python-barcode

Release 0.15.1

Jul 05, 2023

Contents

1 Contents **3**
- 1.1 Getting started . . . 3
- 1.2 Supported Formats . . . 6
- 1.3 Writers . . . 12
- 1.4 Changelog . . . 14
- 1.5 Previous Changelog . . . 16

2 Help **19**

3 Donations **21**

4 Licence **23**

Index **25**

python-barcode is a pure-python library for generating barcodes in various formats. It's 100% pure python.

There are no external dependencies when generating SVG files. Pillow is required for generating images (e.g.: PNGs).

CHAPTER 1

Contents

1.1 Getting started

1.1.1 Installation

The usual way is to use pip:

```
pip install python-barcode
```

Don't forget to add this to our app's dependencies.

If you'll be exporting to images (eg: not just SVG), you'll need the "images" extras:

```
pip install "python-barcode[images]"
# Note: keep the quotes, most shells don't play nice with square brackets.
```

1.1.2 Usage

Let's start off with some code samples.

Keep in mind that checksums are calculated automatically – you don't need to do the math before passing the value for the barcode.

In some systems (Code 39) the checksum is optional. For these, you can provide the `add_checksum=False` keyword argument.

Generating SVG files

```
from io import BytesIO

from barcode import EAN13
```

(continues on next page)

(continued from previous page)

```
from barcode.writer import SVGWriter

# Write to a file-like object:
rv = BytesIO()
EAN13("100000902922", writer=SVGWriter()).write(rv)

# Or to an actual file:
with open("somefile.svg", "wb") as f:
    EAN13(str(100000011111), writer=SVGWriter()).write(f)
```

Generating image files

New in version 0.4b1.

Attention: Keep in mind that SVG files are vectorized, so they will scale a lot better than images. It's recommended to use images only if your medium or target usages does not support SVG.

```
from io import BytesIO

from barcode import EAN13
from barcode.writer import ImageWriter

# Write to a file-like object:
rv = BytesIO()
EAN13(str(100000902922), writer=ImageWriter()).write(rv)

# Or to an actual file:
with open("somefile.jpeg", "wb") as f:
    EAN13("100000011111", writer=ImageWriter()).write(f)
```

Interactive generating an SVG

Using an interactive python interpreter to generate SVG files.

```
>>> import barcode
>>> barcode.PROVIDED_BARCODES
['code128', 'code39', 'ean', 'ean13', 'ean14', 'ean8', 'gs1', 'gs1_128', 'gtin', 'isbn
↪', 'isbn10', 'isbn13', 'issn', 'itf', 'jan', 'pzn', 'upc', 'upca']
>>> EAN = barcode.get_barcode_class('ean13')
>>> EAN
<class 'barcode.ean.EuropeanArticleNumber13'>
>>> my_ean = EAN('5901234123457')
>>> my_ean
<EuropeanArticleNumber13('5901234123457')>
>>> fullname = my_ean.save('ean13_barcode')
>>> fullname
'ean13_barcode.svg'
>>>
```

You can check the generated files (e.g.: `ean13_barcode.svg`) by opening them with any graphical app (e.g.: Firefox).

Interactive generating a PNG

Using an interactive python interpreter to generate PNG files.

```
>>> import barcode
>>> from barcode.writer import ImageWriter
>>> EAN = barcode.get_barcode_class('ean13')
>>> my_ean = EAN('5901234123457', writer=ImageWriter())
>>> fullname = my_ean.save('ean13_barcode')
>>> fullname
'ean13_barcode.png'
>>> from io import BytesIO
>>> fp = BytesIO()
>>> my_ean.write(fp)
>>> my_ean
<EuropeanArticleNumber13('5901234123457')>
>>> with open("path/to/file", "wb") as f:
...     my_ean.write(f)  # Pillow (ImageWriter) produces RAW format here
...
>>> from barcode import generate
>>> name = generate('EAN13', '5901234123457', output='barcode_svg')
>>> name
'barcode_svg.svg'
>>> fp = BytesIO()
>>> generate('EAN13', '5901234123457', writer=ImageWriter(), output=fp)
>>>
```

You can check the generated files (e.g.: `ean13_barcode.png`) by opening them with any graphical app (e.g.: Firefox).

