pyqrcode Documentation

Release 1.2

Michael Nooner

Jun 26, 2018

Contents

The pyqrcode module is a QR code generator that is simple to use and written in pure python. The module is compatible with Python 2.6, 2.7, and 3.x. The module automates most of the building process for you. Generally, QR codes can be created using only two lines of code!

Unlike many other generators, all of the automation can be controlled manually. You are free to set any or all of the properties of your QR code.

QR codes can be saved as SVG, EPS, PNG (by using the pypng module), and plain text. PIL is not used to render the image files. You can also display a QR code directly in a compatible terminal.

The pyqrcode module attempts to follow the QR code standard as closely as possible. The terminology and the encodings used in pyqrcode come directly from the standard. This module also follows the algorithm laid out in the standard.

Contents:

CHAPTER 1

Creating QR Codes

The QRCode object is designed to be smart about how it constructs QR codes. It can automatically figure out what mode and version to use to construct a QR code, based on the data and the amount error correction. The error correction level defaults to the highest possible level of error correction.

Below are some examples of creating QR Codes using the automatated system.

```
>>> url = pyqrcode.create('http://uca.edu')
>>> url = pyqrcode.create('http://uca.edu', error='L')
```

There are many situations where you might wish to have more fine grained control over how the QR Code is generated. You can specify all the properties of your QR code through the optional parameters of the `pyqrcode.create()` function. There are three main properties to a QR code.

The *error* parameter sets the error correction level of the code. Each level has an associated name given by a letter: L, M, Q, or H; each level can correct up to 7, 15, 25, or 30 percent of the data respectively. There are several ways to specify the level, see `pyqrcode.tables.error_level` for all the possible values. By default this parameter is set to 'H' which is the highest possible error correction, but it has the smallest available data capacity for a given version.

The *version* parameter specifies the size and data capacity of the code. Versions are any integer between 1 and 40. Where version 1 is the smallest QR code, and version 40 is the largest. By default, the object uses the data's encoding and error correction level to calculate the smallest possible version. You may want to specify this parameter for consistency when generating several QR codes with varying amounts of data. That way all of the generated codes would have the same size.

Finally, the *mode* parameter sets how the contents will be encoded. Three of the four possible encodings are available. By default, the object uses the most efficient encoding for the contents. You can override this behavior by setting this parameter. See `pyqrcode.tables.modes` for a list of possible values for this parameter. A much longer discussion on modes can be found in the next section *Encoding Data*.

The code below constructs a QR code with 25% error correction, size 27, and forces the encoding to be binary (rather than numeric).

```
>>> big_code = pyqrcode.create('0987654321', error='L', version=27, mode='binary')
```

CHAPTER 2

Encoding Data

The standard calls the data's encoding its *mode*. The QR code standard defines how to encode any given piece of data. There are four possible modes. This module supports three of them: numeric, alphanumeric, and binary.

Each mode is worse at encoding the QR code's contents. In other words, each mode will require more room in the QR code to store the data. How much data a code version can hold is dependent on what mode is used and the error correction level. For example, the binary encoding always requires more code words than the numeric encoding.

Because of this, it is *generally* better to allow the QRCode object to auto-select the most efficient mode for the code's contents.

Note: The QRCode object can automatically choose the best mode based on the data to be encoded. In general, it is best to just let the object figure it out for you.

2.1 Numeric Encoding

The numeric type is the most efficient way to encode digits. Problematically, the standard make no provisions for encoding negative or fractional numbers. This encoding is better than Alphanumeric, when you only have a list of digits.

To use this encoding, simply specify a string of digits as the data. You can also use a positive integer as the code's contents.

```
>>> number = pyqrcode.create(123456789012345)
>>> number2 = pyqrcode.create('0987654321')
```

2.2 Alphanumeric

The alphanumeric type is very limited in that it can only encode some ASCII characters. It encodes:

- Uppercase letters
- Digits 0-9
- The horizontal space
- Eight punctuation characters: $, %, *, +, -, ., /, and :

A complete list of the possible characters can be found in the `pyqrcode.tables.ascii_codes` dictionary. While limited, this encoding is much more efficient than using the binary encoding, in many cases. Luckily, the available characters will let you encode a URL.

```
>>> url = pyqrcode.create('http://uca.edu'.upper())
```

2.3 Kanji

The final mode allows for the encoding of Kanji characters. Denso Wave, the creators of the QR code, is a Japenese company. Hence, they made special provisions for using QR codes with Japenese text.

