Segno Documentation

Author

Jun 25, 2019

Contents

Segno generates QR Codes and Micro QR Codes according to ISO/IEC 18004:2015(E). Aside from other Python QR Code generators, Segno has no 3rd party dependencies and is very simple to use.

Contents:

CHAPTER 1

QR Code creation from the command line

The command line script "segno" can be used to print QR Codes to the terminal or to save them as file (SVG, PNG, EPS, ...).

By default, the script does not create Micro QR Codes, use `--micro` to allow the creation of Micro QR Codes or specify the version (like `--version=M3`) to create a Micro QR Code.

1.1 Usage

Output the QR Code to the terminal:

```
$ segno "Little wing"
```

Same content, but as Micro QR Code (M4):

```
$ segno --micro "Little wing"
```

1.1.1 Version

If the `version` parameter is not provided, Segno chooses the minimal version for the QR Code automatically. The version may be specified as integer or as Micro QR Code identifier.

The content 'Layla' would fit into a version 1 QR Code, but the following command enforces version 5:

```
$ segno --version=5 Layla
$ segno -v=5 Layla
```

Micro QR Code:

```
$ segno -v m4 Layla
$ segno --version M4 Layla
```

1.1.2 Error correction level

The default error correction level is “L”, use the `error` parameter to change it:

```
$ segno --error=q "Ain't no grave"
$ segno -e=h "Heart of Gold"
```

“Heart of Gold” using error level “L”:

“Heart of Gold” using error level “H”:

1.1.3 QR Code serialization

Printing the QR Codes to the terminal is nice but the `output` parameter serializes the QR Code in one of the supported file formats:

```
$ segno --output=white-room.png "White Room"
```

```
$ segno -o=satellite.svg "Satellite Of Love"
```

```
$ segno --output=mrs.eps "Mrs. Robinson"
$ segno --output=dedodo.pdf "De Do Do Do, De Da Da Da"
$ segno --output=tin-pan-alley.svgz "Tin Pan Alley"
```

```
$ segno --output=thrill-gone.txt "The Thrill Is Gone"
```

```
00000000000000000000000000000000000
00000000000000000000000000000000000
00000000000000000000000000000000000
00000000000000000000000000000000000
00001111111010111011001111111110000
00001000001011101111001000001000000
00001011101010001001001011101000000
00001011101011101100001011101000000
00001011101010001011001011101000000
00001000001000111101101000001000000
00001111111010101010101111111110000
00000000000011010100100000000000000
00000110101100000011101011111000000
00001001010100010000111000011000000
00000000111100010100010011111000000
00000000010001000000110011010000000
00001110101100100000101000000000000
00000100000010001001111001011000000
00001000111100010100101110111000000
00000111000010000001010110010000000
00001000101111011110111111111000000
00000000000011000101100010000000000
00001111111011001101101010111000000
00001000001000101111100010011000000
00001011101010110010111110001000000
00001011101001011010100111100000000
00001011101010110111010111010000000
00001000001011101100111110010000000
00001111111001001101111011011000000
00000000000000000000000000000000000
00000000000000000000000000000000000
00000000000000000000000000000000000
00000000000000000000000000000000000
```

1.1.4 Scaling QR Codes

If the resulting QR Code is too small, `scale` can be used to create a more appropriate output:

```
$ segno --scale=10 --output=money-talks.png "Money Talks"
```

```
$ segno -s 10 --output=private-investigations.svg Private Investigations
```

If the serializer does not support a scaling factor (i.e. text output), this parameter is ignored.

1.1.5 Changing the size of the quiet zone

The generated QR Codes will have a recommended quiet zone / border around the symbol. To change the size of the border, `border` can be utilized:

```
$ segno --border=0 --output=black-magic-woman.svg "Black Magic Woman"
```

```
$ segno --border=10 --output=diamond.png "Shine On You Crazy Diamond"
```

```
$ segno -b=30 --output=boom-boom.svg Boom Boom
```

1.1.6 Colors

Usually, all QR Codes are serialized in black and white. Use `color` to change the color of the dark modules and `background` to change the color of the light modules.

Change the foreground color to darkblue:

```
$ segno --color=darkblue --output=excited.png "So Excited"
```

Change the background color to transparent:

```
$ segno --background=transparent --output=hotel.png "Hotel California"
```

Change the foreground color to darkblue and background to yellow:

```
$ segno --color=darkblue --background=yellow --output=dontgiveup.svg "Don't Give Up"
```

If the serializer does not support `color` or `background`, these arguments are ignored.

1.1.7 Structured Append

The *Structured Append* mode can be used to split a message across multiple (max. 16) QR Codes.

To create a sequence of QR Codes, the `seq` argument must be provided. Additonally, either the QR Code version or the desired number of symbols must be provided:

```
$ segno --seq -v 1 "Well you should see Polythene Pam"
$ segno --seq --symbol-count=2 "We all live in a yellow submarine"
$ segno --seq -sc 2 "Half of what I say is meaningless"
```

CHAPTER 2

Creating QR Codes or Micro QR Codes

Segno's (Micro) QR Codes are independent of a concrete output format; it's possible to create more than one rendering (output format) from a single QR Code or Micro QR Code:

```
>>> import segno
>>> qr = segno.make('Henry Lee')
>>> qr.save('henry-lee.svg')  # SVG document
>>> qr.save('henry-lee.png')  # PNG image
>>> qr.save('henry-lee.eps')  # EPS document
>>> qr.save('henry-lee.txt')  # Text output
```

By default, the serialized (Micro) QR Codes are black and have a quiet zone (border) of four (or two for Micro QR Codes) modules. Nearly all output methods provide options to change the color and border.

```
>>> import segno
>>> qr = segno.make('You Know My Name (Look Up The Number)')
>>> qr.save('you-know-my-name-no-border.svg', border=0)  # no border / quiet zone
>>> qr.save('you-know-my-name-color-green.svg', color='green')  # default border,␣
↪dark modules are green
>>> qr.save('you-know-my-name-background-grey.svg', background='#eee')  # default␣
↪border, background grey
```

The factory function *segno.make()* chooses the minimal possible (Micro) QR Code for the provided input.

```
>>> import segno
>>> qr = segno.make('Rain')
>>> qr.version
'M3'
```

The caller may enforce that a QR Code instead of a Micro QR Code should be generated even if the content may fit into a Micro QR Code.

```
>>> import segno
>>> qr = segno.make('Rain', micro=False)
```

(continues on next page)

(continued from previous page)

```
>>> qr.version
1
```

Further, Segno provides two additional factory functions to enforce the creation of QR Codes or Micro QR Codes: *segno.make_qr()* for QR Codes and *segno.make_micro()* to create Micro QR Codes:

```
>>> import segno
>>> mqr = segno.make_micro('The Beatles')  # Micro QR Code
>>> mqr.designator  # Get the version and error level
'M4-M'
>>> qr = segno.make_qr('The Beatles')  # Same content but as QR Code
>>> qr.designator
'1-Q'
```

If the provided content is too large, a *segno.DataOverflowError* is thrown:

```
>>> import segno
>>> qr = segno.make_micro('The Curse of Millhaven')
Traceback (most recent call last):
    ...
DataOverflowError: Data too large. No Micro QR Code can handle the provided data
```

2.1 Version

It's possible to specify the desired version for the provided `content`.

```
>>> import segno
>>> qr = segno.make('Light My Fire')
>>> qr.version
'M4'
>>> qr = segno.make('Light My Fire', version=1)
>>> qr.version
1
```

2.2 Error Correction Level

By default, Segno uses the error correction level "L" to encode the (Micro) QR Code. Use the parameter `error` to change the error correction level. The `error` parameter is case-insensitive; to specify the error correction level "L", "l" and "L" are valid values. Available error correction levels are `L` (lowest error correction level), `M`, `Q` and `H`. The error correction level "H" is not available for Micro QR Codes, if the user specifies the error correction level "H", a QR Code is generated by `make`, never a Micro QR Code.

```
>>> import segno
>>> qr = segno.make('Parisienne Walkways', error='l')  # Explicit error correction␣
↪level
>>> qr.version
2
>>> # Enhancing the error correction level may enforce another QR Code version
>>> qr = segno.make('Parisienne Walkways', error='H')
>>> qr.version
3
```

2.3 Data Masking

Segno chooses by default the optimal mask for the provided input, but the user may specify the preferred mask as well.

```
>>> import segno
>>> qr = segno.make('Ai Du')
>>> qr.mask
0
>>> qr = segno.make('Ai Du', mask=3)
>>> qr.mask
3
```

CHAPTER 3

Structured Append

The Structured Append mode can be used to split a message across several QR Codes (it's not available for Micro QR Codes).

Segno provides a special factory function, *segno.make_sequence()*, to create a sequence of (up to 16) QR Codes. The function returns instances of *segno.QRCodeSequence*.