Command Line usage

New in version 0.7beta4.

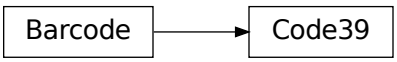
This library also includes a cli app for quickly generating barcodes from the command line or from shell scripts:

```
$ # Save a barcode to outfile.svg:
$ python-barcode create "123456789000" outfile -b ean --text "text to appear under␣
↪barcode"
$ # Generate a PNG (Require Pillow):
$ python-barcode create -t png "My Text" outfile
$ python-barcode --help
usage: python-barcode [-h] [-v] {create,list} ...

Create standard barcodes via cli.

optional arguments:
  -h, --help     show this help message and exit
  -v, --version  show program's version number and exit

Actions:
  {create,list}
    create       Create a barcode with the given options.
    list         List available image and code types.

Image output enabled, use --type option to give image format (png, jpeg, ...).
$
```

1.2 Supported Formats

The following are the supported barcode formats. PRs for other code formats are welcome!

1.2.1 Code 39



class barcode.codex.**Code39**(*code: str*, *writer=None*, *add_checksum: bool = True*)
A Code39 barcode implementation

Parameters

- **code** – Code 39 string without * and without checksum.
- **writer** – A barcode.writer instance used to render the barcode (default: SVGWriter).
- **add_checksum** – Add the checksum to code or not

get_fullcode() → str

Returns The full code as it will be encoded.

render(*writer_options=None*, *text=None*)
Renders the barcode using *self.writer*.

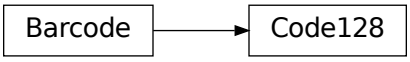
Parameters

- **writer_options** – Options for *self.writer*, see writer docs for details.
- **text** – Text to render under the barcode.

Returns Output of the writers render method.

1.2.2 Code 128

New in version 0.8beta1.



class barcode.codex.**Code128**(*code*, *writer=None*)
Initializes a new Code128 instance. The checksum is added automatically when building the bars.

Parameters

code [String] Code 128 string without checksum (added automatically).

writer [barcode.writer Instance] The writer to render the barcode (default: SVGWriter).

get_fullcode()
Returns the full code, encoded in the barcode.

Returns Full human readable code.

Return type String

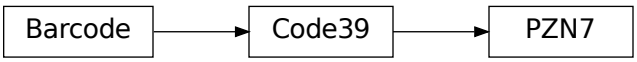
render(*writer_options=None*, *text=None*)
Renders the barcode using *self.writer*.

Parameters

- **writer_options** – Options for *self.writer*, see writer docs for details.
- **text** – Text to render under the barcode.

Returns Output of the writers render method.

1.2.3 PZN7 (aka: PZN)



class barcode.codex.**PZN7**(*pzn*, *writer=None*)
Initializes new German number for pharmaceutical products.

Parameters

pzn [String] Code to render.

writer [barcode.writer Instance] The writer to render the barcode (default: SVGWriter).

get_fullcode()

Returns The full code as it will be encoded.

1.2.4 EAN-13

class barcode.ean.**EuropeanArticleNumber13**(*ean*, *writer=None*, *no_checksum=False*, *guard-bar=False*)

Initializes EAN13 object.

Parameters

ean [String] The ean number as string.

writer [barcode.writer Instance] The writer to render the barcode (default: SVGWriter).

build()

Builds the barcode pattern from *self.ean*.

Returns The pattern as string

Return type String

calculate_checksum()

Calculates the checksum for EAN13-Code.

Returns The checksum for *self.ean*.

Return type Integer

get_fullcode()

Returns the full code, encoded in the barcode.

Returns Full human readable code.

Return type String

render(*writer_options=None*, *text=None*)

Renders the barcode using *self.writer*.

Parameters

- **writer_options** – Options for *self.writer*, see writer docs for details.
- **text** – Text to render under the barcode.