Only one python string encoding for Kanji characters is supported, shift-jis. The auto-detection algorithm will try to encode the given string as shift-jis. if the characters are supported, then the mode will be set to kanji. Alternatively, you can explicitly define the data's encoding.

```
>>> utf8 = ''.encode('utf-8')
>>> monty = pyqrcode.create(utf8, encoding='utf-8')
>>> python = pyqrcode.create('')
```

2.4 Binary

When all else fails the data can be encoded in pure binary. This encoding does not change the data in any way. Instead its pure bytes are represented directly in the QR code. This is the least efficient way to store data in a QR code. You should only use this as a last resort.

The quotation below must be encoded in binary because of the apostrophe, exclamation point, and the new line character. Notice, that the string's characters will not have their case changed.

```
>>> life = pyqrcode.create('''MR. CREOSOTE: Better get a bucket. I'm going to throw␣
↪up.
    MAITRE D: Uh, Gaston! A bucket for monsieur. There you are, monsieur.''')
```

CHAPTER 3

Rendering QR Codes

There are five possible formats for rendering the QR Code. The first is to render it as a string of 1's and 0's. Next, the code can be displayed directly in compatible terminals. There are also three image based renderers. All, but the first, allow you to set the colors used. They also take a scaling factor, that way each module is not rendered as 1 pixel.

3.1 Text Based Rendering

The pyqrcode module includes a basic text renderer. This will return a string containing the QR code as a string of 1's and 0's, with each row of the code on a new line. A *data module* in the QR Code is represented by a 1. Likewise, 0 is used to represent the background of the code.

The purpose of this renderer is to allow users to create their own renderer if none of the built in renderers are satisfactory.

```
>>> number = pyqrcode.create(123)
>>> print(number.text())
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00001111111011110011111110000
00001000001000101010000010000
00001011101001010010111010000
00001011101010011010111010000
00001011101000100010111010000
00001000001001001010000010000
00001111111010101011111110000
00000000000001011000000000000
00000101110110101000100010000
00001011110001111101010010000
00000111111011100101001000000
00001001100011010011110010000
00001111111001101011001110000
```

(continues on next page)

(continued from previous page)

```
000000000000010000000001100000
000011111110001111001001000000
000010000010110101100011000000
000010111010101100001010100000
000010111010011111110101000000
000010111010111010011010100000
000010000010010110011100000000
000011111110000110110110100000
000000000000000000000000000000
000000000000000000000000000000
000000000000000000000000000000
000000000000000000000000000000
```

3.2 Terminal Rendering

QR codes can be directly rendered to a compatible terminal in a manner readable by QR code scanners. The rendering is done using ASCII escape codes. Hence, most Linux terminals are supported. The QR code's colors can even be set.

```
>>> text = pyqrcode.create('Example')
>>> print(text.terminal())
>>> print(text.terminal(module_color='red', background='yellow'))
>>> print(text.terminal(module_color=5, background=123, quiet_zone=1))
```

Rendering colors in a terminal is a tricky business. Beyond the eight named colors, compatibility becomes problematic. With this in mind it is best to stick to the eight well known colors: black, red, green, yellow, blue, magenta, and cyan. Although, these colors are also supported on almost every color terminal: light gray, dark gray, light red, light green, light blue, light yellow, light magenta, light cyan, and white.

There are two additional named colors. The first is "default" it corresponds to the default background color of the terminal. The other is "reverse", this inverts the current background color. These are the default colors used by the terminal method.

The terminal method also support the 256 color scheme. This is the least transportable of the color schemes. To use this color scheme simply supply a number between 0 and 256. This number will act as an index to the terminal's color palette. What color that index actually corresponds to is system dependent. In other words, while most terminal emulators support 256 colors, the there is no way to tell what color will be actually displayed.

3.3 Image Rendering

There are three ways to get an image of the generated QR code. All of the renderers have a few things in common.

Each renderer takes a file path or writable stream and draws the QR code there. The methods should auto-detect which is which.

Each renderer takes a scale parameter. This parameter sets the size of a single *data module* in pixels. Setting this parameter to one, will result in each *data module* taking up 1 pixel. In other words, the QR code would be too small to scan. What scale to use depends on how you plan to use the QR code. Generally, three, four, or five will result in small but scanable QR codes.