3.1 Structured Append by QR Code version

To create a sequence of QR Codes, the QR Code version must be specified. The number of symbols is automatically determined by the QR Code version.

```
>>> import segno
>>> seq = segno.make_sequence('I read the news today oh boy', version=1)
>>> len(seq)
2
>>> # Creates "a-day-in-the-life-02-01.svg" and "a-day-in-the-life-02-02.svg"
>>> seq.save('a-day-in-the-life.svg', scale=10)
```

If the provided content fits into one QR Code, the sequence behaves like a *segno.QRCode* instance.

```
>>> import segno
>>> seq = segno.make_sequence('I read', version=1)
>>> len(seq)
1
>>> seq.designator
'1-H'
>>> # Creates "a-day-in-the-life.svg"
>>> seq.save('a-day-in-the-life.svg', scale=10)
```

3.2 Structured Append by number of symbols

The number of desired QR Code symbols may be specified directly. The utilized QR Code version is automatically determined by the number of symbols.

```
>>> import segno
>>> seq = segno.make_sequence('Day after day, alone on the hill', symbol_count=4)
>>> [qr.designator for qr in seq]
['1-Q', '1-Q', '1-Q', '1-Q']
>>> seq = segno.make_sequence('Day after day, alone on the hill', symbol_count=2)
>>> [qr.designator for qr in seq]
['2-Q', '2-Q']
>>> seq = segno.make_sequence('Day after day, alone on the hill', symbol_count=6)
>>> [qr.designator for qr in seq]
['1-Q', '1-Q', '1-H', '1-H', '1-H', '1-H']
```

CHAPTER 4

Special QR Code factory functions

The `helpers` module provides factory functions to create common QR Codes for encoding WIFI configurations, MeCards or geographic locations.

The created QR Codes use at minimum the error correction level "L". If a better error correction level is possible without changing the QR Code version, the better error correction level will be used.

4.1 Create a QR Code for a WIFI configuration

```
>>> from segno import helpers
>>> # Create a WIFI config with min. error level "L" or better
>>> qr = helpers.make_wifi(ssid='My network', password='secret', security='WPA')
>>> qr.designator
'3-M'
```

If you want more control over the creation of the QR Code (i.e. using a specific version or error correction level, use the "make_wifi_data" factory function, which returns a string which encodes the WIFI configuration.

```
>>> import segno
>>> from segno import helpers
>>> config = helpers.make_wifi_data(ssid='My network', password='secret', security=
↪'WPA')
>>> config
'WIFI:T:WPA;S:My network;P:secret;;'
>>> # Create a QR Code with error correction level "L"
>>> qr = segno.make(config, error='l', boost_error=False)
>>> qr.designator
'3-L'
```

4.2 Create a QR Code encoding geographic information

```
>>> from segno import helpers
>>> latitude, longitude = 38.8976763,-77.0365297
>>> qr = helpers.make_geo(latitude, longitude)
>>> qr.designator
'2-M'
```

A factory function for encoding the geographic information as string is also available.

```
>>> import segno
>>> from segno import helpers
>>> latitude, longitude = 38.8976763, -77.0365297
>>> geo_uri = helpers.make_geo_data(latitude, longitude)
>>> geo_uri
'geo:38.8976763,-77.0365297'
>>> # Use version 4 instead of the minimum version
>>> qr = segno.make(geo_uri, version=4)
>>> qr.designator
'4-H'
```

4.3 Creating a QR Code encoding contact information

4.3.1 MeCard

The function `make_mecard` returns a QR Code which encodes contact information as MeCard.

```
>>> from segno import helpers
>>> qr = helpers.make_mecard(name='Doe,John', email='me@example.org', phone='+1234567
↪')
>>> qr.designator
'3-L'
>>> # Some params accept multiple values, like email, phone, url
>>> qr = helpers.make_mecard(name='Doe,John', email=('me@example.org',
↪'another@example.org'), url=['http://www.example.org', 'https://example.org/~joe'])
>>> qr.save('my-mecard.svg')
```

A factory function which returns the MeCard as string is available as well.

```
>>> import segno
>>> from segno import helpers
>>> mecard = helpers.make_mecard_data(name='Doe,John', email='me@example.org', phone=
↪'+1234567')
>>> mecard
'MECARD:N:Doe,John;TEL:+1234567;EMAIL:me@example.org;;'
>>> qr = segno.make(mecard, error='H')
>>> qr.designator
'6-H'
```

4.3.2 vCard

The function `make_vcard` returns a QR Code which encodes contact information as vCard version 3.0.

```
>>> from segno import helpers
>>> qr = helpers.make_vcard(name='Doe;John', displayname='John Doe', email=
↪'me@example.org', phone='+1234567')
>>> qr.designator
'5-L'
>>> # Some params accept multiple values, like email, phone, url
>>> qr = helpers.make_vcard(name='Doe;John', displayname='John Doe', email=(
↪'me@example.org', 'another@example.org'), url=['http://www.example.org', 'https://
↪example.org/~joe'])
>>> qr.save('my-vcard.svg')
```

A factory function which returns the vCard as string is available as well.

```
>>> import segno
>>> from segno import helpers
>>> vcard = helpers.make_vcard_data(name='Doe;John', displayname='John Doe', email=
↪'me@example.org', phone='+1234567')
>>> vcard
'BEGIN:VCARD\r\nVERSION:3.0\r\nN:Doe;John\r\nFN:John Doe\r\nEMAIL:me@example.
↪org\r\nTEL:+1234567\r\nEND:VCARD\r\n'
>>> qr = segno.make(vcard, error='H')
>>> qr.designator
'9-H'
```

CHAPTER 5

Colors

Almost all serializers support custom settings for the color of the dark modules and the background (light modules) of a (Micro) QR Code.

The color values can be provides as tuple (`(R, G, B)`), as web color name (like 'red') or as hexadecimal `#RRGGBB` value (i.e. '#085A75'). If alpha transparency is supported (i.e. PNG and SVG), hexadecimal values like `#RRGGBBAA` are accepted.

The color values are automatically converted into a meaningful value of the specific output format (note that providing an alpha channel to a serializer which does not accept an alpha channel results into an error).

CHAPTER 6

QR Code and Micro QR Code serialization

A QR Code or Micro QR Code is independent of its output, it's just a matrix. To save a QR Code or Micro QR Code, Segno provides several output formats.

Segno provides a *segno.QRCode.save()* method to serialize the (Micro) QR Code in different formats:

```
>>> import segno
>>> qr = segno.make('Polly')
>>> qr.save('polly.svg')
>>> qr.save('polly.png')
>>> qr.save('polly.eps')
```

All serializers accept a `border` parameter which indicates the "quiet zone" of a (Micro) QR Code. If `border` is `None`, the default border (quiet zone) size will be used. If the resulting (Micro) QR Code should have no border or a custom border, you may specify the border

```
>>> import segno
>>> qr = segno.make('Vampire Blues')
>>> qr.save('vampire-blues.svg', border=0)  # No border
>>> qr.save('vampire-blues.png', border=10)  # Bigger border
```

Most serializers accept a `scale` parameter which indicates the scaling factor of the serialization. By default, the scaling factor is `1` which means that the dark / light modules of a (Micro) QR Code is interpreted as one unit in the specific user space (i.e. 1 pixel for the PNG serializer or 1 point (1/72 of an inch) in EPS). Some serializers (like PNG) accept only an integer value or convert the provided scaling factor to an integer. Other, like SVG and EPS, accept float values and do not "downgrade" it to an integer.

```
>>> import segno
>>> qr = segno.make_qr('The Beatles')
>>> qr.save('the-beatles.png', scale=1.2)   # No scaling at all since int(1.2) is 1
>>> qr.save('the-beatles-2.png', scale=10)  # 1 module == 10 pixels
>>> qr.save('the-beatles.svg', scale=1.2)   # SVG accepts float values
>>> # The SVG serializer provides the "unit" parameter to specify
>>> # how to interpret the values
```

(continues on next page)

(continued from previous page)

```
>>> qr.save('the-beatles-2.svg', scale=10, unit='mm')  # 1 unit = 1 mm
>>> qr.save('the-beatles-2.svg', unit='cm')  # 1 unit = 1 cm, result as above
```

Many serializers accept the parameters `color` and `background` to specify the color of the dark modules and light modules (background). See *Colors* for details.

```
>>> import segno
>>> qr = segno.make('Neil Young')
>>> qr.save('neil-young.svg', color='darkblue', background='yellow')
>>> qr.save('neil-young.png', color='#ccc')
>>> qr.save('neil-young-2.png', background=None)  # Transparent background
>>> # Dark modules = transparent, light modules = black
>>> qr.save('neil-young-3.png', color=None, background='black')
>>> # Dark modules with alpha transparency
>>> qr.save('neil-young-4.png', color='#0000ffcc')
>>> qr.save('neil-young-4.svg', color='#00fc')  # Same as above but SVG
>>> # Anonther color, save as compressed SVG
>>> qr.save('neil-young-5.svgz', color=(8, 90, 117))
```

If the QR Code should be serialized to a buffer, use the `kind` parameter to specify the output format.

```
>>> import segno
>>> import io
>>> qr = segno.make('Neil Young')
>>> buff = io.BytesIO()
>>> qr.save(buff, kind='svg')
>>> # All other serializer parameters are supported as well
>>> buff = io.BytesIO()
>>> qr.save(buff, kind='svg', color='#ccc', background='green')
```

See *segno.QRCode.save()* for a complete reference which parameters are accepted by the specific serializer.

