Django QR Code Documentation

Release 3.0.0

Philippe Docourt

Nov 27, 2021

Contents:

CHAPTER 1

Django QR Code

This is an application that provides tools for displaying QR codes on your Django site.

This application depends on the Segno QR Code generator library.

This app makes no usage of the Django models and therefore do not use any database.

Only Python >= 3.7 is supported.

1.1 Installation

1.1.1 Binary Package from PyPi

In order to use this app in a Django project, the simplest way is to install it from PyPi:

```
pip install django-qr-code
```

1.1.2 From the Source Code

In order to modify or test this app you may want to install it from the source code.

Clone the GitHub repository and then run:

```
pip install -r requirements.txt -r requirements-dev.txt
python manage.py collectstatic --no-input
```

1.2 Usage

Start by adding `qr_code` to your `INSTALLED_APPS` setting like this:

```
INSTALLED_APPS = (
    # ...,
    'qr_code',
)
```

You need to load the tags provided by this app in your template with:

```
{% load qr_code %}
```

The source code on GitHub contains a simple demo app. Please check out the templates folder for an example of template, and the setting and urls files for an example of configuration and integration.

1.2.1 Generating Inline QR Code in your HTML (qr_from_text)

The tag **`qr_from_text`** generates an embedded `svg` or `img` tag within the HTML code produced by your template.

The following renders a tiny "hello world" QR code with a `svg` tag:

```
{% qr_from_text "Hello World!" size="T" %}
```

Here is a medium "hello world" QR code with an `img` tag:

```
{% qr_from_text "Hello World!" size="m" image_format="png" error_correction="L" %}
```

1.2.2 QR Code Rendering Options

The `size` parameter gives the size of each module of the QR code matrix. It can be either a positive integer or one of the following letters:

- t or T: tiny (value: 6)
- s or S: small (value: 12)
- m or M: medium (value: 18)
- l or L: large (value: 30)
- h or H: huge (value: 48)

For PNG image format the size unit is in pixels, while the unit is 1 mm for SVG format.

Here is a "hello world" QR code using the version 12:

```
{% qr_from_text "Hello World!" size=8 version=12 %}
```

The `version` parameter is an integer from 1 to 40 that controls the size of the QR code matrix. Set to None to determine this automatically. The smallest, version 1, is a 21 x 21 matrix. The biggest, version 40, is 177 x 177 matrix. The size grows by 4 modules/side.

Here is a "hello world" QR code using a border of 6 modules:

```
{% qr_from_text "Hello World!" size=10 border=6 %}
```

The `border` parameter controls how many modules thick the border should be (the default is 4, which is the minimum according to the specs).

There are 4 error correction levels used for QR codes, with each one adding different amounts of "backup" data depending on how much damage the QR code is expected to suffer in its intended environment, and hence how much

error correction may be required. The correction level can be configured with the `error_correction` parameter as follows:

- l or L: error correction level L – up to 7% damage
- m or M: error correction level M – up to 15% damage
- q or Q: error correction level Q – up to 25% damage
- h or H: error correction level H – up to 30% damage

You may enforce the creation of a Micro QR Code with `micro=True`. The `micro` option defaults to `False`.

The `encoding` option controls the text encoding used in mode "byte" (used for any general text content). By default `encoding` is `UTF-8`. When set to `None`, the implementation (based on Segno) 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` option). The `encoding` parameter is case-insensitive.

The `boost_error` indicates whether the QR code encoding engine (Segno) tries to increase the error correction level if it does not affect the version. Error correction level is not increased when it impacts the version of the code.

The `eci` option 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.