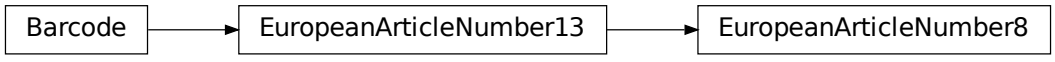
Returns Output of the writers render method.

to_ascii()

Returns an ascii representation of the barcode.

Return type String

1.2.5 EAN-8



class barcode.ean.**EuropeanArticleNumber8**(*ean*, *writer=None*, *no_checksum=False*, *guard-bar=False*)

Represents an EAN-8 barcode. See EAN13's __init__ for details.

Parameters

ean [String] The ean number as string.

writer [barcode.writer Instance] The writer to render the barcode (default: SVGWriter).

build()
Builds the barcode pattern from *self.ean*.

Returns The pattern as string

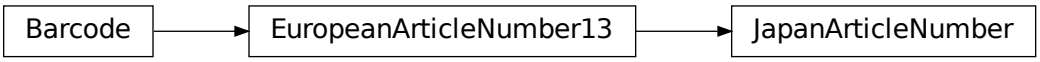
Return type String

get_fullcode()
Returns the full code, encoded in the barcode.

Returns Full human readable code.

Return type String

1.2.6 JAN



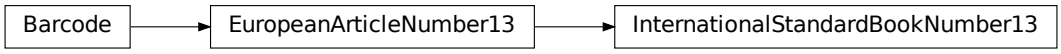
class barcode.ean.**JapanArticleNumber**(*jan*, **args*, ***kwargs*)
Initializes JAN barcode.

Parameters

jan [String] The jan number as string.

writer [barcode.writer Instance] The writer to render the barcode (default: SVGWriter).

1.2.7 ISBN-13



class barcode.isxn.**InternationalStandardBookNumber13**(*isbn*, *writer=None*)
Initializes new ISBN-13 barcode.

Parameters

isbn [String] The isbn number as string.

writer [barcode.writer Instance] The writer to render the barcode (default: SVGWriter).

1.2.8 ISBN-10

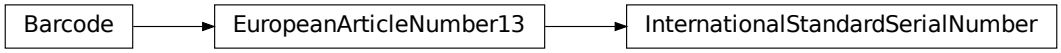
class barcode.isxn.**InternationalStandardBookNumber10**(*isbn*, *writer=None*)
Initializes new ISBN-10 barcode. This code is rendered as EAN-13 by prefixing it with 978.

Parameters

isbn [String] The isbn number as string.

writer [barcode.writer Instance] The writer to render the barcode (default: SVGWriter).

1.2.9 ISSN



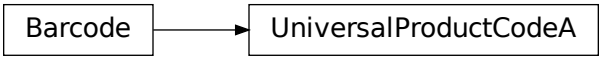
class barcode.isxn.**InternationalStandardSerialNumber**(*issn*, *writer=None*)
Initializes new ISSN barcode. This code is rendered as EAN-13 by prefixing it with 977 and adding 00 between code and checksum.

Parameters

issn [String] The issn number as string.

writer [barcode.writer Instance] The writer to render the barcode (default: SVGWriter).

1.2.10 UPC-A



class barcode.upc.**UniversalProductCodeA**(*upc*, *writer=None*, *make_ean=False*)
Universal Product Code (UPC) barcode.

UPC-A consists of 12 numeric digits.

Initializes new UPC-A barcode.

Parameters

- **upc** (*str*) – The upc number as string.
- **writer** – barcode.writer instance. The writer to render the barcode (default: SVGWriter).
- **make_ean** (*bool*) – Indicates if a leading zero should be added to the barcode. This converts the UPC into a valid European Article Number (EAN).

build()
Builds the barcode pattern from 'self.upc'

Returns The pattern as string

Return type str

calculate_checksum()
Calculates the checksum for UPCA/UPC codes

Returns The checksum for 'self.upc'

Return type int

get_fullcode()
Returns the full code, encoded in the barcode.