CHAPTER 7

Comparison of Python QR Code libraries

7.1 Features

Description	qrcode	PyQRCode	qrcodegen	Segno
Library license	BSD	BSD	MIT	BSD
Library version	6.1	1.3.0	1.3.0	0.3.0
Mode Numeric	Yes	Yes	Yes	Yes
Mode Alphanumeric	Yes	Yes	Yes	Yes
Mode Byte	Yes	Yes	Yes	Yes
Mode Kanji	No	Yes	No	Yes
Mode ECI	No	No	Yes	Yes
Mode FNC1	No	No	No	No
Mode Structured Append	No	No	No	Yes
Mixing modes	Yes	No	Yes	Yes
QR Codes version 1 - 40	Yes	Yes	Yes	Yes
Micro QR Codes version M1 - M4	No	No	No	Yes
Output acc. to ISO/IEC 18004:2015(E) Fig. 1	Yes	No	No	Yes
Output acc. to ISO/IEC 18004:2015(E) I.3.	No (not available)	No (not available)	No (not available)	Yes
Find maximal error correction level	No	No	Yes	Yes

Continued on next page

Table 1 – continued from previous page

Description	qrcode	PyQRCode	qrcodegen	Segno
Optimize QR Codes	Yes	No	No	No
PNG output	Yes	Yes	No	Yes
SVG output	Yes	Yes	Yes	Yes
EPS output	Yes	Yes	No	Yes
PDF output	Yes	No	No	Yes
XBM output	Yes	Yes	No	Yes
XPM output	No	No	No	Yes
PBM output	Yes	No	No	Yes
PAM output	No	No	No	Yes
LaTeX support	No	No	No	Yes
PNG data URI	No	Yes	No	Yes
SVG data URI	No	No	No	Yes
Text output	Yes	Yes	No	Yes
ANSI escape code output	Yes	Yes	No	Yes
Other output formats (i.e. JPEG)	Yes	No	No	No, but via PIL plugin
Black and white QR Codes	Yes	Yes	Yes	Yes
Colored QR Codes	Yes	Yes	No	Yes
Animated QR Codes (GIF, APNG)	No	No	No	No
Changing size of modules (scaling factor)	Yes	Yes	No	Yes
Command line script	Yes	Yes	No	Yes
Plugins	No	No	No	Yes
Default encoding in Byte mode	UTF-8	ISO/IEC 8859-1 or UTF-8	UTF-8	ISO/IEC 8859-1 or UTF-8
3rd party dependencies	six, Pillow or Pymaging and Pymaging-PNG (Windows: colorama)	PyPNG	•	•

7.2 Performance

Some performance indicators. The script benchmarks.py ran on a Mac Mini 2,26 Core2 Duo using CPython 2.7.15. Each SVG / PNG image uses a scaling factor of 10 (aside from qrcodegen which does not support any scaling).

7.2.1 Create a QR Code

Create a 1-M QR Code “QR Code Symbol”.

Create a 30-H QR Code “QR Code Symbol”.

7.2.2 Create a QR Code and serialize it as SVG

Create a QR Code 1-M “QR Code Symbol” and serialize it as SVG document.

7.2.3 Create a QR Code and serialize it as PNG

Create a QR Code 1-M “QR Code Symbol” and serialize it as PNG image.

CHAPTER 8

Replace existing QR Code libs with Segno

The sub-project Segno Mimos provides APIs which are usable to replace common Python QR Code libs like PyQR-Code and qrcode with Segno.

CHAPTER 9

Writing plugins for Segno

segno.QRCode provides a plugin architecture which utilizes egg entry points.

All plugins must use the `segno.plugin.converter` entry point to be recognized. If a plugin is detected, the user may use the the plugin by calling `to_XXX` where `XXX` refers to the plugin name.

The specific plugin is invoked by providing the *segno.QRCode* instance and any further arguments or keywords.

9.1 Simple plugin

This section explains how to write a plugin which writes to `stdout` and uses `X` for dark modules and _ for light modules.

Content of `simple_plugin.py`:

```
import sys

def write(qrcode):
    write = sys.stdout.write
    for row in qrcode.matrix:
        for col in row:
            write('X' if col else '_')
        write('\n')
```

`setup.py`:

```
setup(
    name='simple-plugin',
    version='1.0',
    license='BSD',
    author='John Doe',
    author_email='john@example.org',
    platforms=['any'],
    py_modules=['simple_plugin'],
```

(continues on next page)

(continued from previous page)

```
    entry_points="""
    [segno.plugin.converter]
    simple = simple_plugin:write
    """,
    install_requires=['segno'],
)
```

Once installed, it's possible to call this plugin via:

```
>>> import segno
>>> qr = segno.make('Chelsea Hotel No. 2')
>>> qr.to_simple()
XXXXXXX_XXXX_XX_X_XXXXXXX
X_____X_____________X_____X
X_XXX_X_XX__XXX___X_XXX_X
X_XXX_X__X_XXX_X__X_XXX_X
X_XXX_X__XX_XX__X_X_XXX_X
X_____X_X__XXX__X_X_____X
XXXXXXX_X_X_X_X_X_XXXXXXX
___________X_X__XX_______
X_X___XX__XXXXX_X__X__X_X
_XX__X_XXXXXX__XX_XX_X__X
_X____X____X_XXXX__XX_X_X
_XX__X_XX_XXX__XXXX_XX___
__X_XXXX_XXX_XX_X_XXXX_X_
_____X_X_X___X__XXXX_XX_X
XXXXX_X_X_XX___XX_XXXXX_X
____XX__XXX__X_______X_XX
XX_X__X__XXXXX_XXXXXX__XX
_________X_X_X___X___X____
XXXXXXX_X___XXX_X_X_X___X
X_____X__X_XX_X_X___XX_XX
X_XXX_X___X__XXXXXXXXX_XX
X_XXX_X_____XX__X__XX_X_
X_XXX_X_XXXX_____X_XX_XXX
X_____X___X__X__XX_X_X___
XXXXXXX_XXXXXX__X_XXXX__X
```

CHAPTER 10

Segno package

10.1 Module contents

QR Code and Micro QR Code implementation.

“QR Code” and “Micro QR Code” are registered trademarks of DENSO WAVE INCORPORATED.

exception segno.**QRCodeError**
Generic QR Code error.

exception segno.**ErrorLevelError**
Indicates errors related to QR Code error correction level.

exception segno.**ModeError**
Indicates errors related to QR Code mode.

exception segno.**MaskError**
Indicates errors related to QR Code data mask.

exception segno.**VersionError**
Indicates errors related to the QR Code version.

exception segno.**DataOverflowError**
Indicates a problem that the provided data does not fit into the provided QR Code version or the data is too large in general.

class segno.**QRCode**(*code*)
Represents a (Micro) QR Code.

matrix_iter(*scale=1*, *border=None*)
Returns an iterator over the matrix which includes the border.

The border is returned as sequence of light modules. Dark modules are reported as `0x1`, light modules have the value `0x0`.

The following example converts the QR Code matrix into a list of lists which use boolean values for the modules (True = dark module, False = light module):

```
>>> import segno
>>> qr = segno.make('The Beatles')
>>> size = qr.symbol_size()[0]
>>> res = []
>>> # Scaling factor 2, default border
>>> for row in qr.matrix_iter(scale=2):
>>>     res.append([col == 0x1 for col in row])
>>> size * 2 == len(res[0])
True
```

Parameters

- **scale** (*int*) – The scaling factor (default: `1`).
- **border** (*int*) – The size of border / quiet zone or `None` to indicate the default border.

Raises `ValueError` if the scaling factor or the border is invalid (i.e. negative).

png_data_uri(***kw*)

Converts the QR Code into a PNG data URI.

Uses the same keyword parameters as the usual PNG serializer.

Return type str

save(*out*, *kind=None*, ***kw*)

Serializes the QR Code in one of the supported formats. The serialization format depends on the filename extension.

Common keywords

Name	Description
scale	Integer or float indicating the size of a single module. Default: 1. The interpretation of the scaling factor depends on the serializer. For pixel-based output (like PNG) the scaling factor is interepreted as pixel-size (1 = 1 pixel). EPS interprets `1` as 1 point (1/72 inch) per module. Some serializers (like SVG) accept float values. If the serializer does not accept float values, the value will be converted to an integer value (note: int(1.6) == 1).
border	Integer indicating the size of the quiet zone. If set to `None` (default), the recommended border size will be used (`4` for QR Codes, `2` for a Micro QR Codes).
color	A string or tuple representing a color value for the dark modules. The default value is "black". The color can be provided as `(R, G, B)` tuple, as web color name (like "red") or in hexadecimal format (`#RGB` or `#RRGGBB`). Some serializers (SVG and PNG) accept an alpha transparency value like `#RRGGBBAA`.
background	A string or tuple representing a color for the light modules or background. See "color" for valid values. The default value depends on the serializer. SVG uses no background color (`None`) by default, other serializers use "white" as default background color.