Alternatively, you may use the `options` keyword argument with an instance of `QRCodeOptions` as value instead of listing every requested options. Here is an example of view:

```
from django.shortcuts import render
from qr_code.qrcode.utils import QRCodeOptions

def my_view(request):
    # Build context for rendering QR codes.
    context = dict(
        my_options=QRCodeOptions(size='t', border=6, error_correction='L'),
    )

    # Render the view.
    return render(request, 'my_app/my_view.html', context=context)
```

and an example of template for the view above:

```
{% qr_from_text "Hello World!" options=my_options %}
```

1.2.3 Generating URLs to QR Code Images (qr_url_from_text)

The **`qr_url_from_text`** tag generates an url to an image representing the QR code. It comes with the same options as `qr_from_text` to customize the image format (SVG or PNG), the size, the border, and the matrix size. It also has an additional option **`cache_enabled`** to disable caching of served image.

Here is a medium "hello world" QR code that uses an URL to serve the image in SVG format:

```
<img src="{% qr_url_from_text "Hello World!" %}" alt="Hello World!">
```

Here is a "hello world" QR code in version 10 that uses an URL to serve the image in PNG format:

```
<img src="{% qr_url_from_text "Hello World!" size=8 version=10 image_format='png' %}"␣
↪alt="Hello World!">
```

The image targeted by the generated URL is served by a view provided in `qr_code.urls`. Therefore, you need to include the URLs provided by `qr_code.urls` in your app in order to make this tag work. This can be achieved with something like this:

```
from django.conf.urls import include
from django.urls import path

urlpatterns = [
    path('qr_code/', include('qr_code.urls', namespace="qr_code")),
]
```

The QR code images are served via a URL named **`qr_code:serve_qr_code_image`**. You can customize the path under which the images are served (i.e. the path bound to the URL named `qr_code:serve_qr_code_image`) with the optionnal setting **`SERVE_QR_CODE_IMAGE_PATH`** which defaults to `images/serve-qr-code-image/`. Note that the trailing slash is mandatory and that defining this setting to an empty string leads to using the default value. The example below will serve any QR code image from `<base URL or your application>/qr-code-image/`:

```
# In urls.py
from django.conf.urls import include
from django.urls import path

urlpatterns = [
    path('', include('qr_code.urls', namespace='qr_code')),
]

# In your settings
SERVE_QR_CODE_IMAGE_PATH = 'qr-code-image/'
```

1.2.4 Special encoding modes with qr_from_data and qr_url_from_data

The tags **`qr_from_data`** and **`qr_url_from_data`** produce results similar to those of `qr_from_text` and `qr_url_from_text`, but they avoid converting everything to text (UTF-8 encoded by default, or any supported charset depending on `encoding` option).

The ISO/IEC 18004 standard defines four modes in order to encode the data as efficiently as possible.

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. The encoding engine detects (Segno) the numeric mode if the data is provided as string (e.g. '54') or integer (e.g. 54) to `qr_from_data` or `qr_url_from_data`.

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 $%*+-./:.

Kanji mode

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

Byte mode

The byte mode covers all data which cannot be represented by the other modes. When the `encoding` option is set to `None`, the encoding engine (Segno) detects, 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.

NOTE: When using `qr_from_text` or `qr_url_from_text`, the byte mode with UTF-8 encoding is forced by default . You may use the `encoding` option to change this behavior. It appears to be one of the most portable way of encoding text for proper decoding among the readers.

Examples

The following renders a tiny numeric QR code containing the value `2021` with a `svg` tag:

```
{% qr_from_data 2021 size="T" %}
```

Here is a micro QR code with an `img` tag containing the value `2021`:

```
{% qr_from_data 2021 micro=True image_format="png" %}
```

1.2.5 Caching Served Images

A large QR code (version 40) requires 0.2 second to be generated on a powerful machine (in 2018), and probably more than half a second on a really cheap hosting.

The image served by the *qr_code* app can be cached to improve performances and reduce CPU usage required to generate the QR codes. In order to activate caching, you simply need to declare a cache alias with the setting **`QR_CODE_CACHE_ALIAS`** to indicate in which cache to store the generated QR codes.