Returns Full human readable code.

Return type String

render(*writer_options=None*, *text=None*)
Renders the barcode using *self.writer*.

Parameters

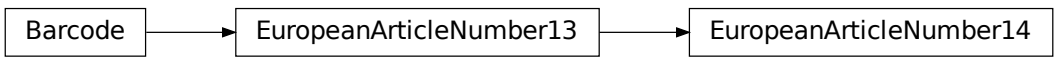
- **writer_options** – Options for *self.writer*, see writer docs for details.
- **text** – Text to render under the barcode.

Returns Output of the writers render method.

to_ascii()
Returns an ascii representation of the barcode.

Return type str

1.2.11 EAN14



class barcode.ean.**EuropeanArticleNumber14**(*ean*, *writer=None*, *no_checksum=False*, *guard_bar=False*)

Represents an EAN-14 barcode. See EAN13's __init__ for details.

Parameters

ean [String] The ean number as string.

writer [barcode.writer Instance] The writer to render the barcode (default: SVGWriter).

calculate_checksum()

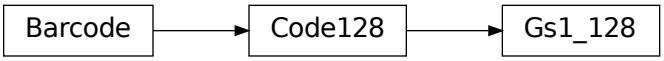
Calculates the checksum for EAN13-Code.

Returns The checksum for *self.ean*.

Return type Integer

1.2.12 GS1-128

New in version v0.10.0.



class barcode.codex.**Gs1_128**(*code*, *writer=None*)

following the norm, a gs1-128 barcode is a subset of code 128 barcode, it can be generated by prepending the code with the FNC1 character https://en.wikipedia.org/wiki/GS1-128 https://www.gs1-128.info/

get_fullcode()

Returns the full code, encoded in the barcode.

Returns Full human readable code.

Return type String

1.3 Writers

1.3.1 Common Writer Options

All writer take the following options (specified as keyword arguments to `Barcode.save(filename, options)` or set via `Writer.set_options(options)`, where `options` is a dictionary where keys are option names and values are option values to be set).

Note: See the documentation of the specific writer for special options, only available for this writer.

module_width The width of one barcode module in mm as *float*. Defaults to **0.2**.

module_height The height of the barcode modules in mm as *float*. Defaults to **15.0**.

quiet_zone Distance on the left and on the right from the border to the first (last) barcode module in mm as *float*. Defaults to **6.5**.

font_path Path to the font file to be used. Defaults to **DejaVuSansMono** (which is bundled with this package).

font_size Font size of the text under the barcode in pt as *integer*. Font size zero suppresses text. Defaults to **10**.

text_distance Distance between the barcode and the text under it in mm as *float*. Defaults to **5.0**.

background The background color of the created barcode as *string*. Defaults to **white**.

foreground The foreground and text color of the created barcode as *string*. Defaults to **black**.

center_text If true (the default) the text is centered under the barcode else left aligned.

New in version 0.6.

Note: Some barcode classes change the above defaults to fit in some kind of specification.

1.3.2 SVGWriter

Renders barcodes as [optionally, compressed] SVG objects.

In addition to the common writer options you can give the following special option.

compress Boolean value to output a compressed SVG object (.svgz). Defaults to `False`

1.3.3 ImageWriter

New in version 0.4b1.

Renders barcodes as image. Supports all the image formats supported by Pillow.

In addition to the common writer options you can give the following special options:

format The image file format as `str`. All formats supported by Pillow are valid (e.g. PNG, JPEG, BMP, ...). Defaults to `PNG`.

dpi DPI as `int` to calculate the image size in pixel. This value is used for all mm to px calculations. Defaults to `300`

1.3.4 Custom writers

It's possible to create your own writer by inheriting from *barcode.writer.BaseWriter*.

In your `__init__` method call BaseWriter's `__init__` and give your callbacks for:

- `initialize(raw_code)`
- `paint_module(xpos, ypos, width, color)`
- `paint_text(xpos, ypos)`
- `finish()`

Now instantiate a new barcode and give an instance of your new writer as argument. If you now call `render` on the barcode instance your callbacks get called.