Scalable Vector Graphics (SVG)

Name	Description
out	Filename or io.BytesIO
kind	“svg” or “svgz” (to create a gzip compressed SVG)
scale	integer or float
color	Default: “#000” (black) `None` is a valid value. If set to `None`, the resulting path won’t have a “stroke” attribute. The “stroke” attribute may be defined via CSS (external). If an alpha channel is defined, the output depends of the used SVG version. For SVG versions >= 2.0, the “stroke” attribute will have a value like “rgba(R, G, B, A)”, otherwise the path gets another attribute “stroke-opacity” to emulate the alpha channel. To minimize the document size, the SVG serializer uses automatically the shortest color representation: If a value like “#000000” is provided, the resulting document will have a color value of “#000”. If the color is “#FF0000”, the resulting color is not “#F00”, but the web color name “red”.
background	Default value `None`. If this paramater is set to another value, the resulting image will have another path which is used to define the background color. If an alpha channel is used, the resulting path may have a “fill-opacity” attribute (for SVG version < 2.0) or the “fill” attribute has a “rgba(R, G, B, A)” value. See keyword “color” for further details.
xmldecl	Boolean value (default: `True`) indicating whether the document should have an XML declaration header. Set to `False` to omit the header.
svgns	Boolean value (default: `True`) indicating whether the document should have an explicit SVG namespace declaration. Set to `False` to omit the namespace declaration. The latter might be useful if the document should be embedded into a HTML 5 document where the SVG namespace is implicitly defined.
title	String (default: `None`) Optional title of the generated SVG document.
desc	String (default: `None`) Optional description of the generated SVG document.
svgid	A string indicating the ID of the SVG document (if set to `None` (default), the SVG element won’t have an ID).
svgclass	Default: “segno”. The CSS class of the SVG document (if set to `None`, the SVG element won’t have a class).
lineclass	Default: “qrline”. The CSS class of the path element (which draws the dark modules (if set to `None`, the path won’t have a class).
omitsize	Indicates if width and height attributes should be omitted (default: `False`). If these attributes are omitted, a `viewBox` attribute will be added to the document.
unit	Default: `None` Inidctaes the unit for width / height and other coordinates. By default, the unit is unspecified and all values are in the user space. Valid values: em, ex, px, pt, pc, cm, mm, in, and percentages (any string is accepted, this parameter is not validated by the serializer)
encoding	Encoding of the XML document. “utf-8” by default.
svgversion	SVG version (default: `None`). If specified (a float), the resulting document has an explicit “version” attribute. If set to `None`, the document won’t have a “version” attribute. This parameter is not validated.
compresslevel	Default: 9. This parameter is only valid, if a compressed SVG document should be created (file extension “svgz”). 1 is fastest and produces the least compression, 9 is slowest and produces the most. 0 is no compression.

Portable Network Graphics (PNG)

Name	Description
out	Filename or io.BytesIO
kind	“png”
scale	integer
color	Default: “#000” (black) `None` is a valid value iff background is not `None`.
background	Default value `#fff` (white) See keyword “color” for further details.
compresslevel	Default: 9. Integer indicating the compression level for the `IDAT` (data) chunk. 1 is fastest and produces the least compression, 9 is slowest and produces the most. 0 is no compression.
dpi	Default: None. Specifies the DPI value for the image. By default, the DPI value is unspecified. Please note that the DPI value is converted into meters (maybe with rounding errors) since PNG does not support the unit “dots per inch”.
addad	Boolean value (default: True) to (dis-)allow a “Software” comment indicating that the file was created by Segno.

Encapsulated PostScript (EPS)

Name	Description
out	Filename or io.StringIO
kind	“eps”
scale	integer or float
color	Default: “#000” (black)
background	Default value: `None` (no background)

Portable Document Format (PDF)

Name	Description
out	Filename or io.BytesIO
kind	“pdf”
scale	integer or float
compresslevel	Default: 9. Integer indicating the compression level. 1 is fastest and produces the least compression, 9 is slowest and produces the most. 0 is no compression.

Text (TXT)

Does not support the “scale” keyword!

Name	Description
out	Filename or io.StringIO
kind	“txt”
color	Default: “1”
background	Default: “0”

ANSI escape code

Supports the “border” keyword, only!

Name	Description
kind	“ans”

Portable Bitmap (PBM)

Name	Description
out	Filename or io.BytesIO
kind	“pbm”
scale	integer
plain	Default: False. Boolean to switch between the P4 and P1 format. If set to `True`, the (outdated) P1 serialization format is used.

Portable Arbitrary Map (PAM)

Name	Description
out	Filename or io.BytesIO
kind	“pam”
scale	integer
color	Default: “#000” (black).
background	Default value `#fff` (white). Use `None` for a transparent background.

LaTeX / PGF/TikZ

To use the output of this serializer, the `PGF/TikZ` (and optionally `hyperref`) package is required in the LaTeX environment. The serializer itself does not depend on any external packages.

Name	Description
out	Filename or io.StringIO
kind	“tex”
scale	integer or float
color	LaTeX color name (default: “black”). The color is written “at it is”, so ensure that the color is a standard color or it has been defined in the enclosing LaTeX document.
url	Default: `None`. Optional URL where the QR Code should point to. Requires the `hyperref` package in your LaTeX environment.

X BitMap (XBM)

Name	Description
out	Filename or io.StringIO
kind	“xbm”
scale	integer
name	Name of the variable (default: “img”)

X PixMap (XPM)

Name	Description
out	Filename or io.StringIO
kind	“xpm”
scale	integer
color	Default: “#000” (black).
background	Default value `#fff` (white) `None` indicates a transparent background.
name	Name of the variable (default: “img”)

Parameters

- **out** – A filename or a writable file-like object with a `name` attribute. Use the *kind* parameter if *out* is a `io.ByteIO` or `io.StringIO` stream which don't have a `name` attribute.
- **kind** – If the desired output format cannot be determined from the `out` parameter, this parameter can be used to indicate the serialization format (i.e. "svg" to enforce SVG output)
- **kw** – Any of the supported keywords by the specific serialization method.

show(*delete_after=20*, *scale=10*, *border=None*, *color=u'#000'*, *background=u'#fff'*)
Displays this QR code.

This method is mainly intended for debugging purposes.

This method saves the output of the `png()` method (by default with a scaling factor of 10) to a temporary file and opens it with the standard PNG viewer application or within the standard webbrowser. The temporary file is deleted afterwards (unless *delete_after* is set to `None`).

If this method does not show any result, try to increase the *delete_after* value or set it to `None`

Parameters **delete_after** – Time in seconds to wait till the temporary file is deleted.

svg_data_uri(*xmldecl=False*, *encode_minimal=False*, *omit_charset=False*, *nl=False*, ***kw*)
Converts the QR Code into a SVG data URI.

The XML declaration is omitted by default (set `xmldecl` to `True` to enable it), further the newline is omitted by default (set `nl` to `True` to enable it).

Aside from the missing `out` parameter and the different `xmldecl` and `nl` default values and the additional parameter `encode_minimal` and `omit_charset` this method uses the same parameters as the usual SVG serializer.

Parameters

- **xmldecl** (*bool*) – Indicates if the XML declaration should be serialized (default: `False`)
- **encode_minimal** (*bool*) – Indicates if the resulting data URI should use minimal percent encoding (disabled by default).
- **omit_charset** (*bool*) – Indicates if the `;charset=...` should be omitted (disabled by default)

Return type str

symbol_size(*scale=1*, *border=None*)
Returns the symbol size (width x height) with the provided border and scaling factor.

Parameters

- **scale** (*int or float*) – Indicates the size of a single module (default: 1). The size of a module depends on the used output format; i.e. in a PNG context, a scaling factor of 2 indicates that a module has a size of 2 x 2 pixel. Some outputs (i.e. SVG) accept floating point values.
- **border** (*int*) – The border size or `None` to specify the default quiet zone (4 for QR Codes, 2 for Micro QR Codes).

Return type tuple (width, height)

terminal(*out=None*, *border=None*)
Serializes the matrix as ANSI escape code.

Parameters

- **out** – Filename or a file-like object supporting to write text. If `None` (default), the matrix is written to `sys.stdout`.
- **border** (*int*) – Integer indicating the size of the quiet zone. If set to `None` (default), the recommended border size will be used (`4` for QR Codes, `2` for a Micro QR Codes).

default_border_size
Indicates the default border size aka quiet zone.

designator
Returns the version and error correction level as string *V-E* where *V* represents the version number and *E* the error level.

error
Error correction level; either a string ("L", "M", "Q", "H") or `None` if the QR Code provides no error correction (Micro QR Code version M1)

is_micro
Indicates if this QR Code is a Micro QR Code

mask = None
Returns the data mask pattern reference (an integer).

matrix = None
Returns the matrix (tuple of bytearrays).

mode
String indicating the mode ("numeric", "alphanumeric", "byte", "kanji"). May be `None` if multiple modes are used.

version
(Micro) QR Code version. Either a string ("M1", "M2", "M3", "M4") or an integer in the range of 1 .. 40.

class segno.**QRCodeSequence**
Represents a sequence of 1 .. n (max. n = 16) *QRCode* instances.

Iff this sequence represents only one item, it behaves like *QRCode*.

save(*out*, *kind=None*, ***kw*)
Saves the sequence of QR Code to *out*.

If *out* is a filename, this method modifies the filename and adds `<Number of QR Codes>-<Current QR Code>` to it. `structured-append.svg` becomes (if the sequence contains two QR Codes): `structured-append-02-01.svg` and `structured-append-02-02.svg`

Please note that using a file or file-like object may result into an invalid serialization format since all QR Codes are written to the same output.

See *QRCode.save()* for a detailed enumeration of options.

terminal(*out=None*, *border=None*)
Serializes the sequence of QR Codes as ANSI escape code.

See *QRCode.terminal()* for details.

segno.**make**(*content*, *error=None*, *version=None*, *mode=None*, *mask=None*, *encoding=None*, *eci=False*, *micro=None*, *boost_error=True*)
Creates a (Micro) QR Code.

This is main entry point to create QR Codes and Micro QR Codes.

Aside from *content*, all parameters are optional and an optimal (minimal) (Micro) QR Code with a maximal error correction level (minimum "M") is generated.