QR codes are also supposed to have a *quiet zone* around them. This area is four modules wide on each side. The purpose of the quiet zone is to make scanning a printed area more reliable. For electronic usages, this may be unnecessary depending on how the code is being displayed. Each of the renderers allows you to set the size of the quiet zone.

Many of the renderers, also, allow you to set the *module* and background colors. Although, how the colors are represented are renderer specific.

3.3.1 XBM Rendering

The XBM file format is a simple black and white image format. The image data takes the form of a valid C header file. XBM rendering is handled via the `pyqrcode.QRCode.xbm()` method.

XMB's are natively supported by Tkinter. This makes displaying QR codes in a Tkinter application very simple.

```
>>> import pyqrcode
>>> import tkinter
>>> # Create and render the QR code
>>> code = pyqrcode.create('Knights who say ni!')
>>> code_xbm = code.xbm(scale=5)
>>> # Create a tk window
>>> top = tkinter.Tk()
>>> # Make generate the bitmap image from the redered code
>>> code_bmp = tkinter.BitmapImage(data=code_xbm)
>>> # Set the code to have a white background,
>>> # instead of transparent
>>> code_bmp.config(background="white")
>>> # Bitmaps are accepted by lots of Widgets
>>> label = tkinter.Label(image=code_bmp)
>>> # The QR code is now visible
>>> label.pack()
```

3.3.2 Scalable Vector Graphic (SVG)

The SVG renderer outputs the QR code as a scalable vector graphic using the `pyqrcode.QRCode.svg()` method.

The method draws the QR code using a set of paths. By default, no background is drawn, i.e. the resulting code has a transparent background. The default foreground (module) color is black.

```
>>> url = pyqrcode.create('http://uca.edu')
>>> url.svg('uca.svg', scale=4)
>>> # in-memory stream is also supported
>>> buffer = io.BytesIO()
>>> url.svg(buffer)
>>> # do whatever you want with buffer.getvalue()
>>> print(list(buffer.getvalue()))
```

You can change the colors of the data-modules using the *module_color* parameter. Likewise, you can specify a background using the *background* parameter. Each of these parameters take a HTML style color.

```
>>> url.svg('uca.svg', scale=4, background="white", module_color="#7D007D")
```

You can also suppress certain parts of the SVG document. In other words you can create a SVG fragment.

3.3.3 Encapsulated PostScript (EPS)

The EPS renderer outputs the QR code an encapsulated PostScript document using the `pyqrcode.QRCode.eps()` method. *This renderer does not require any external modules.*

The method draws the EPS document using lines of contiguous modules. By default, no background is drawn, i.e. the resulting code has a transparent background. The default module color is black. Note, that a scale of 1 equates to a module being drawn at 1 point (1/72 of an inch).

```
>>> qr = pyqrcode.create('Hello world')
>>> qr.eps('hello-world.eps', scale=2.5, module_color='#36C')
>>> qr.eps('hello-world2.eps', background='#eee')
>>> out = io.StringIO()
>>> qr.eps(out, module_color=(.4, .4, .4))
```

3.3.4 Portable Network Graphic (PNG)

The PNG renderer outputs the QR code as a portable network graphic file using the `pyqrcode.QRCode.png()` method.

Note: This renderer requires the pypng module.

```
>>> url = pyqrcode.create('http://uca.edu')
>>> with open('code.png', 'w') as fstream:
...     url.png(fstream, scale=5)
>>> # same as above
>>> url.png('code.png', scale=5)
>>> # in-memory stream is also supported
>>> buffer = io.BytesIO()
>>> url.png(buffer)
>>> # do whatever you want with buffer.getvalue()
>>> print(list(buffer.getvalue()))
```

Colors should be a list or tuple containing numbers between zero an 255. The lists should be of length three (for RGB) or four (for RGBA). The color (0,0,0) represents black and the color (255,255,255) represents white. A value of zero for the fourth element, represents full transparency. Likewise, a value of 255 for the fourth element represents full opacity.

By default, the renderer creates a QR code with the data modules colored black, and the background modules colored white.

```
>>> url.png('uca-colors.png', scale=6,
...         module_color=[0, 0, 0, 128],
...         background=[0xff, 0xff, 0xcc])
```

CHAPTER 4

PyQRCode Module Documentation

CHAPTER 5

PyQRCode

Contents

The pyqrcode module is a QR code generator that is simple to use and written in pure python. The module can automates most of the building process for creating QR codes. Most codes can be created using only two lines of code!

