Segno Documentation

Author

Aug 31, 2020

CONTENTS

Pure Python QR Code generator with no dependencies.

Segno (Italian for “sign” / “symbol”) generates QR Codes and Micro QR Codes according to ISO/IEC 18004:2015(E).

Contents:

CHAPTER

ONE

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 white (or transparent) and have a quiet zone (border) of four (or two for Micro QR Codes) light modules. Nearly all output formats provide options to change at least the scale of the code, the color of the dark modules, and the border, see *segno.QRCode.save()* and *QR code and Micro QR code serialization* for details.

```
>>> 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', dark='green')  # default border, dark␣
↪modules are green
>>> qr.save('you-know-my-name-background-grey.svg', light='#eee')  # default border,␣
↪background grey
```

The factory function *segno.make()* chooses the minimal possible (Micro) QR Code version with a maximal error correction 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'
```

```
>>> import segno
>>> qr = segno.make_qr('The Beatles')  # Same content but as QR Code
>>> qr.designator
'1-Q'
```

```
>>> import segno
>>> qr = segno.make('The Beatles', micro=False)  # Disallow Micro QR Codes
>>> 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
```

1.1 QR Code 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.designator
'M4-M'
```

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

1.2 QR Code error correction level

By default, Segno uses at minimum the error correction level "L" to encode a (Micro) QR code.

Segno tries by default to enhance the provided error correction level if *boost_error* is not set to `False`; it takes the `error` level as minimum error level without changing the (Micro) QR Code version, c.f. *Boost error correction level* for details.

If this behaviour is not desired, *boost_error* must be set to `False` (default: `True`).

Use the parameter *error* to change the (minimum) error correction level.

The *error* parameter is case-insensitive. Available error correction levels are `L` (lowest error correction level: 7% of codewords can be restored), `M` (error correction level "medium": 15% of codewords can be restored), `Q` (error correction level "quartile": 25% of codewords can be restored), and `H` (highest error correction level: 30% of codewords can be restored). 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 *segno.make()*, never a Micro QR code.

```
>>> import segno
>>> qr = segno.make('Parisienne Walkways',
                    error='l')  # Explicit (minimum) error correction level
>>> qr.designator # The error correction level was changed to "Q" since there was␣
↪enough available space
'2-Q'
```

```
>>> import segno
>>> qr = segno.make('Parisienne Walkways', error='l', boost_error=False)  # Explicit␣
↪error level
>>> qr.designator
'2-L'
```

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

1.3 QR Code data masking

Segno chooses by default an optimal mask for the provided input, but the user may specify the preferred mask as well. QR Codes support 8 mask patterns, while Micro QR Codes support 4 mask patterns, only.

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

1.3.1 Micro QR code with different data masks

M3-M codes encoding “Ai Du” with the four different masks:

Fig. 1: Micro QR Code using data mask pattern 00 (mask=0)

Fig. 2: Micro QR Code using data mask pattern 01 (mask=1)

Fig. 3: Micro QR Code using data mask pattern 10 (mask=2)

Fig. 4: Micro QR Code using data mask pattern 11 (mask=3)

1.3.2 QR code with different data masks

1-H codes encoding “Ai Du” using the eight different mask patterns:

Fig. 5: QR Code using data mask pattern 000 (mask=0)

Fig. 6: QR Code using data mask pattern 001 (mask=1)

Fig. 7: QR Code using data mask pattern 010 (mask=2)

Fig. 8: QR Code using data mask pattern 011 (mask=3)

Fig. 9: QR Code using data mask pattern 100 (mask=4)

Fig. 10: QR Code using data mask pattern 101 (mask=5)

Fig. 11: QR Code using data mask pattern 110 (mask=6)

Fig. 12: QR Code using data mask pattern 111 (mask=7)

CHAPTER

TWO

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 a (Micro) QR code in different formats:

```
>>> import segno
>>> qr = segno.make('Tomorrow Never Knows')
>>> qr.save('tomorrow-never-knows.svg')
>>> qr.save('tomorrow-never-knows.png')
>>> qr.save('tomorrow-never-knows.eps')
```

2.1 Border

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, the user may specify the border explicitly.

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

```
>>> import segno
>>> qr = segno.make('Vampire Blues')
>>> qr.save('vampire-blues.png', border=10)  # Larger border
```

2.2 Scale

Most serializers accept a `scale` parameter which indicates the scaling factor of the serialization. By default, the scaling factor is `1` which means that size of a dark / light module 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
```

```
>>> import segno
>>> qr = segno.make_qr('The Beatles')
>>> qr.save('the-beatles.png', scale=10)  # 1 module == 10 pixels
```

```
>>> import segno
>>> qr = segno.make_qr('The Beatles')
>>> qr.save('the-beatles.svg', scale=2.4)   # SVG accepts float values
```

```
>>> import segno
>>> qr = segno.make_qr('The Beatles')
>>> # The SVG serializer provides the "unit" parameter to specify
>>> # how to interpret the values
>>> qr.save('the-beatles.svg', scale=10, unit='mm')  # 1 unit = 1 mm
```

```
>>> import segno
>>> qr = segno.make_qr('The Beatles')
>>> qr.save('the-beatles.svg', unit='cm')  # 1 unit = 1 cm, result as above
```

2.3 Color of dark and light modules

Many serializers accept the parameters `dark` and `light` to specify the color of the dark modules and light modules. See *Colorful QR Codes* for details.

```
>>> import segno
>>> qr = segno.make("Sgt. Pepper's Lonely Hearts Club Band")
>>> qr.save('sgt-peppers.svg', dark='darkred', light='lightblue')
```

```
>>> import segno
>>> qr = segno.make("Sgt. Pepper's Lonely Hearts Club Band")
>>> qr.save('sgt-peppers.svg', dark='#ccc')
```

```
>>> import segno
>>> qr = segno.make("Sgt. Pepper's Lonely Hearts Club Band")
>>> qr.save('sgt-peppers.png', light=None)  # Transparent background
```

```
>>> import segno
>>> qr = segno.make("Sgt. Pepper's Lonely Hearts Club Band")
>>> # Dark modules = transparent, light modules = black
>>> qr.save('sgt-peppers.png', dark=None, light='black')
```

```
>>> import segno
>>> qr = segno.make("Sgt. Pepper's Lonely Hearts Club Band")
```

(continues on next page)

(continued from previous page)

```
>>> # Dark modules with alpha transparency
>>> qr.save('sgt-peppers.png', dark='#0000ffcc')
```

```
>>> import segno
>>> qr = segno.make("Sgt. Pepper's Lonely Hearts Club Band")
>>> qr.save('sgt-peppers.svg', dark='#00fc')  # Same as above but SVG
```

```
>>> import segno
>>> qr = segno.make("Sgt. Pepper's Lonely Hearts Club Band")
>>> # Anonther color, save as compressed SVG
>>> qr.save('sgt-peppers.svgz', dark=(8, 90, 117))
```

2.4 Saving QR codes to streams

If the QR code should be serialized to a buffer, use the *kind* parameter to specify the output format. Please note that some serializers write bytes while others write strings, see *segno.QRCode.save()* for details.

```
>>> import segno
>>> import io
>>> qr = segno.make('Paul McCartney')
>>> buff = io.BytesIO()
>>> qr.save(buff, kind='svg')
>>> # All other serializer parameters are supported as well
>>> buff = io.BytesIO()
>>> qr.save(buff, kind='svg', dark='darkblue', light='#eee')
```

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

2.5 More colorful QR Codes

The *SVG*, *PNG* and *PPM* serializers support more than two colors.

```
>>> import segno
>>> qr = segno.make('Yellow Submarine', error='h')
>>> qr.save('yellow-submarine.png', scale=5, dark='darkred', data_dark='darkorange',
            data_light='yellow')
```

See *Colorful QR Codes* for available options.

2.6 Available serializers

ANSI ANSI escape code. The serializer supports the border keyword, only. See *ANSI* for details.

EPS Encapsulated PostScript (EPS). The serializer provides all default features (scale, border, color of dark / light modules), see *EPS* for details.

LaTeX LaTeX / PGF/TikZ. The serializer provides no support to change the color color of the light modules, but all other default features (scale, border, color) are supported, see *LaTeX* for details.

PAM Portable Arbitrary Map (PAM). The serializer provides all default features (scale, border, color of dark / light modules), see *PAM* for details.

PBM Portable Bitmap (PBM). The serializer does not support any coloring, but the common featurs like scale and border are supported, see *PBM* for details.

PPM Portable Pixmap (PPM). The serializer does not support transparency, but the common featurs like scale and border are supported, see *PPM* for details.

PDF Portable Document Format (PDF). The serializer provides all default features (scale, border, color of dark / light modules), see *PDF* for details.

PNG Portable Network Graphics (PNG). The serializer provides all default features (scale, border, color of dark / light modules) and a few more to customize the output, see *PNG* for details.

SVG Scalable Vector Graphics (SVG). The serializer provides all default features (scale, border, color of dark / light modules) and many more to customize the output, see *SVG* for details. SVGZ (compressed SVG) is supported as well.

TXT Text output. The serializer does not support any scale or color, but the character for the dark and white modules may be specified, see *TXT* for details.

XBM X BitMap (XBM). The serializer does not support any coloring, but scale and border are supported, see *XBM* for details.

XPM X PixMap (XPM). The serializer provides all default features (scale, border, color of dark / light modules) and a few more, see *XPM* for details.

CHAPTER

THREE

BOOST ERROR CORRECTION LEVEL

If the user does not provide any *--version* or *version* keyword parameter for *segno.make()*, Segno uses the minimal possible (Micro) QR Code symbol with a maximal error correction level. The QR Code version dominates, meaning that Segno will never choose a better error correction level if this choice requires a higher (Micro) QR Code version.

If the user provides the *--error* (or *error* keyword for *segno.make()*), the error correction level is treated as minimal error correction level.

To prevent any error correction level enhancement, Segno provides the *--no-error-boost* option and *boost_error=False* option for *segno.make()*.

It's recommended to keep the error level correction boosting, because a better error correction level improves the probability that the QR Code can be read by average QR Code decoders under all circumstances.

3.1 Examples

Keeping the default (boost error level on):

```
>>> import segno
>>> qr = segno.make('The Long and Winding Road')
>>> qr.designator
'2-M'
```

Segno returns a 2-M QR code (version 2, error correction level "M").

If the user does not allow any enhancement of the error correction level, Segno returns a 2-L QR code (version 2, error correction level "L") which does not optimally exploit the possible error corrections:

```
>>> import segno
>>> qr = segno.make('The Long and Winding Road', boost_error=False)
>>> qr.designator
'2-L'
```

As shown, both QR codes use the same version (and are therefore have the same size). However, the first QR code uses a better error correction level (15% vs. 7%) and should be easier to read.

CHAPTER

FOUR

QR CODE MODES

The ISO/IEC 18004 standard defines four modes in order to encode the data as efficiently as possible. If no encoding or mode is provided, Segno tries to find the most efficient encoding / mode.

The mode may be specified by the *mode* (CLI: *--mode* or *-m*) parameter although it is recommended to let Segno decide which mode / encoding should be used.

4.1 Numeric mode

The numeric mode is the most efficient way to encode digits. This mode does not cover negative numbers because it does not support the minus sign (or plus sign).

The numeric mode is supported by QR Codes and Micro QR Codes.

Segno detects the numeric mode if the data is provided as string or integer:

```
>>> import segno
>>> qr = segno.make('64')
>>> qr2 = segno.make(64)
>>> qr.designator
'M1'
>>> qr2.designator
'M1'
>>> qr.mode
'numeric'
>>> qr2.mode
'numeric'
>>> qr == qr2
True
```

Segno chooses by default a minimal QR Code output. Since the numeric mode is supported by Micro QR Codes and QR Codes, Segno chooses a Micro QR Code as most efficient representation.

To enforce a QR Code, use either the factory function *segno.make_qr()* or set the *micro* parameter of *segno.make()* to `False`:

```
>>> import segno
>>> qr = segno.make_qr('64')
>>> qr2 = segno.make(64, micro=False)
```

(continues on next page)

(continued from previous page)

```
>>> qr.designator
'1-H'
>>> qr.mode
'numeric'
>>> qr == qr2
True
```

4.2 Alphanumeric mode

The alphanumeric mode extends the *Numeric mode* by various characters. Namely about the upper case letters `ABCDEFGHIJKLMNOPQRSTUVWXYZ`, a space character " " and other letters `$%*+-./:`.