Parameters

- **content** (*str, int, bytes*) – The data to encode. Either a Unicode string, an integer or bytes. If bytes are provided, the *encoding* parameter should be used to specify the used encoding.
- **error** (*str, unicode or None*) – Error correction level. If `None` (default), error correction level `L` is used (note: Micro QR Code version M1 does not support error correction. If an explicit error level is used, a M1 QR Code won't be generated). Valid values: `None` (allowing generation of M1 codes or use error correction level "L" or better see `boost_error`), "L", "M", "Q", "H" (error correction level "H" isn't available for Micro QR Codes).

Error correction level	Error correction capability
L (Segno's default unless version M1)	recovers 7% of data
M	recovers 15% of data
Q	recovers 25% of data
H (not available for Micro QR Codes)	recovers 30% of data

  Higher error levels may require larger QR Codes (see also *version* parameter).

  The *error* parameter is case insensitive.

  See also the *boost_error* parameter.

- **version** (*int, str, unicode or None.*) – QR Code version. If the value is `None` (default), the minimal version which fits for the input data will be used. Valid values: "M1", "M2", "M3", "M4" (for Micro QR Codes) or an integer between 1 and 40 (for QR Codes). The *version* parameter is case insensitive.
- **mode** (*str, unicode, or None*) – "numeric", "alphanumeric", "byte", or "kanji". If the value is `None` (default) the appropriate mode will automatically be determined. If *version* refers a to Micro QR Code, this function may raise a *ModeError* if the provided *mode* is not supported.

Mode	(Micro) QR Code Version
numeric	1 - 40, M1, M2, M3, M4
alphanumeric	1 - 40, M2, M3, M4
byte	1 - 40, M3, M4
kanji	1 - 40, M3, M4

  The *mode* parameter is case insensitive.

- **mask** (*int or None*) – Data mask. If the value is `None` (default), the appropriate data mask is choosen automatically. If the *mask* parameter if provided, this function may raise a *MaskError* if the mask is invalid.
- **encoding** (*unicode | str | None*) – Indicates the encoding in mode "byte". By default (*encoding* is `None`) the implementation tries to use the standard conform ISO/IEC 8859-1 encoding and if it does not fit, it will use UTF-8. Note that no ECI mode indicator is inserted by default (see *eci*). The *encoding* parameter is case insensitive.
- **eci** (*bool*) – Indicates if binary data which does not use the default encoding (ISO/IEC 8859-1) should enforce the ECI mode. Since a lot of QR Code readers do not support the ECI mode, this feature is disabled by default and the data is encoded in the provided *encoding* using the usual "byte" mode. Set *eci* to `True` if an ECI header should be inserted into the QR Code. Note that the implementation may not know the ECI designator for the

provided *encoding* and may raise an exception if the ECI designator cannot be found. The ECI mode is not supported by Micro QR Codes.

- **micro** (*bool or None*) – If *version* is `None` this parameter can be used to allow the creation of a Micro QR Code. If set to `False`, a QR Code is generated. If set to `None` (default) a Micro QR Code may be generated if applicable. If set to `True` the algorithm generates a Micro QR Code or raises an exception if the *mode* is not compatible or the *content* is too large for Micro QR Codes.
- **boost_error** (*bool*) – Indicates if the error correction level may be increased if it does not affect the version (default: `True`). If set to `True`, the `error` parameter is interpreted as minimum error level. If set to `False`, the resulting (Micro) QR Code uses the provided `error` level (or the default error correction level, if error is `None`)

Raises *QRCodeError*: In case of a problem. In fact, it's more likely that a derived exception is thrown: *ModeError*: In case of problems with the mode (i.e. invalid mode or invalid *mode* / *version* combination. *VersionError*: In case the *version* is invalid or the *micro* parameter contradicts the provided *version*. *ErrorLevelError*: In case the error level is invalid or the error level is not supported by the provided *version*. *DataOverflowError*: In case the data does not fit into a (Micro) QR Code or it does not fit into the provided *version*. *MaskError*: In case an invalid data mask was specified.

Return type *QRCode*

segno.**make_qr**(*content*, *error=None*, *version=None*, *mode=None*, *mask=None*, *encoding=None*, *eci=False*, *boost_error=True*)

Creates a QR Code (never a Micro QR Code).

See *make()* for a description of the parameters.

Return type *QRCode*

segno.**make_micro**(*content*, *error=None*, *version=None*, *mode=None*, *mask=None*, *encoding=None*, *boost_error=True*)

Creates a Micro QR Code.

See *make()* for a description of the parameters.

Note: Error correction level "H" isn't available for Micro QR Codes. If used, this function raises a *segno.ErrorLevelError*.

Return type *QRCode*

segno.**make_sequence**(*content*, *error=None*, *version=None*, *mode=None*, *mask=None*, *encoding=None*, *boost_error=True*, *symbol_count=None*)

Creates a sequence of QR Codes.

If the content fits into one QR Code and neither `version` nor `symbol_count` is provided, this function may return a sequence with one QR Code which does not use the Structured Append mode. Otherwise a sequence of 2 .. n (max. n = 16) QR Codes is returned which use the Structured Append mode.

The Structured Append mode allows to split the content over a number (max. 16) QR Codes.

The Structured Append mode isn't available for Micro QR Codes, therefor the returned sequence contains QR Codes, only.

Since this function returns an iterable object, it may be used as follows:

```
for i, qrcode in enumerate(segno.make_sequence(data, symbol_count=2)):
    qrcode.save('seq-%d.svg' % i, scale=10, color='darkblue')
```

The returned number of QR Codes is determined by the *version* or *symbol_count* parameter

See *make()* for a description of the other parameters.

Parameters **symbol_count** (*int*) – Number of symbols.

Return type *QRCodeSequence*

10.2 Utilities

Utility functions useful for writers or QR Code objects.

segno.utils.**check_valid_border**(*border*)
Raises a ValueError iff *border* is negative.

Parameters **border** (*int*) – Indicating the size of the quiet zone.

segno.utils.**check_valid_scale**(*scale*)
Raises a `ValueError` iff *scale* is negative or zero.

Parameters **scale** – float or integer indicating a scaling factor.

segno.utils.**get_border**(*version*, *border*)
Returns *border* if not `None`, otherwise the default border size for the provided QR Code.

Parameters

- **version** (*int*) – 1 .. 40 or a Micro QR Code version constant
- **border** (*int*) – The size of the quiet zone or `None`.

Return type int

segno.utils.**get_default_border_size**(*version*)
Returns the default border size (quiet zone) for the provided version.

Parameters **version** (*int*) – 1 .. 40 or a Micro QR Code version constant.

Return type int

segno.utils.**get_symbol_size**(*version*, *scale=1*, *border=None*)
Returns the symbol size (width x height) with the provided border and scaling factor.

Parameters

- **version** (*int*) – A version constant.
- **scale** (*int or float*) – Indicates the size of a single module (default: 1). The size of a module depends on the used output format; i.e. in a PNG context, a scaling factor of 2 indicates that a module has a size of 2 x 2 pixel. Some outputs (i.e. SVG) accept floating point values.
- **border** (*int*) – The border size or `None` to specify the default quiet zone (4 for QR Codes, 2 for Micro QR Codes).

Return type tuple (width, height)

segno.utils.**matrix_iter**(*matrix*, *version*, *scale=1*, *border=None*)
Returns an iterator / generator over the provided matrix which includes the border and the scaling factor.

If either the *scale* or *border* value is invalid, a `ValueError` is raised.

Parameters

- **matrix** – An iterable of bytearrays.
- **version** (*int*) – A version constant.

- **scale** (*int*) – The scaling factor (default: 1).
- **border** (*int*) – The border size or None to specify the default quiet zone (4 for QR Codes, 2 for Micro QR Codes).

Raises ValueError if an illegal scale or border value is provided

segno.utils.**matrix_iter_detail**(*matrix*, *version*, *scale=1*, *border=None*)

Returns an iterator / generator over the provided matrix which includes the border and the scaling factor.

This iterator / generator returns different values for dark / light modules and therefor the different parts (like the finder patterns, alignment patterns etc.) are distinguishable. If this information isn't necessary, use the *matrix_iter()* function because it is much cheaper and faster.

If either the *scale* or *border* value is invalid, a py:exc:*ValueError* is raised.

Parameters

- **matrix** – An iterable of bytearrays.
- **version** (*int*) – A version constant.
- **scale** (*int*) – The scaling factor (default: 1).
- **border** (*int*) – The border size or None to specify the default quiet zone (4 for QR Codes, 2 for Micro QR Codes).

Raises ValueError if an illegal scale or border value is provided

segno.utils.**matrix_to_lines**(*matrix*, *x*, *y*, *incby=1*)

Converts the *matrix* into an iterable of ((x1, y1), (x2, y2)) tuples which represent a sequence (horizontal line) of dark modules.

The path starts at the 1st row of the matrix and moves down to the last row.

Parameters

- **matrix** – An iterable of bytearrays.
- **x** – Initial position on the x-axis.
- **y** – Initial position on the y-axis.
- **incby** – Value to move along the y-axis (default: 1).