1.3.5 Creating compressed SVGs

Saving a compressed SVG (SVGZ):

```
>>> import barcode
>>> ean = barcode.get('ean13', '123456789102')
# Now we look if the checksum was added
>>> ean.get_fullcode()
'1234567891026'
>>> filename = ean.save('ean13')
>>> filename
'ean13.svg'
>>> options = dict(compress=True)
>>> filename = ean.save('ean13', options)
>>> filename
'ean13.svgz'
```

Now you have ean13.svg and the compressed ean13.svgz in your current working directory. Open it and see the result.

1.4 Changelog

1.4.1 v0.15.1

- Add missing dependency to release script.

1.4.2 v0.15.0

- **Breaking** Dropped support for Python 3.6.
- Added support for Python 3.11.
- Fixed compatibility with Pillow 10.0.
- Updated ISBN to support newer allocated ranges.
- Improved type hints.

1.4.3 v0.14.0

- **Breaking**: The default dimensions have changed slightly. This is so that the results of generating a PNG and an SVG look more alike.
- Previous versions included an empty text element for SVGs with no comment. This is no longer the case.
- Some internals have been improved so as to allow better subclassing. Subclasses of `Barcode` can now override `default_writer_options` and `default_writer()`.
- A `guardbar` parameter has been added to EAN barcodes. This renders barcodes with guardars (longer bars).
- Added support for Python 3.10.
- The documentation setup has been redone, hopefully squashing a lot of legacy quirks.
- Previous versions installed the *tests* module. This was not intentional and have been fixed.

1.4.4 v0.13.1

- Fix a crash when using the `generate` shortcut function.

1.4.5 v0.13.0

- Added support for transparent backgrounds. This is done by setting the `mode` option for a writer to `RGBA`.
- Dropped support for Python 3.5.
- Added support for Python 3.9.

1.4.6 v0.12.0

- Removed `writer_options` from `barcode.get`. This parameter was not used.
- Add a `with_doctype` flag to `SVGWriter`. Set this to false to avoid including a `DOCTYPE` in the resulting SVG.
- Add support for `Pillow>=8.0.0`.

1.4.7 v0.11.0

- Added basic support for multiline text.
- Dropped lots of older compat-only code and other cleanups.
- Fixed a bug in the API when combining certain barcodes and writers.
- Published documentation again and updated all project references.
- Fix python_barcode.get mixups between *options* as *writer_options*. Previously, some writer/barcode combinations worked fine, while others failed. Now all work consistently.
- The cli tool has been fixed and should now work as expected again.

1.4.8 v0.10.0

- Added support for GS1-128.

1.4.9 v0.9.1

- Officially support Python 3.7
- Refer to Pillow in the docs, rather than PIL.

1.4.10 v0.9.0

- Removed buggy `Barcode.raw` attribute.
- Various CLI errors ironed out.
- Make the default value for `writer_options` consistent across writers.

1.4.11 v0.8.3

- Fix pushing of releases to GitHub.

1.4.12 v0.8.2

- Fix crashes when attempting to use the CLI app.
- Properly include version numbers in SVG comments.

1.4.13 v0.8.1

- Improve README rendering, and point to this fork's location (the outdated README on PyPI was causing some confusion).

1.4.14 v0.8.0

- First release under the name `python-barcode`.

1.5 Previous Changelog

This project is a fork of pyBarcode, which, apparently, is no longer maintained. v0.8.0 is our first release, and is the latest `master` from that parent project.

1.5.1 v0.8

- Code 128 added.
- Data for charsets and bars moved to subpackage barcode.charsets.
- Merged in some improvements.

1.5.2 v0.7

- Fixed some issues with fontsize and fontalignment.
- Added Python 3 support. It's not well tested yet, but the tests run without errors with Python 3.3. Commandline script added.

1.5.3 v0.6

- Changed save and write methods to take the options as a dict not as keyword arguments (fix this in your code). Added option to left align the text under the barcode. Fixed bug with EAN13 generation.