```
>>> import segno
>>> qr = segno.make('REVOLUTION NO. 9')
>>> qr.designator
'M4-M'
>>> qr.mode
'alphanumeric'
```

As stated in *Numeric mode*, Segno tries to find the smallest possible code. To ensure a QR Code (and not a Micro QR Code), use the above mentioned factory functions:

```
>>> import segno
>>> qr = segno.make_qr('REVOLUTION NO. 9')
>>> qr2 = segno.make('REVOLUTION NO. 9', micro=False)
>>> qr.designator
'1-Q'
>>> qr.mode
'alphanumeric'
```

Lower case characters are not covered by the alphanumeric mode, but by the *Byte mode*

```
>>> import segno
>>> qr = segno.make('Revolution No. 9')
>>> qr.mode
'byte'
```

4.3 Kanji mode

Kanji can be encoded compactly and efficiently and requires significantly less space than encoding the characters in UTF-8.

```
>>> import segno
>>> qr = segno.make('')
>>> qr.designator
'M3-L'
>>> qr.mode
'kanji'
```

```
>>> import segno
>>> qr = segno.make_qr('')
>>> qr.designator
'1-Q'
>>> qr.mode
'kanji'
```

4.4 Byte mode

The byte mode covers all data which cannot be represented by the other modes. Segno tries, according to ISO/IEC 18004, to encode the data with ISO 8859-1. In case the data cannot be represented by ISO 8859-1, UTF-8 is used as fallback.

```
>>> import segno
>>> qr = segno.make('Turn off your mind relax and float down stream')
>>> qr.designator
'3-L'
>>> qr.mode
'byte'
```

The byte mode is also supported by Micro QR Code M3 and M4:

```
>>> import segno
>>> qr = segno.make('Let it be')
>>> qr.designator
'M3-L'
>>> qr.mode
'byte'
```

4.5 Hanzi mode

The Hanzi mode is not covered by ISO/IEC 18004 and should be used with care since it is not widely supported by QR Code readers, although the ZXing project supports decoding QR Codes which utilize the Hanzi mode.

Note: Since this mode is not covered by the ISO standard, Segno tries not to detect Hanzi. The user has to enable it explicitly. Further, the Hanzi mode is not available for Micro QR Codes.

```
>>> import segno
>>> qr = segno.make('')
>>> qr.designator
'2-M'
>>> qr.mode
'byte'
```

The QR Code uses the *Byte mode* because no other mode fits.

To enable Hanzi, provide the mode in the factory function:

```
>>> import segno
>>> qr = segno.make('', mode='hanzi')
>>> qr.mode
'hanzi'
>>> qr.designator
'1-M'
```

As shown, the input is encoded much more compact and a 1-M instead of a 2-M QR Code is generated.

CHAPTER

FIVE

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).

Example: The 2-L QR code encodes the same information ("I read the news today oh boy") as the following 1-L QR codes which are using Structured Append:

With Structured Append (version 1):

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*.

5.1 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)
```

5.2 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']
```

Example: The 6-L QR code encodes the same information (first verse of the song “Yesterday”) as the four 2-L QR codes.

The following 2-L QR codes were created by specifying that 4 codes should be generated (*symbol_count=4*). The result would be the same if the user specifies that a sequence of QR codes with version 2 should be created.

CHAPTER

SIX

COLORFUL QR CODES

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

The color values can be provided 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 value `None` is accepted by many serialization formats and indicates transparency, i.e. `light=None` indicates that all light modules should be transparent.

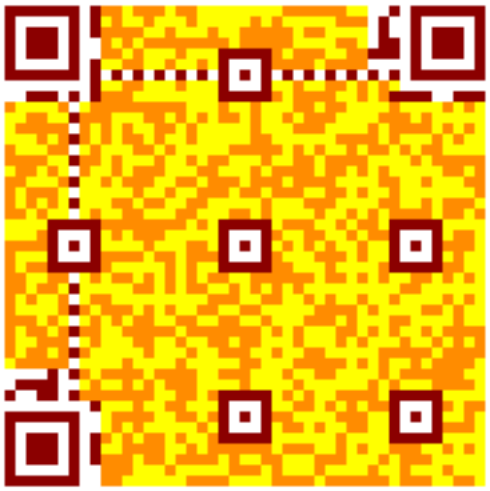
In almost all cases the color values are automatically converted into a meaningful value of the specific output format.

The keywords are optional, if not provided, all serializers use reasonable default values (dark modules are rendered in black, light modules in white).

Note: Providing an alpha channel to a serializer which does not accept an alpha channel results usually into an error.

The *SVG*, *PNG* and *PPM* serializers support more than two colors, every module type may have its own color.

```
>>> import segno
>>> # Force version 7 since smaller versions don't carry any version information (see␣
↪below)
>>> qr = segno.make('Yellow Submarine', version=7, error='h')
>>> qr.save('yellow-submarine.png', scale=4, dark='darkred', data_dark='darkorange',
            data_light='yellow')
```



```
>>> import segno
>>> mqr = segno.make('Rain', error='q')
>>> mqr.save('rain.png', scale=4, dark='darkblue', data_dark='steelblue')
```

6.1 Module names

The following examples show the results of all supported module types. The unaffected modules are rendered as grey or white modules, the red modules show the result of the keyword.

The keywords “dark” and “light” (at the command line *--dark* and *--light*) are supported by almost all serialization formats.

6.1.1 dark / --dark

Sets the (default) color of dark modules.

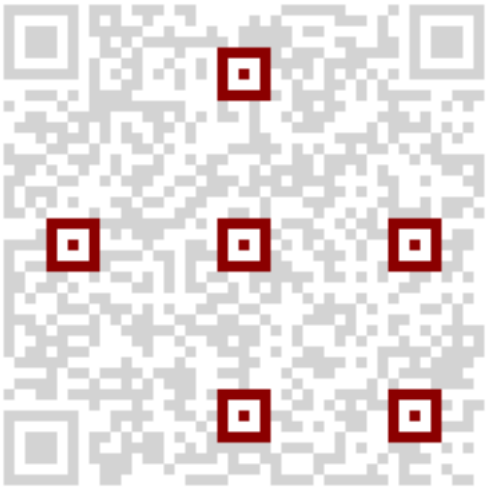
6.1.2 light / `--light`

Sets the (default) color of light modules.

6.1.3 alignment_dark / `--align-dark`

Sets the color of the dark alignment pattern modules.

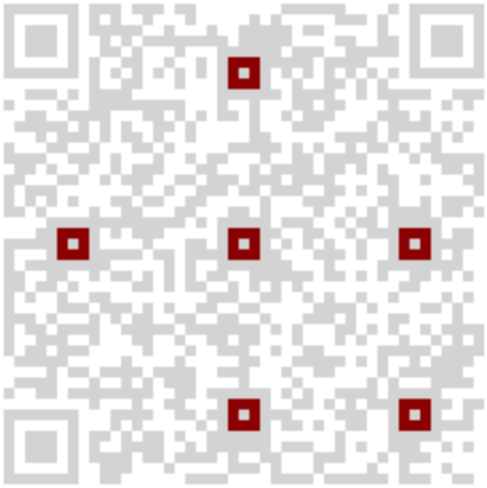
Micro QR Codes don't have alignment patterns.



6.1.4 alignment_light / `--align-light`

Sets the color of the light alignment pattern modules.

Micro QR Codes don't have alignment patterns.



6.1.5 dark_module / `--dark-module`

Sets the color of the dark module.

Micro QR Codes don't have a dark module.

6.1.6 data_dark / `--data-dark`

Sets the color of the dark data modules.

6.1.7 data_light / `--data-light`

Sets the color of the light data modules.

6.1.8 finder_dark / `--finder-dark`

Sets the color of the dark modules of the finder pattern.

6.1.9 finder_light / `--finder-light`

Sets the color of the light modules of the finder pattern.

6.1.10 format_dark / `--format-dark`

Sets the color of the dark modules of the format information.

6.1.11 format_light / `--format-light`

Sets the color of the light modules of the format information.

6.1.12 quiet_zone / `--quiet-zone`

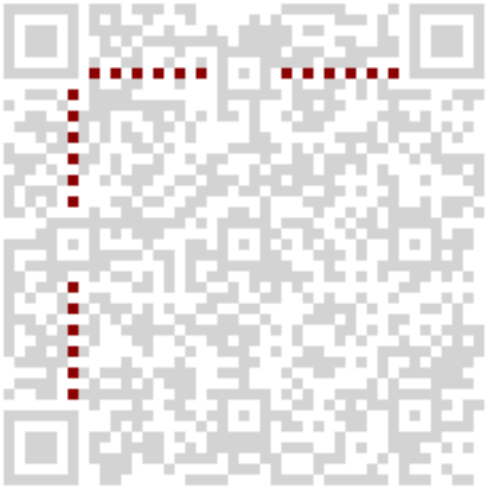
Sets the color of the quiet zone.

6.1.13 separator / `--separator`

Sets the color of the separator.

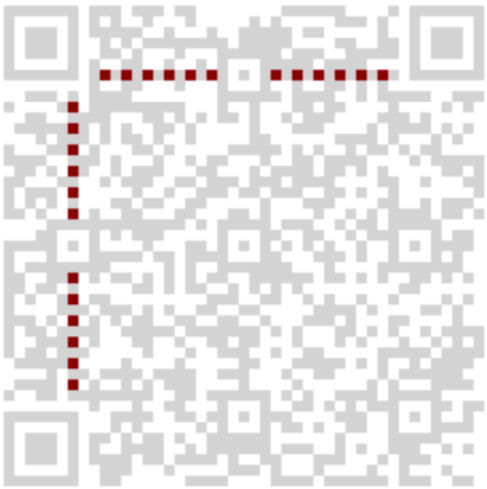
6.1.14 timing_dark / `--timing-dark`

Sets the color of the dark modules of the timing pattern.



6.1.15 timing_light / `--timing-light`

Sets the color of the light modules of the timing pattern.



6.1.16 version_dark / `--version-dark`

Sets the color of the dark modules of the version information.

Micro QR Codes and QR Codes lesser than version 7 don't carry any version information.

6.1.17 version_light / `--version-light`

Sets the color of the light modules of the version information.

Micro QR Codes and QR Codes lesser than version 7 don't carry any version information.

CHAPTER

SEVEN

SVG OPTIONS

All options of the *Colorful QR Codes* are supported. Additionally, the following options are provided by the SVG serializer.

7.1 Options

7.1.1 xmldecl

Boolean (default: `True`) to enable (default) or omit (CLI: `--no-xmldecl`) the XML declaration

7.1.2 svgns

Boolean to enable (default) or omit (CLI: `--no-namespace`) the SVG namespace declaration.

7.1.3 svgid / `--svgid`

String (default: `None`). CSS identifier of the `svg` element.

7.1.4 svgclass / `--svgclass`

String (default: "segno"). CSS class of the `svg` element. Use `None` or an empty string to omit the attribute.

7.1.5 lineclass / `--lineclass`

String (default: "qrline"). CSS class of all paths. Use `None` or an empty string to omit the attribute.

7.1.6 omitsize / `--no-size`

Boolean to disable the `width` and `height` attributes (CLI: `--no-size`).

If set to `True` (default: `False`) the attributes will be replaced by a `viewBox` attribute.

7.1.7 nl

Boolean (default: `True`) to enable / disable a trailing new line character (`\n`) at the end of the document. It's enabled by default. Set to `False` (CLI: *`--no-newline`*) to omit it.

7.1.8 title / `--title`

String (default: `None`).

Sets the title of the graphic. If empty or `None`, the title is omitted.

7.1.9 desc / `--desc`

String (default: `None`).

Sets the description of the graphic. If empty or `None`, the description is omitted.

7.1.10 unit / `--unit`

String (default: `None`).

Sets the unit of the `width` / `height` attributes. The unit is not checked, any non empty value is used as appendix to the numeric width / height attributes. Common values are "mm" or "cm".

7.1.11 encoding / `--encoding`

String (default: "utf–8").

Sets the encoding of the XML document. If set to `None`, the encoding parameter of the XML declaration is omitted but the XML default encoding "UTF-8" is used.

> **Note:** It is not recommended to set the encoding to another value than "utf-8" if `xmldecl` is set to `False`.

7.1.12 draw_transparent

Boolean (default: `False`).

Set to `True` (CLI: *`--draw-transparent`* to draw transparent paths.

7.1.13 svgversion / `--svgversion`

`int` or `float` (default: `None`)

Sets the SVG `version` attribute. By default, the attribute is omitted. Any given value is converted into a string. The value may have an impact on the generated paths since this lib assumes that a value lesser than `2` does not support CSS Color Module Level 3.

> **Note:** It's recommended to keep the default value.

7.2 Optimizing SVG

The SVG serializer supports several options to optimize the output. By default, a minimal, standalone SVG graphic is generated which includes the XML declaration, the SVG namespace and a trailing newline.