Return type iterable of (x1, y1), (x2, y2) tuples

segno.utils.**TYPE_ALIGNMENT_PATTERN_DARK = 2560**

Dark alignment pattern module.

segno.utils.**TYPE_ALIGNMENT_PATTERN_LIGHT = 10**

Light alignment pattern module.

segno.utils.**TYPE_DARKMODULE = 512**

A single dark module which occurs in QR Codes (but not in Micro QR Codes).

segno.utils.**TYPE_DATA_DARK = 1024**

Dark module in the encoding area (either a data module or an error correction module).

segno.utils.**TYPE_DATA_LIGHT = 4**

Light module in the encoding area (either a data module or an error correction module).

segno.utils.**TYPE_FINDER_PATTERN_DARK = 1536**

Dark finder module.

segno.utils.**TYPE_FINDER_PATTERN_LIGHT = 6**

Light finder module

segno.utils.**TYPE_FORMAT_DARK = 3584**
Dark format information module.

segno.utils.**TYPE_FORMAT_LIGHT = 14**
Light format information module.

segno.utils.**TYPE_QUIET_ZONE = 18**
Border of light modules.

segno.utils.**TYPE_SEPARATOR = 8**
Separator around the finder patterns (light module)

segno.utils.**TYPE_TIMING_DARK = 3072**
Dark timing patten module.

segno.utils.**TYPE_TIMING_LIGHT = 12**
Light timing pattern module.

segno.utils.**TYPE_VERSION_DARK = 4096**
Dark version information module.

segno.utils.**TYPE_VERSION_LIGHT = 16**
Light version information module.

10.3 High level QR Code factories

Additional factory functions for common QR Codes.

The factory functions which return a QR Code with the minimum error correction level "L" (or better). To create a (Micro) QR Code which should use a specific error correction level or version etc., use the "_data" factory functions which return a string which can be used as input for *segno.make()*.

segno.helpers.**make_email**(*to*, *cc=None*, *bcc=None*, *subject=None*, *body=None*)
Encodes either a simple e-mail address or a complete message with (blind) carbon copies and a subject and a body.

Parameters

- **to** (*str|iterable*) – The email address (recipient). Multiple values are allowed.
- **cc** (*str|iterable|None*) – The carbon copy recipient. Multiple values are allowed.
- **bcc** (*str|iterable|None*) – The blind carbon copy recipient. Multiple values are allowed.
- **subject** (*str|None*) – The subject.
- **body** (*str|None*) – The message body.

segno.helpers.**make_geo**(*lat*, *lng*)
Returns a QR Code which encodes geographic location using the `geo` URI scheme.

Parameters

- **lat** (*float*) – Latitude
- **lng** (*float*) – Longitude

Return type *segno.QRCode*

segno.helpers.**make_geo_data**(*lat*, *lng*)
Creates a geo location URI.

Parameters

- **lat** (*float*) – Latitude
- **lng** (*float*) – Longitude

Return type str

segno.helpers.**make_make_email_data**(*to*, *cc=None*, *bcc=None*, *subject=None*, *body=None*)

Creates either a simple "mailto:" URL or complete e-mail message with (blind) carbon copies and a subject and a body.

Parameters

- **to** (*str|iterable*) – The email address (recipient). Multiple values are allowed.
- **cc** (*str|iterable|None*) – The carbon copy recipient. Multiple values are allowed.
- **bcc** (*str|iterable|None*) – The blind carbon copy recipient. Multiple values are allowed.
- **subject** (*str|None*) – The subject.
- **body** (*str|None*) – The message body.

segno.helpers.**make_mecard**(*name*, *reading=None*, *email=None*, *phone=None*, *videophone=None*, *memo=None*, *nickname=None*, *birthday=None*, *url=None*, *pobox=None*, *roomno=None*, *houseno=None*, *city=None*, *prefecture=None*, *zipcode=None*, *country=None*)

Returns a QR Code which encodes a MeCard

Parameters

- **name** (*str*) – Name. If it contains a comma, the first part is treated as lastname and the second part is treated as forename.
- **reading** (*str|None*) – Designates a text string to be set as the kana name in the phonebook
- **email** (*str|iterable*) – E-mail address. Multiple values are allowed.
- **phone** (*str|iterable*) – Phone number. Multiple values are allowed.
- **videophone** (*str|iterable*) – Phone number for video calls. Multiple values are allowed.
- **memo** (*str*) – A notice for the contact.
- **nickname** (*str*) – Nickname.
- **birthday** (*str|int|date*) – Birthday. If a string is provided, it should encode the date as YYYYMMDD value.
- **url** (*str|iterable*) – Homepage. Multiple values are allowed.
- **pobox** (*str|None*) – P.O. box (address information).
- **roomno** (*str|None*) – Room number (address information).
- **houseno** (*str|None*) – House number (address information).
- **city** (*str|None*) – City (address information).
- **prefecture** (*str|None*) – Prefecture (address information).
- **zipcode** (*str|None*) – Zip code (address information).
- **country** (*str|None*) – Country (address information).

Return type *segno.QRCode*

segno.helpers.**make_mecard_data**(*name*, *reading=None*, *email=None*, *phone=None*, *videophone=None*, *memo=None*, *nickname=None*, *birthday=None*, *url=None*, *pobox=None*, *roomno=None*, *houseno=None*, *city=None*, *prefecture=None*, *zipcode=None*, *country=None*)

Creates a string encoding the contact information as MeCard.

Parameters

- **name** (*str*) – Name. If it contains a comma, the first part is treated as lastname and the second part is treated as forename.
- **reading** (*str|None*) – Designates a text string to be set as the kana name in the phonebook
- **email** (*str|iterable*) – E-mail address. Multiple values are allowed.
- **phone** (*str|iterable*) – Phone number. Multiple values are allowed.
- **videophone** (*str|iterable*) – Phone number for video calls. Multiple values are allowed.
- **memo** (*str*) – A notice for the contact.
- **nickname** (*str*) – Nickname.
- **birthday** (*(str|int|date)*) – Birthday. If a string is provided, it should encode the date as YYYYMMDD value.
- **url** (*str|iterable*) – Homepage. Multiple values are allowed.
- **pobox** (*str|None*) – P.O. box (address information).
- **roomno** (*str|None*) – Room number (address information).
- **houseno** (*str|None*) – House number (address information).
- **city** (*str|None*) – City (address information).
- **prefecture** (*str|None*) – Prefecture (address information).
- **zipcode** (*str|None*) – Zip code (address information).
- **country** (*str|None*) – Country (address information).

Return type str

segno.helpers.**make_vcard**(*name*, *displayname*, *email=None*, *phone=None*, *fax=None*, *videophone=None*, *memo=None*, *nickname=None*, *birthday=None*, *url=None*, *pobox=None*, *street=None*, *city=None*, *region=None*, *zipcode=None*, *country=None*, *org=None*, *lat=None*, *lng=None*, *source=None*, *rev=None*, *title=None*)

Creates a QR Code which encodes a vCard version 3.0.

Only a subset of available vCard properties is supported.

Parameters

- **name** (*str*) – The name. If it contains a semicolon, , the first part is treated as lastname and the second part is treated as forename.
- **displayname** (*str*) – Common name.
- **email** (*str|iterable*) – E-mail address. Multiple values are allowed.
- **phone** (*str|iterable*) – Phone number. Multiple values are allowed.

- **fax** (*str|iterable*) – Fax number. Multiple values are allowed.
- **videophone** (*str|iterable*) – Phone number for video calls. Multiple values are allowed.
- **memo** (*str*) – A notice for the contact.
- **nickname** (*str*) – Nickname.
- **birthday** (*str|date*) – Birthday. If a string is provided, it should encode the date as YYYY-MM-DD value.
- **url** (*str|iterable*) – Homepage. Multiple values are allowed.
- **pobox** (*str|None*) – P.O. box (address information).
- **street** (*str|None*) – Street address.
- **city** (*str|None*) – City (address information).
- **region** (*str|None*) – Region (address information).
- **zipcode** (*str|None*) – Zip code (address information).
- **country** (*str|None*) – Country (address information).
- **org** (*str*) – Company / organization name.
- **lat** (*float*) – Latitude.
- **lng** (*float*) – Longitude.
- **source** (*str*) – URL where to obtain the vCard.
- **rev** (*str|date*) – Revision of the vCard / last modification date.
- **title** (*str|iterable|None*) – Job Title. Multiple values are allowed.

Return type *segno.QRCode*

segno.helpers.**make_vcard_data**(*name*, *displayname*, *email=None*, *phone=None*, *fax=None*, *videophone=None*, *memo=None*, *nickname=None*, *birthday=None*, *url=None*, *pobox=None*, *street=None*, *city=None*, *region=None*, *zipcode=None*, *country=None*, *org=None*, *lat=None*, *lng=None*, *source=None*, *rev=None*, *title=None*, *photo_uri=None*)

Creates a string encoding the contact information as vCard 3.0.

Only a subset of available vCard properties is supported.

Parameters

- **name** (*str*) – The name. If it contains a semicolon, , the first part is treated as lastname and the second part is treated as forename.
- **displayname** (*str*) – Common name.
- **email** (*str|iterable*) – E-mail address. Multiple values are allowed.
- **phone** (*str|iterable*) – Phone number. Multiple values are allowed.
- **fax** (*str|iterable*) – Fax number. Multiple values are allowed.
- **videophone** (*str|iterable*) – Phone number for video calls. Multiple values are allowed.
- **memo** (*str*) – A notice for the contact.
- **nickname** (*str*) – Nickname.

- **birthday** (*str|date*) – Birthday. If a string is provided, it should encode the date as YYYY-MM-DD value.
- **url** (*str|iterable*) – Homepage. Multiple values are allowed.
- **pobox** (*str|None*) – P.O. box (address information).
- **street** (*str|None*) – Street address.
- **city** (*str|None*) – City (address information).
- **region** (*str|None*) – Region (address information).
- **zipcode** (*str|None*) – Zip code (address information).
- **country** (*str|None*) – Country (address information).
- **org** (*str*) – Company / organization name.
- **lat** (*float*) – Latitude.
- **lng** (*float*) – Longitude.
- **source** (*str*) – URL where to obtain the vCard.
- **rev** (*str|date*) – Revision of the vCard / last modification date.
- **title** (*str|iterable|None*) – Job Title. Multiple values are allowed.
- **photo_uri** (*str|iterable|None*) – Photo URI. Multiple values are allowed.