Unlike other generators, all of the helpers can be controlled manually. You are free to set any or all of the properties of your QR code.

QR codes can be saved as SVG, PNG (by using the pypng module), and plain text. They can also be displayed directly in most Linux terminal emulators. PIL is not used to render the image files.

The pyqrcode module attempts to follow the QR code standard as closely as possible. The terminology and the encodings used in pyqrcode come directly from the standard. This module also follows the algorithm laid out in the standard.

Homepage: https://github.com/mnooner256/pyqrcode

Documentation: http://pythonhosted.org/PyQRCode/

5.1 Requirements

The pyqrcode module only requires Python 2.6, Python 2.7, or Python 3. You may want to install pypng in order to render PNG files, but it is optional. Note, pypng is a pure python PNG writer which does not require any other libraries.

5.2 Installation

Installation is simple. It can be installed from pip using the following command:

```
$ pip install pyqrcode
```

Or from the terminal:

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

5.3 Usage

The pyqrcode module aims to be as simple to use as possible. Below is a simple example of creating a QR code for a URL. The code is rendered out as an svg file.

```
>>> import pyqrcode
>>> url = pyqrcode.create('http://uca.edu')
>>> url.svg('uca-url.svg', scale=8)
>>> url.eps('uca-url.eps', scale=2)
>>> print(url.terminal(quiet_zone=1))
```

The pyqrcode module, while easy to use, is powerful. You can set every property of the QR code. If you install the optional pypng module, you can render the code as a PNG image. Below is a more complex example:

```
>>> big_code = pyqrcode.create('0987654321', error='L', version=27, mode='binary')
>>> big_code.png('code.png', scale=6, module_color=[0, 0, 0, 128], background=[0xff,␣
↪0xff, 0xcc])
>>> big_code.show()
```

CHAPTER 6

Glossary

error

error level QR codes can use one of four possible error correction values. They are referred to by the letters: L, M, Q, and H. The *L* error correction level corresponds to 7% of the code can be corrected. The *M* error correction level corresponds to 15% of the code can be corrected. The *Q* error correction level corresponds to 25% of the code can be corrected. The *H* error correction level corresponds to 30% of the code can be corrected.

mode The encoding used to represent the data in a QR code. There are four possible encodings: binary, numeric, alphanumeric, kanji.

module

data module A square dot on a QR code. Generally, only the "black" dots count. The "white" squares are considered part of the background.

quiet zone An empty area around the QR code. The area is the background module in color. According to the standard this area should be four modules wide.

QR code

Quick Response code A two dimensional barcode developed by Denso Wave.

version A version is one of 40 different possible sizes a QR code comes in. The version of a QR Code determines it maximum possible data capacity.

CHAPTER 7

Requirements

The pyqrcode module only requires Python 2.6, 2.7, 3.x. You may want to install pypng in order to render PNG files, but it is optional. Note, pypng is a pure python PNG writer which does not require any other libraries.

CHAPTER 8

Installation

Installation is simple. PyQRCode can be installed from pip using the following command:

```
$ pip install pyqrcode
```

Or from the command line using:

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

CHAPTER 9

Usage

The pyqrcode module aims to be as simple to use as possible. Below is a simple example of creating a QR code for a URL. The code is rendered out as a black and white scalable vector graphics file.

```
>>> import pyqrcode
>>> url = pyqrcode.create('http://uca.edu')
>>> url.svg('uca-url.svg', scale=8)
>>> print(url.terminal(quiet_zone=1))
```

The pyqrcode module, while easy to use, is powerful. You can set all of the properties of the QR code. If you install the optional pypng library, you can also render the code as a PNG image. Below is a more complex example:

```
>>> big_code = pyqrcode.create('0987654321', error='L', version=27, mode='binary')
>>> big_code.png('code.png', scale=6, module_color=[0, 0, 0, 128], background=[0xff,␣
↪0xff, 0xcc])
```

CHAPTER 10

Developer Documentation

10.1 PyQRCode Tables

10.2 PyQRCode Builder Documentation

CHAPTER 11

Indices and tables

- genindex
- modindex
- search

Index