```
>>> import segno
>>> qr = segno.make('Penny Lane', error='h')
>>> qr.save('penny-lane.svg', scale=4)
```

XML markup:

```
<?xml version="1.0" encoding="utf-8"?>
<svg xmlns="http://www.w3.org/2000/svg" width="132" height="132" class="segno"><path␣
↪transform="scale(4)" stroke="#000" class="qrline" d="M4 4.5h7m3 0h1m1 0h1m1 0h1m3␣
↪0h7m-25 1h1m5 0h1m3 0h2m1 0h4m1 0h1m5 0h1m-25 1h1m1 0h3m1 0h1m2 0h2m5 0h1m1 0h1m1␣
↪0h3m1 0h1m-25 1h1m1 0h3m1 0h1m2 0h2m1 0h1m1 0h1m3 0h1m1 0h3m1 0h1m-25 1h1m1 0h3m1␣
↪0h1m1 0h2m4 0h3m1 0h1m1 0h3m1 0h1m-25 1h1m5 0h1m2 0h1m2 0h2m2 0h1m1 0h1m5 0h1m-25␣
↪1h7m1 0h1m1 0h1m1 0h1m1 0h1m1 0h1m1 0h7m-17 1h1m2 0h1m1 0h4m-15 1h2m2 0h5m3 0h1m1␣
↪0h3m1 0h1m-19 1h2m1 0h1m1 0h10m2 0h1m2 0h1m-23 1h2m1 0h5m1 0h1m1 0h1m1 0h1m2 0h1m5␣
↪0h1m-23 1h1m1 0h3m2 0h1m2 0h3m1 0h2m1 0h1m2 0h4m-24 1h1m1 0h1m1 0h1m1 0h2m3 0h2m1␣
↪0h1m3 0h1m1 0h5m-21 1h1m2 0h4m1 0h3m1 0h2m1 0h6m-24 1h1m1 0h1m1 0h5m1 0h3m1 0h2m6␣
↪0h1m-24 1h1m2 0h2m2 0h1m3 0h1m4 0h1m1 0h1m5 0h1m-22 1h1m2 0h2m2 0h1m2 0h11m-16␣
↪1h1m2 0h1m1 0h1m1 0h2m3 0h1m2 0h1m-24 1h7m1 0h5m2 0h2m1 0h1m1 0h1m2 0h2m-25 1h1m5␣
↪0h1m4 0h1m1 0h2m1 0h1m3 0h1m2 0h2m-25 1h1m1 0h3m1 0h1m6 0h1m2 0h5m3 0h1m-25 1h1m1␣
↪0h3m1 0h1m1 0h2m3 0h1m4 0h1m1 0h2m2 0h1m-25 1h1m1 0h3m1 0h1m1 0h7m2 0h1m1 0h2m2␣
↪0h1m-24 1h1m5 0h1m2 0h2m2 0h1m2 0h4m1 0h2m-23 1h7m5 0h1m1 0h1m1 0h1m2 0h3m1 0h2"/></
↪svg>
```

To suppress the XML declaration, use `xmldecl=False`.

```
>>> import segno
>>> qr = segno.make('Penny Lane', error='h')
>>> qr.save('penny-lane.svg', scale=4, xmldecl=False)
```

XML markup:

```
<svg xmlns="http://www.w3.org/2000/svg" width="132" height="132" class="segno"><path␣
↪transform="scale(4)" stroke="#000" class="qrline" d="M4 4.5h7m3 0h1m1 0h1m1 0h1m3␣
↪0h7m-25 1h1m5 0h1m3 0h2m1 0h4m1 0h1m5 0h1m-25 1h1m1 0h3m1 0h1m2 0h2m5 0h1m1 0h1m1␣
↪0h3m1 0h1m-25 1h1m1 0h3m1 0h1m2 0h2m1 0h1m1 0h1m3 0h1m1 0h3m1 0h1m-25 1h1m1 0h3m1␣
↪0h1m1 0h2m4 0h3m1 0h1m1 0h3m1 0h1m-25 1h1m5 0h1m2 0h1m2 0h2m2 0h1m1 0h1m5 0h1m-25␣
↪1h7m1 0h1m1 0h1m1 0h1m1 0h1m1 0h1m1 0h7m-17 1h1m2 0h1m1 0h4m-15 1h2m2 0h5m3 0h1m1␣
↪0h3m1 0h1m-19 1h2m1 0h1m1 0h10m2 0h1m2 0h1m-23 1h2m1 0h5m1 0h1m1 0h1m1 0h1m2 0h1m5␣
↪0h1m-23 1h1m1 0h3m2 0h1m2 0h3m1 0h2m1 0h1m2 0h4m-24 1h1m1 0h1m1 0h1m1 0h2m3 0h2m1␣
↪0h1m3 0h1m1 0h5m-21 1h1m2 0h4m1 0h3m1 0h2m1 0h6m-24 1h1m1 0h1m1 0h5m1 0h3m1 0h2m6␣
↪0h1m-24 1h1m2 0h2m2 0h1m3 0h1m4 0h1m1 0h1m5 0h1m-22 1h1m2 0h2m2 0h1m2 0h11m-16␣
↪1h1m2 0h1m1 0h1m1 0h2m3 0h1m2 0h1m-24 1h7m1 0h5m2 0h2m1 0h1m1 0h1m2 0h2m-25 1h1m5␣
↪0h1m4 0h1m1 0h2m1 0h1m3 0h1m2 0h2m-25 1h1m1 0h3m1 0h1m6 0h1m2 0h5m3 0h1m-25 1h1m1␣
↪0h3m1 0h1m1 0h2m3 0h1m4 0h1m1 0h2m2 0h1m-25 1h1m1 0h3m1 0h1m1 0h7m2 0h1m1 0h2m2␣
↪0h1m-24 1h1m5 0h1m2 0h2m2 0h1m2 0h4m1 0h2m-23 1h7m5 0h1m1 0h1m1 0h1m2 0h3m1 0h2"/></
↪svg>
```

If the SVG graphic should be embedded into a HTML5 context, the namespace declaration is superfluous, omit it via `svgns=False`.

```
>>> import segno
>>> qr = segno.make('Penny Lane', error='h')
>>> qr.save('penny-lane.svg', scale=4, xmldecl=False, svgns=False)
```

XML markup:

```
<svg width="132" height="132" class="segno"><path transform="scale(4)" stroke="#000"␣
↪class="qrline" d="M4 4.5h7m3 0h1m1 0h1m1 0h1m3 0h7m-25 1h1m5 0h1m3 0h2m1 0h4m1␣
↪0h1m5 0h1m-25 1h1m1 0h3m1 0h1m2 0h2m5 0h1m1 0h1m1 0h3m1 0h1m-25 1h1m1 0h3m1 0h1m2␣
↪0h2m1 0h1m1 0h1m3 0h1m1 0h3m1 0h1m-25 1h1m1 0h3m1 0h1m1 0h2m4 0h3m1 0h1m1 0h3m1␣
↪0h1m-25 1h1m5 0h1m2 0h1m2 0h2m2 0h1m1 0h1m5 0h1m-25 1h7m1 0h1m1 0h1m1 0h1m1 0h1m1␣
↪0h1m1 0h7m-17 1h1m2 0h1m1 0h4m-15 1h2m2 0h5m3 0h1m1 0h3m1 0h1m-19 1h2m1 0h1m1␣
↪0h10m2 0h1m2 0h1m-23 1h2m1 0h5m1 0h1m1 0h1m1 0h1m2 0h1m5 0h1m-23 1h1m1 0h3m2 0h1m2␣
↪0h3m1 0h2m1 0h1m2 0h4m-24 1h1m1 0h1m1 0h1m1 0h2m3 0h2m1 0h1m3 0h1m1 0h5m-21 1h1m2␣
↪0h4m1 0h3m1 0h2m1 0h6m-24 1h1m1 0h1m1 0h5m1 0h3m1 0h2m6 0h1m-24 1h1m2 0h2m2 0h1m3␣
↪0h1m4 0h1m1 0h1m5 0h1m-22 1h1m2 0h2m2 0h1m2 0h11m-16 1h1m2 0h1m1 0h1m1 0h2m3 0h1m2␣
↪0h1m-24 1h7m1 0h5m2 0h2m1 0h1m1 0h1m2 0h2m-25 1h1m5 0h1m4 0h1m1 0h2m1 0h1m3 0h1m2␣
↪0h2m-25 1h1m1 0h3m1 0h1m6 0h1m2 0h5m3 0h1m-25 1h1m1 0h3m1 0h1m1 0h2m3 0h1m4 0h1m1␣
↪0h2m2 0h1m-25 1h1m1 0h3m1 0h1m1 0h7m2 0h1m1 0h2m2 0h1m-24 1h1m5 0h1m2 0h2m2 0h1m2␣
↪0h4m1 0h2m-23 1h7m5 0h1m1 0h1m1 0h1m2 0h3m1 0h2"/></svg>
```

By default, Segno adds a `class` attribute to the `svg` element, further, it adds a `class` attribute to all paths. To omit the `class` attribute of the `svg` element, use `svgclass=None`. To omit the `class` attribute of the paths, use `lineclass=None`.