Return type str

segno.helpers.**make_wifi**(*ssid*, *password*, *security*, *hidden=False*)

Creates a WIFI configuration QR Code.

Parameters

- **ssid** (*str*) – The SSID of the network.
- **password** (*str|None*) – The password.
- **security** (*str|None*) – Authentication type; the value should be "WEP" or "WPA". Set to `None` to omit the value. "nopass" is equivalent to setting the value to `None` but in the former case, the value is not omitted.
- **hidden** (*bool*) – Indicates if the network is hidden (default: `False`)

Return type *segno.QRCode*

segno.helpers.**make_wifi_data**(*ssid*, *password*, *security*, *hidden=False*)

Creates WIFI configuration string.

Parameters

- **ssid** (*str*) – The SSID of the network.
- **password** (*str|None*) – The password.
- **security** (*str|None*) – Authentication type; the value should be "WEP" or "WPA". Set to `None` to omit the value. "nopass" is equivalent to setting the value to `None` but in the former case, the value is not omitted.
- **hidden** (*bool*) – Indicates if the network is hidden (default: `False`)

Return type str

CHAPTER 11

Segno's low level API

segno.make(), *segno.make_qr()*, and *segno.make_micro()* return a *segno.QRCode* which implements almost no logic but uses the result of *segno.encoder.encode()* glued together with the functionality of *segno.writers* to provide a simple (supposed to be user-friendly) API.

A completely different API is possible by utilizing *segno.encoder.encode()* which returns just a tuple: `(matrix, version, error, mask, segments)`. The module *segno.writers* is independent of the *segno.encoder* module and vice versa.

11.1 segno.encoder

QR Code and Micro QR Code encoder.

"QR Code" and "Micro QR Code" are registered trademarks of DENSO WAVE INCORPORATED.

exception segno.encoder.**QRCodeError**
Generic QR Code error.

exception segno.encoder.**VersionError**
Indicates errors related to the QR Code version.

exception segno.encoder.**ModeError**
Indicates errors related to QR Code mode.

exception segno.encoder.**ErrorLevelError**
Indicates errors related to QR Code error correction level.

exception segno.encoder.**MaskError**
Indicates errors related to QR Code data mask.

exception segno.encoder.**DataOverflowError**
Indicates a problem that the provided data does not fit into the provided QR Code version or the data is too large in general.

segno.encoder.**encode**(*content*, *error=None*, *version=None*, *mode=None*, *mask=None*, *encoding=None*, *eci=False*, *micro=None*, *boost_error=True*)

Creates a (Micro) QR Code.

See *segno.make()* for a detailed description of the parameters.

Contrary to `make` this function returns a named tuple: `(matrix, version, error, mask, segments)`

Note that `version` is always an integer referring to the values of the `segno.consts` constants. `error` is `None` iff a M1 QR Code was generated, otherwise it is always an integer.

Return type namedtuple

segno.encoder.**encode_sequence**(*content*, *error=None*, *version=None*, *mode=None*, *mask=None*, *encoding=None*, *eci=False*, *boost_error=True*, *symbol_count=None*)

EXPERIMENTAL: Creates a sequence of QR Codes in Structured Append mode.

Returns Iterable of named tuples, see *encode()* for details.

11.2 segno.writers

Standard serializers and utility functions for serializers.

The serializers are independent of the *segno.QRCode* (and the `segno.encoder.Code`) class; they just need a matrix (tuple of bytearrays) and the version constant.

segno.writers.**as_png_data_uri**(*matrix*, *version*, *scale=1*, *border=None*, *color=u'#000'*, *background=u'#fff'*, *compresslevel=9*, *addad=True*)

Converts the provided matrix into a PNG data URI.

Return type str

segno.writers.**as_svg_data_uri**(*matrix*, *version*, *scale=1*, *border=None*, *color=u'#000'*, *background=None*, *xmldecl=False*, *svgns=True*, *title=None*, *desc=None*, *svgid=None*, *svgclass=u'segno'*, *lineclass=u'qrline'*, *omitsize=False*, *unit=u''*, *encoding=u'utf-8'*, *svgversion=None*, *nl=False*, *encode_minimal=False*, *omit_charset=False*)

Converts the matrix to a SVG data URI.

The XML declaration is omitted by default (set `xmldecl` to `True` to enable it), further the newline is omitted by default (set `nl` to `True` to enable it).

Aside from the missing `out` parameter and the different `xmldecl` and `nl` default values and the additional parameter `encode_minimal` and `omit_charset` this function uses the same parameters as the usual SVG serializer.

Parameters

- **encode_minimal** (*bool*) – Indicates if the resulting data URI should use minimal percent encoding (disabled by default).
- **omit_charset** (*bool*) – Indicates if the `;charset=...` should be omitted (disabled by default)

Return type str

segno.writers.**save**(*matrix*, *version*, *out*, *kind=None*, ***kw*)

Serializes the matrix in any of the supported formats.

Parameters

- **matrix** – The matrix to serialize.
- **version** (*int*) – The (Micro) QR code version
- **out** – A filename or a writable file-like object with a `name` attribute. If a stream like `io.ByteIO` or `io.StringIO` object without a `name` attribute is provided, use the *kind* parameter to specify the serialization format.
- **kind** – If the desired output format cannot be extracted from the filename, this parameter can be used to indicate the serialization format (i.e. "svg" to enforce SVG output)
- **kw** – Any of the supported keywords by the specific serialization method.

`segno.writers.`**`writable`**(**args*, ***kwds*)

Returns a writable file-like object.

Usage:

```
with writable(file_name_or_path, 'wb') as f:
    ...
```

Parameters

- **file_or_path** – Either a file-like object or a filename.
- **mode** (*str*) – String indicating the writing mode (i.e. `'wb'`)

`segno.writers.`**`write_eps`**(*matrix*, *version*, *out*, *scale=1*, *border=None*, *color=u'#000'*, *background=None*)

Serializes the QR Code as EPS document.

Parameters

- **matrix** – The matrix to serialize.
- **version** (*int*) – The (Micro) QR code version
- **out** – Filename or a file-like object supporting to write strings.
- **scale** – Indicates the size of a single module (default: 1 which corresponds to 1 point (1/72 inch) per module).
- **border** (*int*) – Integer indicating the size of the quiet zone. If set to `None` (default), the recommended border size will be used (`4` for QR Codes, `2` for a Micro QR Codes).
- **color** – Color of the modules (default: black). The color can be provided as `(R, G, B)` tuple (this method acceppts floats as R, G, B values), as web color name (like "red") or in hexadecimal format (`#RGB` or `#RRGGBB`).
- **background** – Optional background color (default: `None` = no background color). See *color* for valid values.

`segno.writers.`**`write_pam`**(*matrix*, *version*, *out*, *scale=1*, *border=None*, *color=u'#000'*, *background=u'#fff'*)

Serializes the matrix as PAM image.

Parameters

- **matrix** – The matrix to serialize.
- **version** (*int*) – The (Micro) QR code version
- **out** – Filename or a file-like object supporting to write binary data.

- **scale** – Indicates the size of a single module (default: 1 which corresponds to 1 x 1 pixel per module).
- **border** (*int*) – Integer indicating the size of the quiet zone. If set to `None` (default), the recommended border size will be used (`4` for QR Codes, `2` for a Micro QR Codes).
- **color** – Color of the modules (default: black). The color can be provided as `(R, G, B)` tuple, as web color name (like "red") or in hexadecimal format (`#RGB` or `#RRGGBB`).
- **background** – Optional background color (default: white). See *color* for valid values. In addition, `None` is accepted which indicates a transparent background.

`segno.writers.`**write_pbm**(*matrix*, *version*, *out*, *scale=1*, *border=None*, *plain=False*)

Serializes the matrix as PBM image.

Parameters

- **matrix** – The matrix to serialize.
- **version** (*int*) – The (Micro) QR code version
- **out** – Filename or a file-like object supporting to write binary data.
- **scale** – Indicates the size of a single module (default: 1 which corresponds to 1 x 1 pixel per module).
- **border** (*int*) – Integer indicating the size of the quiet zone. If set to `None` (default), the recommended border size will be used (`4` for QR Codes, `2` for a Micro QR Codes).
- **plain** (*bool*) – Indicates if a P1 (ASCII encoding) image should be created (default: False). By default a (binary) P4 image is created.

`segno.writers.`**write_pdf**(*matrix*, *version*, *out*, *scale=1*, *border=None*, *color=u'#000'*, *background=None*, *compresslevel=9*)

Serializes the QR Code as PDF document.

Parameters

- **matrix** – The matrix to serialize.
- **version** (*int*) – The (Micro) QR code version
- **out** – Filename or a file-like object supporting to write bytes.
- **scale** – Indicates the size of a single module (default: 1 which corresponds to 1 x 1 pixel per module).
- **border** (*int*) – Integer indicating the size of the quiet zone. If set to `None` (default), the recommended border size will be used (`4` for QR Codes, `2` for a Micro QR Codes).
- **color** – Color of the modules (default: black). The color can be provided as `(R, G, B)` tuple, as web color name (like "red") or in hexadecimal format (`#RGB` or `#RRGGBB`).
- **background** – Optional background color (default: `None` = no background color). See *color* for valid values.
- **compresslevel** (*int*) – Integer indicating the compression level (default: 9). 1 is fastest and produces the least compression, 9 is slowest and produces the most. 0 is no compression.

