

add_field_data_msi(*data*, *check_digit=None*)

add_field_data_planet_code(*data*)

add_field_data_plessey(*data*)

add_field_data_postal(*data*)

add_field_data_standard_2_of_5(*data*)

add_field_data_tlc39(**args*, ***kwargs*)
Add data field for tlc39 barcode

Parameters

- **eci_number** – exactly 6 digit number
- **serial_number** – optional up to 26 character alphanumeric
- **additional_data** – string or list/tuple

add_field_data_upc_a(*data*)

add_field_data_upc_e(*data*)

add_field_data_upc_ean_extensions(*data*)

add_field_origin(**args*, ***kwargs*)
Field Origin (^FO)

Location where Field should start.

Parameters

- **x_pos** – x axis position in dots (0 to 32000)
- **y_pos** – y axis position in dots (0 to 32000)
- **justification** –
  - 0 - left
  - 1 - right
  - 2 - auto

add_font(**args*, ***kwargs*)
Specify font to use in text field (^A)

Parameters

- **font_name** – A-Z or 0-9 of font stored in printer
- **orientation** –
  - 'N' - Normal
  - 'R' - Rotated 90 clockwise
  - 'I' - Inverted
  - 'B' - Bottom Up (270 rotate)

- **character_height** – 10 to 32000 dots
- **width** – 10 to 32000 dots

add_graphic_box(**args*, ***kwargs*)
Produce Graphic Box on Label (^GB)

Parameters

- **width** – border to 32000
- **height** – border to 32000
- **border** – 1 to 32000
- **line_color** –
  - 'B' - black
  - 'W' - white
- **corner_rounding** – 0 (none) to 8 (heaviest rounding)

add_graphic_circle(**args*, ***kwargs*)
Produce Circle on Label (^GC)

Parameters

- **diameter** – 3 to 4095
- **border** – 1 to 4095
- **color** –
  - 'B' - black
  - 'W' - white

add_label_home(**args*, ***kwargs*)
Label Home Position (^LH)

Parameters

- **x_pos** – x axis position in dots (0 to 32000)
- **y_pos** – y axis position in dots (0 to 32000)

add_print_quantity(**args*, ***kwargs*)
Control Printing Quantity and Pausing (^PQ)

Parameters

- **quantity** – total quantity of labels to print (1 to 99,999,999)
- **pause_and_cut_count** – number before pause and cut (0 to 99,999,999 with 0 as disabled)
- **replicates_of_serial** – number of serial duplicates (0 to 99,999,999 with 0 as none)
- **override_pause** – override pause count ('Y', 'N')
- **cut_on_error** – cut on RFID void error label ('Y', 'N')

render_png(*label_width*, *label_height*, *dpmm=8*, *index=0*)
Uses labelary.com api to generate a PNG file

See: examples/display_label_png.py

Parameters

- **label_width** – width in inches
- **label_height** – height in inches
- **dpmm** – dots per mm, default 8
- **index** – label index (only important if you use in label)

Returns byte array of PNG file

zpl_bytes
Renders zpl code as bytestring in UTF-8 formatting.

This is what you would typically send to a printer.

Returns byte array

zpl_text
Renders zpl text as string for debugging.

Returns text string

NetworkPrinter

class simple_zpl2.printer.**NetworkPrinter**(*ip_address*, *port=9100*)
Object to send ZPL to zebra network printer using sockets

print_zpl(*label_formatter*, *timeout=10*)
Send ZPL2 formatted text to a network label printer

Parameters

- **label_formatter** – Formatter object, fully build for label.
- **timeout** – Socket timeout for printer connection, default 10.

Returns None

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/sacherjj/simple_zpl2/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

Write Documentation

Simple ZPL2 could always use more documentation, whether as part of the official Simple ZPL2 docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/sacherjj/simple_zpl2/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *simple_zpl2* for local development.

1. Fork the *simple_zpl2* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/simple_zpl2.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv simple_zpl2
$ cd simple_zpl2/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 simple_zpl2 tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.3, 3.4, 3.5, 3.6 and for PyPy. Check https://travis-ci.org/sacherjj/simple_zpl2/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ py.test tests.test_simple_zpl2
```

CHAPTER 6

Credits

Development Lead

- Joe Sacher <sacherjj@gmail.com>

Contributors

None yet. Why not be the first?

CHAPTER 7

History

0.1.0 (2017-05-26)

- Initial Creation as Standalone package.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

A

E

F

J

N

O

P

Q

R

S

T

Z