```
>>> import segno
>>> qr = segno.make('Penny Lane', error='h')
>>> qr.save('penny-lane.svg', scale=4, xmldecl=False, svgns=False, svgclass=None,␣
↪lineclass=None)
```

```
<svg width="132" height="132"><path transform="scale(4)" stroke="#000" d="M4 4.5h7m3␣
↪0h1m1 0h1m1 0h1m3 0h7m-25 1h1m5 0h1m3 0h2m1 0h4m1 0h1m5 0h1m-25 1h1m1 0h3m1 0h1m2␣
↪0h2m5 0h1m1 0h1m1 0h3m1 0h1m-25 1h1m1 0h3m1 0h1m2 0h2m1 0h1m1 0h1m3 0h1m1 0h3m1␣
↪0h1m-25 1h1m1 0h3m1 0h1m1 0h2m4 0h3m1 0h1m1 0h3m1 0h1m-25 1h1m5 0h1m2 0h1m2 0h2m2␣
↪0h1m1 0h1m5 0h1m-25 1h7m1 0h1m1 0h1m1 0h1m1 0h1m1 0h1m1 0h7m-17 1h1m2 0h1m1 0h4m-15␣
↪1h2m2 0h5m3 0h1m1 0h3m1 0h1m-19 1h2m1 0h1m1 0h10m2 0h1m2 0h1m-23 1h2m1 0h5m1 0h1m1␣
↪0h1m1 0h1m2 0h1m5 0h1m-23 1h1m1 0h3m2 0h1m2 0h3m1 0h2m1 0h1m2 0h4m-24 1h1m1 0h1m1␣
↪0h1m1 0h2m3 0h2m1 0h1m3 0h1m1 0h5m-21 1h1m2 0h4m1 0h3m1 0h2m1 0h6m-24 1h1m1 0h1m1␣
↪0h5m1 0h3m1 0h2m6 0h1m-24 1h1m2 0h2m2 0h1m3 0h1m4 0h1m1 0h1m5 0h1m-22 1h1m2 0h2m2␣
↪0h1m2 0h11m-16 1h1m2 0h1m1 0h1m1 0h2m3 0h1m2 0h1m-24 1h7m1 0h5m2 0h2m1 0h1m1 0h1m2␣
↪0h2m-25 1h1m5 0h1m4 0h1m1 0h2m1 0h1m3 0h1m2 0h2m-25 1h1m1 0h3m1 0h1m6 0h1m2 0h5m3␣
↪0h1m-25 1h1m1 0h3m1 0h1m1 0h2m3 0h1m4 0h1m1 0h2m2 0h1m-25 1h1m1 0h3m1 0h1m1 0h7m2␣
↪0h1m1 0h2m2 0h1m-24 1h1m5 0h1m2 0h2m2 0h1m2 0h4m1 0h2m-23 1h7m5 0h1m1 0h1m1 0h1m2␣
↪0h3m1 0h2"/></svg>
```

To replace the `width` and `height` attributes with `viewBox`, use `omitsize=True`. Since the graphic should take all available space, the `scale` may be omitted, too.

```
>>> import segno
>>> qr = segno.make('Penny Lane', error='h')
>>> qr.save('penny-lane.svg', xmldecl=False, svgns=False, svgclass=None,␣
↪lineclass=None, omitsize=True)
```

XML markup.

```
<svg viewBox="0 0 33 33"><path stroke="#000" d="M4 4.5h7m3 0h1m1 0h1m1 0h1m3 0h7m-25␣
↪1h1m5 0h1m3 0h2m1 0h4m1 0h1m5 0h1m-25 1h1m1 0h3m1 0h1m2 0h2m5 0h1m1 0h1m1 0h3m1␣
↪0h1m-25 1h1m1 0h3m1 0h1m2 0h2m1 0h1m1 0h1m3 0h1m1 0h3m1 0h1m-25 1h1m1 0h3m1 0h1m2␣
↪0h2m4 0h3m1 0h1m1 0h3m1 0h1m-25 1h1m5 0h1m2 0h1m2 0h2m2 0h1m1 0h1m5 0h1m-25 1h7m1␣
↪0h1m1 0h1m1 0h1m1 0h1m1 0h1m1 0h7m-17 1h1m2 0h1m1 0h4m-15 1h2m2 0h5m3 0h1m1 0h3m1␣
↪0h1m-19 1h2m1 0h1m1 0h10m2 0h1m2 0h1m-23 1h2m1 0h5m1 0h1m1 0h1m1 0h1m2 0h1m5 0h1m
↪23 1h1m1 0h3m2 0h1m2 0h3m1 0h2m1 0h1m2 0h4m-24 1h1m1 0h1m1 0h1m1 0h2m3 0h2m1 0h1m3␣
↪0h1m1 0h5m-21 1h1m2 0h4m1 0h3m1 0h2m1 0h6m-24 1h1m1 0h1m1 0h5m1 0h3m1 0h2m6 0h1m-24␣
↪1h1m2 0h2m2 0h1m3 0h1m4 0h1m1 0h1m5 0h1m-22 1h1m2 0h2m2 0h1m2 0h11m-16 1h1m2 0h1m1␣
```

(continues on next page)

(continued from previous page)

Too squeeze the file size futher, omit the trailing newline character via `nl=False`.

```
>>> import segno
>>> qr = segno.make('Penny Lane', error='h')
>>> qr.save('penny-lane.svg', xmldecl=False, svgns=False, svgclass=None,␣
↪lineclass=None, omitsize=True, nl=False)
```

The result is almost he same, but you've saved a few (abt. 130) bytes

CHAPTER

EIGHT

SPECIAL QR CODE FACTORY FUNCTIONS

The *segno.helpers* module provides factory functions to create common QR codes for encoding WIFI configurations, *vCards and MeCards*, *EPC QR Codes* 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.

8.1 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 *segno.helpers.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 "H"
>>> qr = segno.make(config, error='h')
>>> qr.designator
'4-H'
```

8.2 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 error correction level "H"
>>> qr = segno.make(geo_uri, error='H')
>>> qr.designator
'4-H'
```

CHAPTER

NINE

CREATING A QR CODE ENCODING CONTACT INFORMATION

9.1 MeCard

The function *segno.helpers.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', scale=4)
```

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'
```

9.2 vCard

The function *segno.helpers.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
```

(continues on next page)

(continued from previous page)

```
>>> 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', scale=4)
```

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

TEN

EPC QR CODES

European Payments Council Quick Response Codes (EPC QR Codes) are used to initiate SEPA credit transfers. They encode all necessary data (IBAN, amount, recipient) and may be placed on invoices to eliminate the error-prone typing of payment Information.

See the EPC guidelines (PDF document) for more information.

The function *segno.helpers.make_epc_qr()* is used to create a EPC QR code. It is not possible to modify the error correction level or to change the version of the QR code. The EPC guidelines specify that the error correction level is fixed to "M" and the QR Code version must not be higher than 13.

```
>>> from segno import helpers
>>> qr = helpers.make_epc_qr(name='Wikimedia Foerdergesellschaft',
                             iban='DE33100205000001194700',
                             amount=20, text='Spende fuer Wikipedia')
>>> qr.save('spende-wikipedia.png', scale=3)
```

The QR code shown above encodes the following information:

```
BCD
002
2
SCT

Wikimedia Foerdergesellschaft
DE33100205000001194700
EUR20


Spende fuer Wikipedia
```

By default, the function uses the minimal possible encoding to encode the provided data. If another encoding, i.e. UTF-8, is required, the user may specify the encoding with an integer referring to one of the supported encodings (1: UTF-8, 2: ISO 8859-1, 3: ISO 8859-2, 4: ISO 8859-4, 5: ISO 8859-5, 6: ISO 8859-7, 7: ISO 8859-10, 8: ISO 8859-15) or the (case insensitive) name of the encoding like 'utf-8' or 'ISO-8859-1'.

```
>>> from segno import helpers
>>> qr = helpers.make_epc_qr(name='Wikimedia Foerdergesellschaft',
                             iban='DE33100205000001194700', amount=13.05,
                             text='Spende fuer Wikipedia', encoding='utf-8')
>>> qr.save('spende-wikipedia2.png', scale=3)
```

Decoded infomation:

```
BCD
002
1
SCT

Wikimedia Foerdergesellschaft
DE33100205000001194700
EUR13.05


Spende fuer Wikipedia
```

CHAPTER

ELEVEN

ARTISTIC QR CODES

Segno focuses on creating (Micro) QR codes and offers many *output formats* without additional dependencies on other libraries.

Advanced graphic operations require the qrcode-artistic plug-in, which in turn depends on the Pillow library.

The plugin can be used to create animated QR codes or static QR codes with a background image.

To install the plugin, use:

```
pip install qrcode-artistic
```

After that, every QR code created with *segno.make()* has two additional methods "to_pil" and "to_artistic".

The former returns a Pillow Image instance, which can be used for further manipulations (e.g. rotating the QR code).

```
>>> import segno
>>> qr = segno.make('Yellow Submarine', error='h')
>>> img = qr.to_pil(scale=3).rotate(45, expand=True)
>>> img.save('yellow-submarine-rotated.png')
```

The "to_pil" method provides all options of *Colorful QR Codes*.

```
>>> import segno
>>> qr = segno.make('Yellow Submarine', error='h')
>>> img = qr.to_pil(scale=4, dark='darkred', data_dark='darkorange',
                    data_light='yellow')
>>> img.save('yellow-submarine.png')
```

The “to_artistic” method can create animated or static QR codes.

```
>>> import segno
>>> qr = segno.make('The Beatles -- Albums', error='h')
>>> qr.to_artistic(background='src/albums.gif'), target='albums.gif' scale=8)
```

If the Pillow installation supports animated WebP images the plugin can save animated WebP images as well.

```
>>> import segno
>>> qr = segno.make('The Beatles -- Abbey Road', error='h')
>>> qr.to_artistic(background='src/abbey-road-walking.gif', target='abbey-road.webp'␣
↪scale=4)
```

The plugin also supports static backgrounds

```
>>> import segno
>>> qr = segno.make('The Beatles -- Let It Be', error='h')
>>> qr.to_artistic(background='src/letitbe.jpg', target='letitbe.jpg' scale=5)
```

CHAPTER

TWELVE

WEB DEVELOPMENT

A few proposals how to use Segno with popular Python web application frameworks.

12.1 Flask

There are various ways to output QR codes in Jinja templates in conjunction with Flask. See also the Flask example in the repository: https://github.com/heuer/segno/tree/develop/examples/flask_qrcode

12.1.1 Data URIs

Create a QR code in the Flask view and use the *segno.QRCode.svg_data_uri()* or *segno.QRCode.png_data_uri()* methods in the template.

```
from flask import Flask, render_template
import segno

app = Flask(__name__)

@app.route('/')
def home():
    qr = segno.make('The Continuing Story of Bungalow Bill')
    return render_template('example.html', qr=qr)
```

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>QR Codes</title>
  </head>
  <body>
    <img src="{{ qr.png_data_uri(dark='darkblue', data_dark='steelblue', alignment_
↪dark='darkgreen', scale=3) }}"><br>
    <img src="{{ qr.svg_data_uri(dark='darkblue', scale=4) }}">
  </body>
</html>
```

12.1.2 Embed SVG QR codes into HTML

Since HTML5 supports SVG directly, it's also possible to embed the generated SVG directly into a template.

Create the QR code within the Flask view and use the *segno.QRCode.svg_inline()* method in conjunction with the Jinja `|safe` filter.

```
@app.route('/')
def home():
    qr = segno.make('While My Guitar Gently Weeps')
    return render_template('example.html', qr=qr)
```

```
<div>
  {{ qr.svg_inline(scale=3) | safe }}
</div>
```

12.1.3 Create a view

Another possibility is to create the QR codes dynamically in a Flask view and then to deliver them with `send_file`.