`segno.writers.`**write_png**(*matrix*, *version*, *out*, *scale=1*, *border=None*, *color=u'#000'*, *background=u'#fff'*, *compresslevel=9*, *dpi=None*, *addad=True*)

Serializes the QR Code as PNG image.

By default, the generated PNG will be a greyscale image with a bitdepth of 1. If different colors are provided, an indexed-color image with the same bitdepth is generated.

Parameters

- **matrix** – The matrix to serialize.
- **version** (*int*) – The (Micro) QR code version
- **out** – Filename or a file-like object supporting to write bytes.
- **scale** – Indicates the size of a single module (default: 1 which corresponds to 1 x 1 pixel per module).
- **border** (*int*) – Integer indicating the size of the quiet zone. If set to `None` (default), the recommended border size will be used (`4` for QR Codes, `2` for a Micro QR Codes).
- **color** – Color of the modules (default: black). The color can be provided as `(R, G, B)` tuple, as web color name (like "red") or in hexadecimal format (`#RGB` or `#RRGGBB`).
- **background** – Optional background color (default: white). See *color* for valid values. In addition, `None` is accepted which indicates a transparent background.
- **dpi** (*int*) – Optional DPI setting. By default (`None`), the PNG won't have any DPI information. Note that the DPI value is converted into meters since PNG does not support any DPI information.
- **compresslevel** (*int*) – Integer indicating the compression level (default: 9). 1 is fastest and produces the least compression, 9 is slowest and produces the most. 0 is no compression.

segno.writers.**write_svg**(*matrix*, *version*, *out*, *scale=1*, *border=None*, *color=u'#000'*, *background=None*, *xmldecl=True*, *svgns=True*, *title=None*, *desc=None*, *svgid=None*, *svgclass=u'segno'*, *lineclass=u'qrline'*, *omitsize=False*, *unit=None*, *encoding=u'utf-8'*, *svgversion=None*, *nl=True*)

Serializes the QR Code as SVG document.

Parameters

- **matrix** – The matrix to serialize.
- **version** (*int*) – The (Micro) QR code version
- **out** – Filename or a file-like object supporting to write bytes.
- **scale** – Indicates the size of a single module (default: 1 which corresponds to 1 x 1 pixel per module).
- **border** (*int*) – Integer indicating the size of the quiet zone. If set to `None` (default), the recommended border size will be used (`4` for QR Codes, `2` for a Micro QR Codes).
- **color** – Color of the modules (default: `#000`). Any value which is supported by SVG can be used. In addition, `None` is a valid value. The resulting path won't have a `stroke` attribute.
- **background** – Optional background color (default: `None` = no background color). See *color* for valid values.
- **xmldecl** (*bool*) – Inidcates if the XML declaration header should be written (default: `True`)
- **svgns** (*bool*) – Indicates if the SVG namespace should be written (default: `True`).
- **title** (*str*) – Optional title of the generated SVG document.
- **desc** (*str*) – Optional description of the generated SVG document.
- **svgid** – The ID of the SVG document (if set to `None` (default), the SVG element won't have an ID).

- **svgclass** – The CSS class of the SVG document (if set to `None`, the SVG element won't have a class).
- **lineclass** – The CSS class of the path element (which draws the "black" modules (if set to `None`, the path won't have a class).
- **omitsize** (*bool*) – Indicates if width and height attributes should be omitted (default: `False`). If these attributes are omitted, a `viewBox` attribute will be added to the document.
- **unit** (*str*) – Unit for width / height and other coordinates. By default, the unit is unspecified and all values are in the user space. Valid values: em, ex, px, pt, pc, cm, mm, in, and percentages
- **encoding** (*str*) – Encoding of the XML document. "utf-8" by default.
- **svgversion** (*float*) – SVG version (default: None)
- **nl** (*bool*) – Indicates if the document should have a trailing newline (default: `True`)

segno.writers.**write_svg_debug**(*matrix*, *version*, *out*, *scale=15*, *border=None*, *fallback_color=u'fuchsia'*, *color_mapping=None*, *add_legend=True*)

Internal SVG serializer which is useful to debugging purposes.

This function is not exposed to the QRCode class by intention and the resulting SVG document is very inefficient (lots of <rect/>s). Dark modules are black and light modules are white by default. Provide a custom *color_mapping* to override these defaults. Unknown modules are red by default.

Parameters

- **matrix** – The matrix
- **version** – Version constant
- **out** – binary file-like object or file name
- **scale** – Scaling factor
- **border** – Quiet zone
- **fallback_color** – Color which is used for modules which are not 0x0 or 0x1 and for which no entry in *color_mapping* is defined.
- **color_mapping** – dict of module values to color mapping (optional)
- **add_legend** (*bool*) – Indicates if the bit values should be added to the matrix (default: True)

segno.writers.**write_terminal**(*matrix*, *version*, *out*, *border=None*)

Function to write to a terminal which supports ANSI escape codes.

Parameters

- **matrix** – The matrix to serialize.
- **version** (*int*) – The (Micro) QR code version.
- **out** – Filename or a file-like object supporting to write text.
- **border** (*int*) – Integer indicating the size of the quiet zone. If set to `None` (default), the recommended border size will be used (`4` for QR Codes, `2` for a Micro QR Codes).

segno.writers.**write_terminal_win**(*matrix*, *version*, *border=None*)

Function to write a QR Code to a MS Windows terminal.

Parameters

- **matrix** – The matrix to serialize.
- **version** (*int*) – The (Micro) QR code version
- **border** (*int*) – Integer indicating the size of the quiet zone. If set to `None` (default), the recommended border size will be used (`4` for QR Codes, `2` for a Micro QR Codes).

`segno.writers.`**write_tex**(*matrix*, *version*, *out*, *scale=1*, *border=None*, *color=u'black'*, *unit=u'pt'*, *url=None*)

Serializes the matrix as LaTeX PGF picture.

Requires the PGF/TikZ package (i.e. `\usepackage{pgf}`) in the LaTeX source.

Parameters

- **matrix** – The matrix to serialize.
- **version** (*int*) – The (Micro) QR code version
- **out** – Filename or a file-like object supporting to write text data.
- **scale** – Indicates the size of a single module (default: 1 which corresponds to 1 x 1 in the provided unit per module).
- **border** (*int*) – Integer indicating the size of the quiet zone. If set to `None` (default), the recommended border size will be used (`4` for QR Codes, `2` for a Micro QR Codes).
- **color** (*str*) – LaTeX color name. The color name is taken at it is, so ensure that it refers either to a default color name or that the color was defined previously.
- **unit** – Unit of the drawing (default: `pt`)
- **url** – Optional URL where the QR Code should point to. Requires the "hyperref" package. Default: `None`.

`segno.writers.`**write_txt**(*matrix*, *version*, *out*, *border=None*, *color=u'1'*, *background=u'0'*)

Serializes QR code in a text format.

Parameters

- **matrix** – The matrix to serialize.
- **version** (*int*) – The (Micro) QR code version
- **out** – Filename or a file-like object supporting to write text.
- **border** (*int*) – Integer indicating the size of the quiet zone. If set to `None` (default), the recommended border size will be used (`4` for QR Codes, `2` for a Micro QR Codes).
- **color** – Character to use for the black modules (default: '1')
- **background** – Character to use for the white modules (default: '0')

`segno.writers.`**write_xbm**(*matrix*, *version*, *out*, *scale=1*, *border=None*, *name=u'img'*)

Serializes the matrix as XBM image.

Parameters

- **matrix** – The matrix to serialize.
- **version** (*int*) – The (Micro) QR code version
- **out** – Filename or a file-like object supporting to write text data.
- **scale** – Indicates the size of a single module (default: 1 which corresponds to 1 x 1 in the provided unit per module).

- **border** (*int*) – Integer indicating the size of the quiet zone. If set to `None` (default), the recommended border size will be used (`4` for QR Codes, `2` for a Micro QR Codes).
- **name** – Prefix for the variable names. Default: "img". The prefix is used to construct the variable names: `` `#define <prefix>_width` `static unsigned char <prefix>_bits[]` ``

segno.writers.**write_xpm**(*matrix*, *version*, *out*, *scale=1*, *border=None*, *color=u'#000'*, *background=u'#fff'*, *name=u'img'*)

Serializes the matrix as XPM image.

Parameters

- **matrix** – The matrix to serialize.
- **version** (*int*) – The (Micro) QR code version
- **out** – Filename or a file-like object supporting to write binary data.
- **scale** – Indicates the size of a single module (default: 1 which corresponds to 1 x 1 pixel per module).
- **border** (*int*) – Integer indicating the size of the quiet zone. If set to `None` (default), the recommended border size will be used (`4` for QR Codes, `2` for a Micro QR Codes).
- **color** – Color of the modules (default: black). The color can be provided as `(R, G, B)` tuple, as web color name (like "red") or in hexadecimal format (`#RGB` or `#RRGGBB`).
- **background** – Optional background color (default: white). See *color* for valid values. `None` indicates a transparent background.
- **name** (*str*) – Name of the image (must be a valid C-identifier). Default: "img".

CHAPTER 12

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

V

W