1.5.4 v0.5.0

- Added new generate function to do all generation in one step.
- Moved writer from a subpackage to a module (this breaks some existing code). UPC is now rendered as real UPC, not as EAN13 with the leading "0".

1.5.5 v0.4.3

- Fixed bug in new write method (related to PIL) and updated docs.

1.5.6 v0.4.2

- Added write method to support file like objects as target.

1.5.7 v0.4.1

- Bugfix release. Removed redundancy in input validation.
- EAN8 was broken. It now works as expected.

1.5.8 v0.4

- Removed **options from writers __init__ method. These options never had effect. They were always overwritten by default_options.
- New config option available: text_distance (the distance between barcode and text).

1.5.9 v0.4b2

- Basic documentation included. The barcode object now has a new attribute called *raw* to have the rendered output without saving to disk.

1.5.10 v0.4b1

- Support for rendering barcodes as images is implemented. PIL is required to use it.

1.5.11 v0.3

- Compression for SVG output now works.

1.5.12 v0.3b1

- Writer API has changed for simple adding new (own) writers.
- SVG output is now generated with xml.dom module instead of stringformatting (makes it more robust).

1.5.13 v0.2.1

- API of render changed. Now render takes keyword arguments instead of a dict.

1.5.14 v0.2

- More tests added.

1.5.15 v0.1

- First release.

CHAPTER 2

Help

If you think you found a bug, or need help with something specific:

- Please check existing issues for similar topics.
- If there's nothing relevant, please open a new issue describing your problem, and what you've tried so far.

Issues and source code are all in GitHub.

CHAPTER 3

Donations

Donations are welcome. See here for further details.

CHAPTER 4

Licence

python-barcode is licensed under the MIT licence. See LICENCE for details.

Index

B

build() (*barcode.ean.EuropeanArticleNumber13 method*), 8
build() (*barcode.ean.EuropeanArticleNumber8 method*), 9
build() (*barcode.upc.UniversalProductCodeA method*), 11

C

calculate_checksum() (*barcode.ean.EuropeanArticleNumber13 method*), 8
calculate_checksum() (*barcode.ean.EuropeanArticleNumber14 method*), 12
calculate_checksum() (*barcode.upc.UniversalProductCodeA method*), 11
Code128 (*class in barcode.codex*), 6
Code39 (*class in barcode.codex*), 6

E

EuropeanArticleNumber13 (*class in barcode.ean*), 7
EuropeanArticleNumber14 (*class in barcode.ean*), 11
EuropeanArticleNumber8 (*class in barcode.ean*), 8

G

get_fullcode() (*barcode.codex.Code128 method*), 7
get_fullcode() (*barcode.codex.Code39 method*), 6
get_fullcode() (*barcode.codex.Gs1_128 method*), 12
get_fullcode() (*barcode.codex.PZN7 method*), 7
get_fullcode() (*barcode.ean.EuropeanArticleNumber13 method*), 8
get_fullcode() (*barcode.ean.EuropeanArticleNumber8 method*), 9
get_fullcode() (*barcode.upc.UniversalProductCodeA method*), 11
Gs1_128 (*class in barcode.codex*), 12

I

InternationalStandardBookNumber10 (*class in barcode.isxn*), 10
InternationalStandardBookNumber13 (*class in barcode.isxn*), 9
InternationalStandardSerialNumber (*class in barcode.isxn*), 10

J

JapanArticleNumber (*class in barcode.ean*), 9

P

PZN7 (*class in barcode.codex*), 7

R

render() (*barcode.codex.Code128 method*), 7
render() (*barcode.codex.Code39 method*), 6
render() (*barcode.ean.EuropeanArticleNumber13 method*), 8
render() (*barcode.upc.UniversalProductCodeA method*), 11

T

to_ascii() (*barcode.ean.EuropeanArticleNumber13 method*), 8
to_ascii() (*barcode.upc.UniversalProductCodeA method*), 11

U

UniversalProductCodeA (*class in barcode.upc*), 10