```
BEATLES_SONGS = {'Yellow Submarine', 'Let It Be', 'Rocky Raccoon'}

@app.route('/qr-png/')
def qrcode_png():
    data = request.args.get('data')
    # Check if the data is acceptable otherwise a 404 error is generated
    if data not in BEATLES_SONGS:
        return abort(404)
    buff = io.BytesIO()
    segno.make(data, micro=False) \
         .save(buff, kind='png', scale=4, dark='darkblue',
               data_dark='#474747', light='#efefef')
    buff.seek(0)
    return send_file(buff, mimetype='image/png')
```

```
<img src="{{ url_for('qrcode_png', data='Rocky Raccoon') }}">
```

12.2 Django

The project django-segno-qr provides a template tag for creating SVG QR codes in Django templates.

Apart from that, the aforementioned information for Flask should also be adaptable to Django, so here is just a hint on how to save QR codes in a Django `ImageField`.

The complete code is in the repository: https://github.com/heuer/segno/tree/develop/examples/django_qrcode

12.2.1 Saving a QR code to an ImageField

Assuming this simple model.

```
from django.db import models


class Ticket(models.Model):
    name = models.CharField(max_length=150, unique=True)
    qrcode = models.ImageField(upload_to='ticket-qrcodes/')
```

Create a QR code with Segno and save it as PNG into a `io.BytesIO` instance.

```
import io
import segno

out = io.BytesIO()
qr = segno.make('JULIA')
# Save the QR code with transparent background and use dark blue for
# the dark modules
qr.save(out, kind='png', dark='#00008b', light=None, scale=3)
```

Now you can use the content of the buffer as input for a Django `ContentFile`.

```
ticket = Ticket(name='JULIA')
ticket.qrcode.save('JULIA.png', ContentFile(out.getvalue()), save=False)
ticket.save()
```

If for some reason the QR codes should be stored in the lossy file format JPEG, the `qrcode-artistic` plugin is required (see also *Artistic QR Codes*):

```
$ pip install qrcode-artistic
```

```
import io
import segno

out = io.BytesIO()
qr = segno.make('JULIA')
# img is a Pillow Image instance
img = qr.to_pil(dark='#00008b', scale=3)
# Now use Pillow Image.save() to save the QR code
img.save(out, format='jpg')

# ...

ticket.qrcode.save('JULIA.jpg', ContentFile(out.getvalue()), save=False)
```

CHAPTER

THIRTEEN

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 in supported formats like 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.

See the *CLI man page* for a detailed reference of all command line options.

13.1 Usage

Output the QR code to the terminal:

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

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

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

13.2 QR Code version

If the *--version* / *-v* parameter is not provided, Segno chooses the minimal version for the QR code automatically. The version may be specified as an integer or as Micro QR Code identifier.

The content 'Layla' would fit into a 1-H QR code, but the following commands enforce version 5:

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

Micro QR Code:

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

13.3 QR Code error correction level

The default error correction level is "L" (or any better if `--no-error-boost` was not set, c.f. *Boost error correction level*), use the `--error` / `-e` parameter to change it:

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

13.4 QR code serialization

Printing the QR code to the terminal is nice but the `--output` / `-o` 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
```

13.5 Scaling QR codes

If the resulting QR code is too small, *--scale* or *-s* 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.

13.6 Changing the size of the quiet zone

The generated QR code will have a recommended quiet zone / border around the symbol. To change the size of the border, *--border* or the shortcut *-b* can be utilized:

```
$ segno --border 0 --scale 4 --output=black-magic-woman.svg "Black Magic Woman"
```

```
$ segno --border 10 --scale 4 --output diamond.png "Shine On You Crazy Diamond"
```

```
$ segno -b 30 --scale 4 --output=boom-boom.svg Boom Boom
```

13.7 Colors

Usually, all QR codes are serialized in black and white. Use *--dark* to change the color of the dark modules and *--light* to change the color of the light modules.

Change the color of the dark modules to to darkblue:

```
$ segno --dark=darkblue --scale 4 --output=excited.png "So Excited"
```

Change the color of the light modules to transparent:

```
$ segno --light=transparent --scale 4 --output=hotel.png "Hotel California"
```

Change the color of the dark modules to darkblue and the color of the light modules to yellow:

```
$ segno --dark=darkblue --light=yellow --scale 4 --output=dontgiveup.svg "Don't Give␣
↪Up"
```

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

The *SVG*, *PNG* and *PPM* serializers support more than two colors, each module type (finder pattern (dark / light), alignment pattern (dark / light) etc.) may have its own color, see *Colorful QR Codes* for details.

```
$ segno --dark darkred --data-dark darkorange --data-light yellow --scale 5 -o yellow-
↪submarine.png Yellow Submarine
```

13.8 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. Additionally, either the `--version` or the desired number of symbols (`--symbol-count`) must be provided:

```
$ segno --seq -v 1 --scale 3 -o polythene-pam.png "Well you should see Polythene Pam"
```

Specify the max. number of symbols:

```
$ segno --seq --symbol-count=2 -s 3 -o sa_yellow-submarine.png "We all live in a␣
↪yellow submarine"
```

Shortcut to specify the number of symbols:

```
$ segno --seq -sc 2 -s 3 -o julia.png "Half of what I say is meaningless"
```

CHAPTER

FOURTEEN

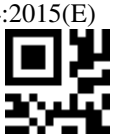
COMPARISON OF PYTHON QR CODE LIBRARIES

14.1 Features

Description	qrcode	PyQRCode	qrcodegen	Segno
Library license	BSD	BSD	MIT	BSD
Library version	6.1	1.2.1	1.6.0	1.3.0
Mode Numeric	Yes	Yes	Yes	Yes
Mode Alphanumeric	Yes	Yes	Yes	Yes
Mode Byte	Yes	Yes	Yes	Yes
Mode Kanji	No	Yes	Yes	Yes
Mode ECI	No	No	Yes	Yes
Mode FNC1	No	No	No	No
Mode Structured Append	No	No	No	Yes
Mode Hanzi[1]	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[2]	No 	No 	No 	Yes 

Continued on next page

Table 1 – continued from previous page

Description	qrcode	PyQRCode	qrcodegen	Segno
Output acc. to ISO/IEC 18004:2015(E) Fig. 2 	•	•	•	Yes 
Find maximal error correction level	No	No	Yes	Yes
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
PAM output	No	No	No	Yes
PBM output	Yes	No	No	Yes
PPM output	Yes	No	No	Yes
LaTeX support	No	No	No	Yes
XBM output	Yes	Yes	No	Yes
XPM output	No	No	No	Yes
PNG data URI	No	No (no valid URI)	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	Yes via plugin
Black and white QR Codes	Yes	Yes	Yes	Yes
Colored QR Codes	Yes	Yes	No	Yes
Animated QR Codes (GIF, APNG, WebP)	No	No	No	Yes via plugin
Changing size of modules (scaling factor)	Yes	Yes	No	Yes
Command line script	Yes	No	No	Yes
Plugins	No	No	No	Yes
Default encoding in Byte mode	ISO/IEC 8859-1 or UTF-8	ISO/IEC 8859-1 or UTF-8	ISO/IEC 8859-1 or UTF-8	ISO/IEC 8859-1 or UTF-8
3rd party dependencies	six, Pillow or Pymaging and Pymaging-PNG (Windows: colorama)	PyPNG	•	•

[1] The Hanzi mode is not part of ISO/IEC 18004 and may not be supported by all QR Code decoders. Segno uses the Hanzi mode if the user

14.2 Performance

Some performance indicators. The script benchmarks.py ran on Intel i7-8559U / CPython 3.7. Each SVG / PNG image uses a scaling factor of 10 (aside from qrcodegen which does not support any scaling).

14.2.1 Create a 1-M QR code

1-M QR code encoding “QR Code Symbol”

14.2.2 Create a 7-Q QR code

7-Q QR code encoding “QR Code Symbol”

14.2.3 Create a 30-H QR code

30-H QR code encoding “QR Code Symbol”

14.2.4 Create a QR code and serialize it as SVG

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

14.2.5 Create a QR code and serialize it as PNG

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

enables it explicitly, see *Hanzi mode* for details

[2] Even if all libs generate the same byte output (`40 e5 15 22 04 36 f6 46 52 05 37 96 d6 26 f6 c0`), the generated QR code may look different because they choose a different mask pattern. ISO/IEC 18004:2015(E) (cf. page 7) uses mask 5, while qrcode and qrcodegen use mask 4 and PyQRCode uses mask 6. All these QR codes can be read by common QR Code readers.

CHAPTER

FIFTEEN

MAN PAGES

Documentation of the command line interface.

15.1 segno

15.1.1 Synopsis

segno [*options*] content

15.1.2 Description

segno creates QR Codes and Micro QR Codes.

It can be used to print the codes to a terminal or to serialize them in several output formats (like SVG and PNG).

15.1.3 Command Line Options

--ver, -V
: Shows Segno's version and exit

-h, --help
: Show a help message which lists all commands and exit

15.1.3.1 QR Code Options

--version VERSION, **-v** VERSION
: QR Code version: 1 .. 40 or Micro Code Version "M1", "M2", "M3", "M4"

--error {L,M,Q,H,-}, **-e** {L,M,Q,H,-}
: Error correction level: "L": 7% (default), "M": 15%, "Q": 25%, "H": 30%, "-": no error correction (used for M1 symbols)

--mode {numeric,alphanumeric,byte,kanji,hanzi}, **-m** {numeric,alphanumeric,byte,kanji,hanzi}
: Mode. If unspecified (default), an optimal mode is chosen for the given input.

--pattern PATTERN, **-p** PATTERN
: Mask pattern to use. If unspecified (default), an optimal mask pattern is used. Valid values for QR Codes: 0 .. 7 Valid values for Micro QR Codes: 0 .. 3

--encoding ENCODING
Specifies the encoding of the provided content. If not specified (default) either ISO 8859-1 is used or if it does not fit, UTF-8 or Shift_JIS (for Kanji content) is used.

--micro
Allow the creation of Micro QR Codes

--no-micro
Disallow creation of Micro QR Codes (default)

--no-error-boost
Disables the automatic error correction level incrementation. By default, the maximal error correction level is used (without changing the version).

--seq
Creates a sequence of QR Codes (Structured Append mode). The *--version* or *--symbol-count* must be provided

--symbol-count SYMBOL_COUNT, **-sc** SYMBOL_COUNT
Number of symbols to create

--border BORDER, **-b** BORDER
Size of the border / quiet zone of the output. By default, the standard border (4 modules for QR Codes, 2 modules for Micro QR Codes) will be used. A value of 0 omits the border

15.1.3.2 Output Options

--scale SCALE, **-s** SCALE
Scaling factor of the output. By default, a scaling factor of 1 is used which can result into too small images. Some output formats, i.e. SVG, accept a decimal value.

--output OUTPUT, **-o** OUTPUT
Output file. If not specified, the QR Code is printed to the terminal

15.1.3.3 Module Colors

Arguments to specify the module colors. Multiple colors are supported for SVG and PNG. The module color support varies between the serialization formats. Most serializers support at least "–dark" and "–light". Unsupported arguments are ignored.

--dark DARK
Sets the (default) color of the dark modules. The color may be specified as web color name, i.e. "red" or as hexadecimal value, i.e. "#0033cc". Some serializers, i.e. SVG and PNG, support alpha channels (8-digit hexadecimal value) and some support "transparent" as color value. The standard color is black

--light LIGHT
Sets the (default) color of the light modules. The standard value is either white or transparent. See *--dark* for a description of allowed values.

--align-dark ALIGN_DARK
Sets the color of the dark modules of the alignment patterns. See *--dark* for a description of allowed values.

--align-light ALIGN_LIGHT
Sets the color of the light modules of the alignment patterns. See *--dark* for a description of allowed values.

--dark-module DARK_MODULE
Sets the color of the dark module. See *--dark* for a description of allowed values.

--data-dark DATA_DARK
Sets the color of the dark data modules. See *--dark* for a description of allowed values.

--data-light DATA_LIGHT
Sets the color of the light data modules. See *--dark* for a description of allowed values.

--finder-dark FINDER_DARK
Sets the color of the dark modules of the finder pattern. See *--dark* for a description of allowed values.

--finder-light FINDER_LIGHT
Sets the color of the light modules of the finder pattern. See *--dark* for a description of allowed values.

--format-dark FORMAT_DARK
Sets the color of the dark modules of the format information. See *--dark* for a description of allowed values.

--format-light FORMAT_LIGHT
Sets the color of the light modules of the format information. See *--dark* for a description of allowed values.

--quiet-zone QUIET_ZONE
Sets the color of the quiet zone (border). See *--dark* for a description of allowed values.

--separator SEPARATOR
Sets the color of the separator. See *--dark* for a description of allowed values.

--timing-dark TIMING_DARK
Sets the color of the dark modules of the timing pattern. See *--dark* for a description of allowed values.

--timing-light TIMING_LIGHT
Sets the color of the light modules of the timing pattern. See *--dark* for a description of allowed values.

--version-dark VERSION_DARK
Sets the color of the dark modules of the version information. See *--dark* for a description of allowed values.

--version-light VERSION_LIGHT
Sets the color of the light modules of the version information. See *--dark* for a description of allowed values.

15.1.3.4 SVG Options

--no-classes
Omits the (default) SVG classes

--no-xmldecl
Omits the XML declaration header

--no-namespace
Indicates that the SVG document should have no SVG namespace declaration

--no-newline
Indicates that the SVG document should have no trailing newline

--title TITLE
Specifies the title of the SVG document

--desc DESC
Specifies the description of the SVG document

--svgid SVGID
Indicates the ID of the <svg/> element

--svgclass SVGCLASS
Indicates the CSS class of the <svg/> element (default: 'segno'). An empty string omits the attribute.

--lineclass LINECLASS
: Indicates the CSS class of the <path/> elements. An empty string omits the attribute.

--no-size
: Indicates that the SVG document should not have “width” and “height” attributes

--unit UNIT
: Indicates SVG coordinate system unit

--svgversion SVGVERSION
: Indicates the SVG version

--svgencoding ENCODING
: Specifies the encoding of the document

--draw-transparent
: Indicates if invisible paths should be added to the SVG document. By default all transparent paths are omitted.

15.1.3.5 PNG Options

--dpi DPI
: Sets the DPI value of the PNG file

15.1.4 Exit Status

segno exits 0 on success, and >0 if an error occurs.

15.1.5 Examples

```
$ segno "Up jumped the devil"
```

Prints a 2-Q QR code to the terminal

```
$ segno -o=yesterday.png "Yesterday"
```

Saves the 1-Q QR code as PNG image.

```
$ segno -o=fool.svg --title="Example QR code" "The Fool on the Hill"
```

Saves the 2-Q QR code as SVG document with the given title.

```
$ segno -o=a-day-in-the-life.svg --scale=10 --dark darkblue "A Day in the Life"
```

Saves the 1-L QR code as SVG document, using a scaling factor of 10 and the dark modules use the color “darkblue” instead of black.

```
$ segno -o rain.png -s 4 --dark "#003399" --micro RAIN
```

Saves the Micro QR Code (M2-M) as PNG image, using the color #003399 for dark modules. Each module corresponds to 4 x 4 pixels because the scaling factor was set to 4.

CHAPTER

SIXTEEN

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.

16.1 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'],
    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__XXXXXXXX_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

SEVENTEEN

API

17.1 Module contents

QR Code and Micro QR Code implementation.

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

exception segno.**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.

This exception is inherited from `ValueError` and is only raised if the data does not fit into the provided (Micro) QR Code version.

Basically it is sufficient to catch a `ValueError`.

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 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 or None*) – Error correction level. If `None` (default), error correction level `L` is used (note: Micro QR Code version M1 does not support any error correction. If an explicit error correction 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 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 or None*) – "numeric", "alphanumeric", "byte", "kanji" or "hanzi". If the value is None (default) the appropriate mode will automatically be determined. If *version* refers to a Micro QR code, this function may raise a ValueError if the provided *mode* is not supported.

  The *mode* parameter is case insensitive.

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
hanzi	1 - 40

Note: The Hanzi mode may not be supported by all QR code readers since it is not part of ISO/IEC 18004:2015(E). For this reason, this mode must be specified explicitly by the user:

```
import segno
qr = segno.make('', mode='hanzi')
```

- **mask** (*int or None*) – Data mask. If the value is None (default), the appropriate data mask is chosen automatically. If the *mask* parameter is provided, this function may raise a ValueError if the mask is invalid.
- **encoding** (*str or 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 (default) 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 `ValueError` or *DataOverflowError*: In case the data does not fit into a (Micro) QR Code or it does not fit into the provided `version`.

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 using the Structured Append mode.

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 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*

`class` `segno.`**`QRCode`**(*code*)

Represents a (Micro) QR Code.

`matrix`

Returns the matrix.

Return type tuple of `bytearray` instances.

`mask`

Returns the data mask pattern reference

Return type int

`property version`

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

Return type str or int

property 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)

Return type str

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

Return type str or None

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

Return type str

property default_border_size
Indicates the default border size aka quiet zone.

QR Codes have a quiet zone of four light modules, while Micro QR Codes have a quiet zone of two light modules.

Return type int

property is_micro
Indicates if this QR code is a Micro QR code

Return type bool

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)

matrix_iter(*scale=1*, *border=None*, *verbose=False*)
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')
>>> width, height = qr.symbol_size(scale=2)
>>> res = []
>>> # Scaling factor 2, default border
>>> for row in qr.matrix_iter(scale=2):
```

(continues on next page)

(continued from previous page)

```
>>>     res.append([col == 0x1 for col in row])
>>> width == len(res[0])
True
>>> height == len(res)
True
```

If *verbose* is `True`, the iterator returns integer constants which indicate the type of the module, i.e. `segno.consts.TYPE_FINDER_PATTERN_DARK`, `segno.consts.TYPE_FINDER_PATTERN_LIGHT`, `segno.consts.TYPE_QUIET_ZONE` etc.

To check if the returned module type is dark or light, use:

```
if mt >> 8:
    print('dark module')

if not mt >> 8:
    print('light module')
```

Parameters

- **scale** (*int*) – The scaling factor (default: `1`).
- **border** (*int*) – The size of border / quiet zone or `None` to indicate the default border.
- **verbose** (*bool*) – Indicates if the type of the module should be returned instead of `0x1` and `0x0` values. See `segno.consts` for the return values. This feature is currently in EXPERIMENTAL state.

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

show(*delete_after=20*, *scale=10*, *border=None*, *dark='#000'*, *light='#fff'*)

Displays this QR code.

This method is mainly intended for debugging purposes.

This method saves the QR code as an image (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** (*int or None*) – Time in seconds to wait till the temporary file is deleted.
- **scale** (*int*) – Integer indicating the size of a single module.
- **border** (*int or None*) – Integer indicating the size of the quiet zone. If set to `None` (default), the recommended border size will be used.
- **dark** – The color of the dark modules (default: black).
- **light** – The color of the light modules (default: white).

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, the different *xmldecl* and *nl* default values, and the additional parameters `encode_minimal` and `omit_charset`, this method uses the same parameters as the usual SVG serializer, see `save()` and the available *SVG parameters*

Note: In order to embed a SVG image in HTML without generating a file, the `svg_inline()` method could serve better results, as it usually produces a smaller output.

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)
- **nl** (*bool*) – Indicates if the document should have a trailing newline (default: `False`)

Return type str

svg_inline(***kw*)

Returns a SVG representation which is embeddable into HTML5 contexts.

Due to the fact that HTML5 directly supports SVG, various elements of a SVG document can or should be suppressed (i.e. the XML declaration and the SVG namespace).

This method returns a string that can be used in an HTML context.

This method uses the same parameters as the usual SVG serializer, see `save()` and the available *SVG parameters*

The returned string can be used directly in Jinja / Django templates, provided the `safe` filter is used:

```
<div>{{ qr.svg_inline(dark='#228b22', scale=3) | safe }}</div>
```

Return type str

png_data_uri(***kw*)

Converts the QR code into a PNG data URI.

Uses the same keyword parameters as the usual PNG serializer, see `save()` and the available *PNG parameters*

Return type str

terminal(*out=None*, *border=None*)

Serializes the matrix as ANSI escape code.

Under Windows, no ANSI escape sequence is generated but the Windows API is used *unless* `out` is a writable object or using WinAPI fails.

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 Micro QR Codes).

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). A value of `0` indicates that border should be omitted.
dark	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 (i.e. *SVG* and *PNG*) accept an alpha transparency value like `#RRGGBBAA`.
light	A string or tuple representing a color for the light modules. See *dark* for valid values. The default value depends on the serializer. *SVG* uses no color (`None`) for light modules by default, other serializers, like *PNG*, use "white" as default light color.

Module Colors

Name	Description
finder_dark	Color of the dark modules of the finder patterns Default: undefined, use value of "dark"
finder_light	Color of the light modules of the finder patterns Default: undefined, use value of "light"
data_dark	Color of the dark data modules Default: undefined, use value of "dark"
data_light	Color of the light data modules. Default: undefined, use value of "light".
version_dark	Color of the dark modules of the version information. Default: undefined, use value of "dark".
version_light	Color of the light modules of the version information, Default: undefined, use value of "light".
format_dark	Color of the dark modules of the format information. Default: undefined, use value of "dark".
format_light	Color of the light modules of the format information. Default: undefined, use value of "light".
alignment_dark	Color of the dark modules of the alignment patterns. Default: undefined, use value of "dark".
alignment_light	Color of the light modules of the alignment patterns. Default: undefined, use value of "light".
timing_dark	Color of the dark modules of the timing patterns. Default: undefined, use value of "dark".
timing_light	Color of the light modules of the timing patterns. Default: undefined, use value of "light".
separator	Color of the separator. Default: undefined, use value of "light".
dark_module	Color of the dark module (a single dark module which occurs in all QR Codes but not in Micro QR Codes. Default: undefined, use value of "dark".
quiet_zone	Color of the quiet zone / border. Default: undefined, use value of "light".

Scalable Vector Graphics (SVG)

All *common keywords* and *module colors* are supported.

Name	Description
out	Filename or `io.BytesIO`
kind	"svg" or "svgz" (to create a gzip compressed SVG)
scale	integer or float
dark	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".
light	Default value `None`. If this parameter is set to another value, the resulting image will have another path which is used to define the color of the light modules. 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.
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` Indicates 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.
draw_transparent	Indicates if transparent SVG paths should be added to the graphic (default: `False`)
nl	Indicates if the document should have a trailing newline (default: `True`)

Portable Network Graphics (PNG)

This writes either a grayscale (maybe with transparency) PNG (color type 0) or a palette-based (maybe with transparency) image (color type 3). If the dark / light values are `None`, white or black, the serializer chooses the more compact grayscale mode, in all other cases a palette-based image is written.

All *common keywords* and *module colors* are supported.

Name	Description
out	Filename or `io.BytesIO`
kind	"png"
scale	integer
dark	Default: "#000" (black) `None` is a valid value iff light is not `None`. If set to `None`, the dark modules become transparent.
light	Default value "#fff" (white) See keyword "dark" 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".

Encapsulated PostScript (EPS)

All *common keywords* are supported.

Name	Description
out	Filename or `io.StringIO`
kind	"eps"
scale	integer or float
dark	Default: "#000" (black)
light	Default value: `None` (transparent light modules)

Portable Document Format (PDF)

All *common keywords* are supported.

Name	Description
out	Filename or `io.BytesIO`
kind	"pdf"
scale	integer or float
dark	Default: "#000" (black)
light	Default value: `None` (transparent light modules)
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)

Aside of "scale", all *common keywords* are supported.

Name	Description
out	Filename or `io.StringIO`
kind	"txt"
dark	Default: "1"
light	Default: "0"

ANSI escape code

Supports the "border" keyword, only!

Name	Description
out	Filename or `io.StringIO`
kind	"ans"

Portable Bitmap (PBM)

All *common keywords* are supported.

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)

All *common keywords* are supported.

Name	Description
out	Filename or `io.BytesIO`
kind	"pam"
scale	integer
dark	Default: "#000" (black).
light	Default value "#fff" (white). Use `None` for transparent light modules.

Portable Pixmap (PPM)

All *common keywords* and *module colors* are supported.

Name	Description
out	Filename or `io.BytesIO`
kind	"ppm"
scale	integer
dark	Default: "#000" (black).
light	Default value "#fff" (white).

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.

All *common keywords* are supported.

Name	Description
out	Filename or `io.StringIO`
kind	"tex"
scale	integer or float
dark	LaTeX color name (default: "black"). The color is written "at it is", please 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 the LaTeX environment.

X BitMap (XBM)

All *common keywords* are supported.

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

X PixMap (XPM)

All *common keywords* are supported.

Name	Description
out	Filename or `io.StringIO`
kind	"xpm"
scale	integer
dark	Default: "#000" (black).
light	Default value "#fff" (white) `None` indicates transparent light modules.
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.BytesIO` or `io.StringIO` stream which don't have a `name` attribute.
- **kind** (*str*) – Default `None`. 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). The value is case insensitive.
- **kw** – Any of the supported keywords by the specific serializer.

class segno.**QRCodeSequence**

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

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

terminal(*out=None*, *border=None*)

Serializes the sequence of QR codes as ANSI escape code.

See *QRCode.terminal()* for details.

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

Saves the sequence of QR codes 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.

17.2 High level QR Code factories

Additional factory functions for common QR codes.

Aside from *make_epc_qr()*, the factory functions 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 or iterable of strings*) – The email address (recipient). Multiple values are allowed.
- **cc** (*str, iterable of strings, or None*) – The carbon copy recipient. Multiple values are allowed.
- **bcc** (*str, iterable of strings, or None*) – The blind carbon copy recipient. Multiple values are allowed.
- **subject** (*str or None*) – The subject.
- **body** (*str or None*) – The message body.

Return type *segno.QRCode*

segno.helpers.**make_epc_qr**(*name*, *iban*, *amount*, *text=None*, *reference=None*, *bic=None*, *purpose=None*, *encoding=None*)

Creates and returns an European Payments Council Quick Response Code (EPC QR Code) version 002.

The returned *segno.QRCode* uses always the error correction level "M" and utilizes max. version 13 to fulfill the constraints of the EPC QR Code standard.

Note: Either the `text` or `reference` must be provided but not both

Note: Neither the IBAN, BIC, nor remittance reference number or any other information is validated (aside from checks regarding the allowed string lengths).

Parameters

- **name** (*str*) – Name of the recipient.
- **iban** (*str*) – International Bank Account Number (IBAN)
- **amount** (*int, float, decimal.Decimal*) – The amount (in EUR) to transfer. The currency is always Euro, no other currencies are supported.
- **text** (*str*) – Remittance Information (unstructured)
- **reference** (*str*) – Remittance Information (structured)
- **bic** (*str*) – Bank Identifier Code (BIC). Optional, only required for non-EEA countries.
- **purpose** (*str*) – SEPA purpose code.

- **encoding** (*str or int*) – By default, this function tries to find the best, minimal encoding. If another encoding should be used, the encoding name or the encoding constant (an integer) can be provided: `1`: "UTF-8", `2`: "ISO 8859-1", `3`: "ISO 8859-2", `4`: "ISO 8859-4", `5`: "ISO 8859-5", `6`: "ISO 8859-7", `7`: "ISO 8859-10", `8`: "ISO 8859-15"

  The encoding is case-insensitive.

Return type *segno.QRCode*

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

Return type str

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 or None*) – Designates a text string to be set as the kana name in the phonebook

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

- **pobox** (*str or None*) – P.O. box (address information).
- **roomno** (*str or None*) – Room number (address information).
- **houseno** (*str or None*) – House number (address information).
- **city** (*str or None*) – City (address information).
- **prefecture** (*str or None*) – Prefecture (address information).
- **zipcode** (*str or None*) – Zip code (address information).
- **country** (*str or 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*, *photo_uri=None*)

Creates a QR code which encodes a vCard version 3.0.

Only a subset of available *vCard 3.0 properties <https://tools.ietf.org/html/rfc2426>* 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 of strings, or None*) – E-mail address. Multiple values are allowed.
- **phone** (*str, iterable of strings, or None*) – Phone number. Multiple values are allowed.
- **fax** (*str, iterable of strings, or None*) – Fax number. Multiple values are allowed.
- **videophone** (*str, iterable of strings, or None*) – Phone number for video calls. Multiple values are allowed.
- **memo** (*str or None*) – A notice for the contact.
- **nickname** (*str or None*) – Nickname.
- **birthday** (*str, datetime.date or None*) – Birthday. If a string is provided, it should encode the date as `YYYY-MM-DD` value.
- **url** (*str, iterable of strings, or None*) – Homepage. Multiple values are allowed.
- **pobox** (*str or None*) – P.O. box (address information).
- **street** (*str or None*) – Street address.
- **city** (*str or None*) – City (address information).
- **region** (*str or None*) – Region (address information).
- **zipcode** (*str or None*) – Zip code (address information).
- **country** (*str or None*) – Country (address information).
- **org** (*str or None*) – Company / organization name.

- **lat** (*float or None*) – Latitude.
- **lng** (*float or None*) – Longitude.
- **source** (*str or None*) – URL where to obtain the vCard.
- **rev** (*str, datetime.date or None*) – Revision of the vCard / last modification date.
- **title** (*str, iterable of strings, or None*) – Job Title. Multiple values are allowed.
- **photo_uri** (*str, iterable of strings, or None*) – Photo URI. 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 3.0 properties <https://tools.ietf.org/html/rfc2426>* 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 of strings, or None*) – E-mail address. Multiple values are allowed.
- **phone** (*str, iterable of strings, or None*) – Phone number. Multiple values are allowed.
- **fax** (*str, iterable of strings, or None*) – Fax number. Multiple values are allowed.
- **videophone** (*str, iterable of strings, or None*) – Phone number for video calls. Multiple values are allowed.
- **memo** (*str or None*) – A notice for the contact.
- **nickname** (*str or None*) – Nickname.
- **birthday** (*str, datetime.date or None*) – Birthday. If a string is provided, it should encode the date as `YYYY-MM-DD` value.
- **url** (*str, iterable of strings, or None*) – Homepage. Multiple values are allowed.
- **pobox** (*str or None*) – P.O. box (address information).
- **street** (*str or None*) – Street address.
- **city** (*str or None*) – City (address information).
- **region** (*str or None*) – Region (address information).
- **zipcode** (*str or None*) – Zip code (address information).
- **country** (*str or None*) – Country (address information).

- **org** (*str or None*) – Company / organization name.
- **lat** (*float or None*) – Latitude.
- **lng** (*float or None*) – Longitude.
- **source** (*str or None*) – URL where to obtain the vCard.
- **rev** (*str, datetime.date or None*) – Revision of the vCard / last modification date.
- **title** (*str, iterable of strings, or None*) – Job Title. Multiple values are allowed.
- **photo_uri** (*str, iterable of strings, or 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 or None*) – The password.
- **security** (*str or 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 or None*) – The password.
- **security** (*str or 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

EIGHTEEN

PYTHON 2 STATEMENT

While Segno is a Python 3-first package, Python 2.7 is still supported after January 2020.

There are no plans to drop support for Python 2, Segno supports it as long as technically possible.

CHAPTER

NINETEEN

CHANGES

19.1 1.3.0 – 2020-08-31

- Fixed #84: CLI `--encoding` was used for the encoding of SVG documents and not for the encoding of the QR code. Added `--svgencoding` to specify the encoding of SVG documents.

19.2 1.2.1 – 2020-08-27

- Improved API docs
- Added support for type hints for the public API (stub files)

19.3 1.2.0 – 2020-08-18

- Added `QRCode.svg_inline` method which returns a string which can be used to embed the SVG directly in HTML pages.
- Improved documentation <https://segno.readthedocs.org/>
- Improved code quality
- Switched from tox to nox
- Utilize flake8
- Moved metadata from setup.py to setup.cfg

19.4 1.1.0 – 2020-08-05

- Support for Hanzi mode, implemented by Shi Yan
- Fixed #81: Wrong character count in Kanji mode if the user provided the QR code data as bytes.
- Improved documentation <https://segno.readthedocs.org/>
- Improved API docs

19.5 1.0.2 – 2020-07-30

- Fixed error in Kanji encoding: Data was incomplete. Again, discovered by Shi Yan
- Better test coverage for Kanji encoding

19.6 1.0.1 – 2020-07-28

- Fixed wrong information about character count in Kanji mode (discovered and fixed by Shi Yan)
- Fixed #72: Encodings for `helpers.make_epc_qr` may be specified by name or by a numeric constant.
- Added support for Netpbm PPM images.
- Documentation improvements (also thanks to James Addison for pr #73)
- Removed "version" parameter from `encoder.prepare_data` (does not belong to the public API anyway)

19.7 1.0.0 – 2020-02-14

- Removed support for `color` / `background` keywords (deprecated in 0.4.0). Use `dark` and `light`.
- Reintroduced `segno.DataOverflowError` (inherited from `ValueError`) to indicate that the provided data does not fit into the provided (Micro) QR Code parameters.
- Documentation improvements
- Although this lib made backwards incompatible changes since version 0.1.0, the changes should be clear since almost all changes were made very conservative with backwards compatibility in mind. Even early adopters should find a clear update path. This version marks a stable API acc. to Semantic Versioning.
- The initial stable release after nearly four years of development. Happy valentine ;)

19.8 0.4.0 – 2020-01-21

- Removed deprecated functions, modules etc. See #56, #57, #59, #61, #67.
- Deprecated usage of keywords "color" and "background". Replacements: "dark" and "light". See #60. The deprecated keywords will be removed in 1.0.0.
- Minor performance improvements for writing SVG (at least for Py 3.7, YMMV).
- Documentation improvements

19.9 0.3.9 – 2020-01-19

- Fixed #71: Dark / light modules of the finder pattern may be interpreted wrong if set to `None`
- Removed segno.encoder, segno.writers and segno.utils from public API (fixes #69)
- Removed segno.colors (part of segno.writers now)
- Documentation improvements

19.10 0.3.8 – 2020-01-15

- Added support for multiple (more than two) colors to SVG (fixes #64)
- Fixed several test cases
- Removed `QRCodeError` and all derived exceptions from public API (still available but not thrown and they will be removed in 0.4.0)
- Documentation improvements

19.11 0.3.7 – 2020-01-09

- Documentation improvements: Added several examples, fixed docs
- Fixed #62: PNG serializer adds only those colors to the PLTE which are actually needed for the given (Micro) QR Code.
- Minor performance improvements

19.12 0.3.6 – 2020-01-06

- Backwards incompatibility change: QRCode.show() uses "dark" instead of "color" and "light" instead of "background" to define the color of the dark / light modules
- Backwards incompatibility change: All `segno.writers` use "dark" instead of "color" and "light" instead of "background". This does not affect normal users, but only users of the low level API.
- Changed the keyword for setting the color of the dark modules from "color" to "dark" and for setting the light modules from "background" to "light" The former keywords are still supported. Their usage will issue a DeprecationWarning in the future.
- Added `--dark` and `--light` to the command line interface, see point above. `` `--color` `` and `--background` are still supported.
- Fixed typos, improved documentation
- Deprecated `segno.moduletypes` (will be removed in release 0.4.0), moved all constants to `segno.consts`
- Deprecated usage of parameter "colormap" (introduced in 0.3.4). It still works but a deprecation warning is issued. Instead of:

```
colormap = {mt.TYPE_FINDER_PATTERN_DARK: 'darkred',
            mt.TYPE_ALIGNMENT_PATTERN_DARK: 'darkred',
            mt.TYPE_TIMING_DARK: 'darkred',
            mt.TYPE_DARKMODULE: 'darkred',
            mt.TYPE_DATA_DARK: 'darkorange',
            mt.TYPE_DATA_LIGHT: 'yellow',
            mt.TYPE_FORMAT_DARK: 'darkred'}

qr.save('qrcode.png', scale=5, colormap=colormap)
```

use:

```
qr.save('qrcode.png', scale=5, dark='darkred', data_dark='darkorange',
        data_light='yellow')
```

See Colorful QR Codes for a description of available module names.

19.13 0.3.5 – 2020-01-03

- Added support for colorful (more than two colors) QR Codes to the CLI script (fixes #58).
- Fixed Read the Docs build
- Improved documentation
- Minor performance and code improvements.

19.14 0.3.4 – 2020-01-02

- Fixed issue #54: After last change (see 0.3.3), white background with transparent QR Code did not work. Enhanced test suite to cover all possible inputs for PNG grayscale mode
- Removed interpretation of `addad` from PNG serializer. Contradicts the claim to create small images by default. It still belongs to the function signature but will be removed in release 0.4.0
- The option `--no-ad` (CLI) is still available but ignored and will be removed in release 0.4.0. Removed the option from man page.
- Added option to PNG serializer to provide more than two colors. Each module type may have its own color.
- Added support for EPC QR Codes.
- Fixed bug in `helpers.make_vcard_data` function (the "source" URL was not used, but the usual URL was added to the SOURCE field)
- Better test coverage for the `segno.helpers` module

19.15 0.3.3 – 2019-12-29

- Fixed issue #54: PNGs with white color and transparent background were rendered as transparent PNG with a *black* QR Code.
- Removed test environments CPython 3.4 and 3.6 from tox
- Improved documentation
- Refactored source code
- Added test cases
- Fixed bugs in `helpers.make_vcard_data` function (superfluous semicolon in birthday line, check geo coordinates)
- Renamed `utils.matrix_iter_detail` into `utils.matrix_iter_verbose`. Kept `matrix_iter_detail` for backwards compatibility (deprecated, will be removed in release 0.4.0)
- Moved module constants from `segno.utils` into `segno.moduletypes`, Constants from `segno.utils` will be removed in release 0.4.0.
- Added option `verbose` (default: `False`) to `segno.QRCode.matrix_iter()` which returns an iterator which provides information about the module type (i.e. quiet zone, dark data module, light data module).

19.16 0.3.2 – 2019-07-15

- Performance improvements
- Added man page for the CLI (fixes #41)
- Added more documentation and examples
- Fixed missing charts of <https://segno.readthedocs.io/en/stable/comparison-qrcode-libs.html>
- Added PyQRCodeNG <https://pypi.org/project/PyQRCodeNG/> to comparison table
- Updated CSS for a better layout of tables with a lot of content
- Removed deprecated functions `encoder.score_n1`, `encoder.score_n2`, `encoder.score_n3`, and `encoder.score_n4` (they didn't belong to the public API anyway)
- Fixed Read the Docs build

19.17 0.3.1 – 2019-07-15

- See 0.3.2

19.18 0.3.0 – 2019-06-25

- Performance improvements (evaluation of mask scores)
- Faster PNG output
- Faster `utils.matrix_iter` (which improves several writers, i.e. PNG)
- Deprecation of `encoder.score_n1`, `encoder.score_n2`, `encoder.score_n3`, and `encoder.score_n4`. Use `encoder.mask_scores` or `encoder.evaluate_mask`.

19.19 0.2.9 – 2019-04-24

- Fixed typos
- PDF serializer: Added support for stroke and background color, initial code contributed by Serge Morel (pr #52).

19.20 0.2.8 – 2018-10-17

- Fixed #45: CLI does not raise exceptions but indicates errors with return code 1 and writes the error message to `sys.stderr`
- Added experimental `utils.matrix_iter_detail()` function which returns an iterator over the matrix to distinguish different dark and light modules by their function (i.e. separator, finder pattern etc.)
- Minor performance improvements
- Removed Python 2.6 from test environment
- Added support for vCard TITLE attribute, contributed by Stefano Borini (pr #48)
- Added support for vCard PHOTO URI attribute, suggested by Arthur Reinhart

19.21 0.2.7 – 2018-02-18

- Fixed dist package

19.22 0.2.6 – 2018-02-18

- Updated and fixed docs
- Added PyPy 3 to test environment

19.23 0.2.5 – 2017-02-14

- Added experimental support for Structured Append (divide content into max. 16 QR Code symbols)
- Internal refactoring (i.e. segno/scripts/cmd.py -> segno/cli.py)
- Added `-s` shortcut to Segno's command line interface to provide the scaling factor
- Added `-b` shortcut to Segno's command line interface to provide the border / quiet zone
- CLI accepts unquoted, whitespace separated content: `segno "Comfortably Numb"` can be written as `segno Comfortably Numb`

19.24 0.2.4 – 2017-01-31

- Fixed #33: Some Micro QR Codes may be unreadable due to wrong format information. Further, M1 and M3 codes may be wrong due to wrong encoding of final data symbol character (8 bits instead of (correct) 4 bits). Thanks to Nicolas Boullis for the bug report, initial fix, tests and patience.
- Fixed #34: Change default error level from "M" to "L" to avoid surprises that the content does not fit into the provided version. This change is somewhat backwards incompatible.
- Fixed #35: Check of user supplied mask pattern index was wrong.
- Fixed #36: Wrong placement of codeword in M1 and M3 symbols.
- Fixed #37: Generation of M1 / M3 symbols fail if the data modules are completely filled.
- Fixed #38: Optimized mask pattern choosing algorithm: If the user supplied a preferred mask, the mask evaluation step is skipped and the preferred mask is chosen
- Added more internal checks to ensure correct (Micro) QR Codes; provided helpful exceptions
- Removed `writers.get_writable` (replaced by `writers.writable`)
- Added support for serializing QR Codes as XBM (X BitMap) (supports black / white images)
- Added support for serializing QR Codes as XPM (X PixMap) (supports colors and transparency)
- Added support for encoding contact information as vCard version 3.0 (`segno.helpers`)
- Added -V shortcut to Segno's command line script to show version information
- Better test coverage for command line script
- Better test coverage for M1 and M3 symbols

19.25 0.2.3 – 2016-10-17

- Fixed #27: Email URI is wrong if CC or BCC is used.
- Fixed #32: Don't add version attribute if SVG >= 2.0
- Deprecated `writers.get_writable`; use `writers.writable` `writers.writable` closes file-like objects automatically (if necessary); replace `writable, must_close = writers.get_writable(filename_or_buffer, mode)` with `with writers.writable(filename_or_buffer, mode) as f`
- Added option to PNG serializer to specify an optional DPI value (thanks to Markus Ueberall for support)

- Added PAM (Portable Arbitrary Map) as serialization format (supports colors and transparency)

19.26 0.2.2 – 2016-09-21

- Command line script reports Segno's version (`--ver`) and the version is also mentioned in the help message (`-h`) (#24)
- Support for creating email addresses or complete messages (`segno.helpers`)
- Internal optimizations and more correct minimal version finding (#26)

19.27 0.2.1 – 2016-09-15

- Fixed Python packaging (source distribution did not work), again

19.28 0.2.0 – 2016-09-15

- Fixed Python packaging

19.29 0.1.9 – 2016-09-15

- Added "color" parameter to the LaTeX serializer to define the color of the dark modules.
- Fixed serious issue #23: Segno creates invalid QR Codes if boost_error is not disabled (enabled by default)

19.30 0.1.8 – 2016-09-14

- Removed `utils.matrix_with_border_iter`
- Fixed #21 (type error while writing to terminal under Windows)
- Added option to serialize QR Codes as LaTeX vector graphic
- Added module `segno.helpers` which provides additional factory functions to create common QR Codes like a WIFI configuration, a geo location or MeCard

19.31 0.1.7 – 2016-09-04

- Changed API: Added a feature to increase the error correction level if it fits. Disable this feature via `boost_error=False` (#16)
- Added `--no-error-boost` to the command line script to disable error correction level incrementation (#17)
- Command line script: Internal changes and better test coverage
- Added tests for issue #18
- Added PBM (P1 and P4) serialization.

- Deprecated `utils.matrix_with_border_iter`, use `utils.matrix_iter`
- `utils.matrix_with_border_iter` will be removed in the next release
- API change: `QRCode.matrix_iter(border)` -> `QRCode.matrix_iter(scale=1, border=None)`

19.32 0.1.6 – 2016-08-25

- Fixed setup

19.33 0.1.5 – 2016-08-24

- Added QRCode.matrix_iter(border) which returns an iterator over the matrix and includes the border (as light modules).
- Invalid (empty) SVG identifiers / class names are ignored and do not result into an invalid SVG document (issue #8).
- SVG serializer: If `unit` was set to `None`, an invalid SVG document was generated (issue #14).
- Better command line support:
  - The command line script recognizes all SVG options (#9)
  - Added `--mode`/`-m`, renamed `--mask`/`-m` to `--pattern`/`-p` (issue #10)
  - The script used an empty string as default value for the data to encode. The data to encode has no default value anymore (issue #11)
  - Added `--no-ad` to omit the comment `Software` in PNG images (issue #12)

19.34 0.1.4 – 2016-08-21

- Better terminal output
- Fixed issue #5: QRCode.terminal() uses a special output function (if it detects Windows) to support MS Windows which may not support ANSI escape codes.

19.35 0.1.3 – 2016-08-20

- Added command line script "segno"
- Registered new file extension "ans" which serializes the QR Code as ANSI escape code (same output as QRCode.terminal())
- Removed deprecated methods "eps", "svg", "png", "pdf", and "txt" from segno.QRCode
- Switched from nose tests to py.test

19.36 0.1.2 – 2016-08-17

- Updated docs
- Backward incompatible changes: Deprecated “eps”, “svg”, “png”, “pdf”, and “txt” methods from QRCode. Use QRCode.save. Methods will be removed in 0.1.3
- Fixed issue #3 (M1 and M3 codes may have undefined areas)
- Fixed issue #4 (wrong ‘error’ default value for encoder.encode(), factory function segno.make() wasn’t affected)

19.37 0.1.1 – 2016-08-14

- Initial release

CHAPTER

TWENTY

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

Symbols

D

E

I

M

P

Q

S

T

V