For instance, you may declare an additional cache for your QR codes like this in your Django settings:

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.locmem.LocMemCache',
    },
    'qr-code': {
        'BACKEND': 'django.core.cache.backends.locmem.LocMemCache',
        'LOCATION': 'qr-code-cache',
        'TIMEOUT': 3600
    }
}

QR_CODE_CACHE_ALIAS = 'qr-code'
```

The `QR_CODE_CACHE_ALIAS = 'qr-code'` tells the *qr_code* app to use that cache for storing the generated QR codes. All QR codes will be cached with the specified *TIMEOUT* when a non-empty value is set to `QR_CODE_CACHE_ALIAS`.

If you want to activate the cache for QR codes, but skip the caching for some specific codes, you can use the keyword argument `cache_enabled=False` when using `qr_url_from_text`.

Here is a "hello world" QR code in version 20 with an error correction level Q (25% of redundant data) that uses a URL to serve the image in SVG format, and disable caching for served image:

```
<img src="{% qr_url_from_text "Hello World!" size=8 version=20 error_correction="Q"␣
↪cache_enabled=False %}" alt="Hello World!">
```

1.2.6 Protecting Access to QR Code Images

The default settings protect the URLs that serve QR code images against external requests, and thus against possibly easy (D)DoS attacks.

Here are the available settings to manage the protection for served images:

```
from qr_code.qrcode import constants

QR_CODE_URL_PROTECTION = {
    constants.TOKEN_LENGTH: 30,                          # Optional random token␣
↪length for URL protection. Defaults to 20.
    constants.SIGNING_KEY: 'my-secret-signing-key',      # Optional signing key for␣
↪URL token. Uses SECRET_KEY if not defined.
    constants.SIGNING_SALT: 'my-signing-salt',           # Optional signing salt for␣
↪URL token.
    constants.ALLOWS_EXTERNAL_REQUESTS_FOR_REGISTERED_USER: True  # Tells whether a␣
↪registered user can request the QR code URLs from outside a site that uses this app.
↪ It can be a boolean value used for any user or a callable that takes a user as␣
↪parameter. Defaults to False (nobody can access the URL without the signature␣
↪token).
}
```

Signing Request URLs

By default, the application only serves QR code images for authenticated URLs (requests generated from your application and addressed to your application). The authentication uses a HMAC to sign the request query arguments. The authentication code is passed as a query argument named **`token`** which is automatically generated by `qr_url_from_text` or `qr_url_from_data`. Whenever the signature is invalid, the application returns a *HTTP 403 Permission denied* response when processing the request for serving a QR code image.

This mechanism ensures that, by default, nobody can send external requests to your application to obtain custom QR codes for free. This is especially useful if you display QR code URLs on public pages (no user authentication).

The `token` query argument is not mandatory and, when a request reaches the `qr_code:serve_qr_code_image` URL without that token, the protection mechanism falls back to the user authentication mechanism (see chapter below).

It is possible to explicitly remove the signature token that protects a specific URL with the parameter **`url_signature_enabled=False`**. Here is a "hello world" QR code that uses a URL to serve the image in SVG format without the `token` query argument :

```
<img src="{% qr_url_from_text "Hello World!" url_signature_enabled=False %}" alt=
↪"Hello World!">
```

The `token` parameter will not be part of the query string of the generated URL, making possible to build a simpler, predictable URL. However, this will trigger the user authentication mechanism (see chapter below).

Handling User Authentication when Serving QR Code Images

If you are interested in providing the QR code images as a service, there is a setting named **ALLOWS_EXTERNAL_REQUESTS_FOR_REGISTERED_USER** to grant access to some controlled users. This setting tells who can bypass the url signature token (see chapter above). It can be a boolean value used for any authenticated user, or a callable that takes a user as only parameter.

Setting the `ALLOWS_EXTERNAL_REQUESTS_FOR_REGISTERED_USER` option to `True` tells the application to serve QR code images to authenticated users only. Hence, to grant access to any authenticated user to generated images, you can use this in your settings:

```
from qr_code.qrcode import constants

QR_CODE_URL_PROTECTION = {
    constants.ALLOWS_EXTERNAL_REQUESTS_FOR_REGISTERED_USER: True
}
```

Setting the option `ALLOWS_EXTERNAL_REQUESTS_FOR_REGISTERED_USER` to a callable that always returns `True` (even for anonymous users) will allow anyone to access QR code image generation from outside your Django app. The following settings will grant access to anonymous users to generated images:

```
from qr_code.qrcode import constants

QR_CODE_URL_PROTECTION = {
    constants.ALLOWS_EXTERNAL_REQUESTS_FOR_REGISTERED_USER: lambda u: True
}
```

Please note that, if your service is available on the Internet, allowing anyone to generate any kind of QR code via your Django application - as shown above - might generate a very heavy load on your server.

1.2.7 QR Codes for Apps

Aside from generating a QR code from a given text, you can also generate codes for specific application purposes, that a reader can interpret as an action to take: open a mail client to send an e-mail to a given address, add a contact to your phone book, connect to a Wi-Fi, start a SMS, etc. See this documentation about what a QR code can encode.

Django QR Code proposes several utility tags to ease the generation of such codes, without having to build the appropriate text representation for each action you need. This remove the hassle of reading the specifications and handling all the required escaping for reserved chars.

Please note that some commands are common patterns, rather than formal specifications. Therefore, there is no guarantee that all QR code readers will handle them properly.

The following tags targeting apps are available:

- `qr_for_email` and `qr_url_for_email`
- `qr_for_tel` and `qr_url_for_tel`
- `qr_for_sms` and `qr_url_for_sms`
- `qr_for_geolocation` and `qr_url_for_geolocation`
- `qr_for_google_maps` and `qr_url_for_google_maps`
- `qr_for_youtube` and `qr_url_for_youtube`
- `qr_for_google_play` and `qr_url_for_google_play`
- `qr_for_mecard` and `qr_url_for_mecard`

- `qr_for_vcard` and `qr_url_for_vcard`
- `qr_for_wifi` and `qr_url_for_wifi`
- `qr_for_epc` and `qr_url_for_epc`
- `qr_for_contact` and `qr_url_for_contact` (legacy, do not use in new projects)

You could write a view like this:

```
from datetime import date
from django.shortcuts import render
from qr_code.qrcode.utils import MeCard, VCard, EpcData, WifiConfig, Coordinates,␣
↪QRCodeOptions


def application_qr_code_demo(request):
    # Use a MeCard instance to encapsulate the detail of the contact.
    mecard_contact = MeCard(
        name='Doe, John',
        phone='+41769998877',
        email='j.doe@company.com',
        url='http://www.company.com',
        birthday=date(year=1985, month=10, day=2),
        memo='Development Manager',
        org='Company Ltd'
    )

    # Use a VCard instance to encapsulate the detail of the contact.
    vcard_contact = VCard(
        name='Doe; John',
        phone='+41769998877',
        email='j.doe@company.com',
        url='http://www.company.com',
        birthday=date(year=1985, month=10, day=2),
        street='Cras des Fourches 987',
        city='Delémont',
        zipcode=2800,
        region='Jura',
        country='Switzerland',
        memo='Development Manager',
        org='Company Ltd'
    )

    # Use a WifiConfig instance to encapsulate the configuration of the connexion.
    wifi_config = WifiConfig(
        ssid='my-wifi',
        authentication=WifiConfig.AUTHENTICATION.WPA,
        password='wifi-password'
    )

    # Use a EpcData instance to encapsulate the data of the European Payments Council␣
↪Quick Response Code.
    epc_data = EpcData(
        name='Wikimedia Foerdergesellschaft',
        iban='DE33100205000001194700',
        amount=50.0,
        text='To Wikipedia'
    )
```

(continues on next page)

(continued from previous page)

```
    # Build coordinates instances.
    google_maps_coordinates = Coordinates(latitude=586000.32, longitude=250954.19)
    geolocation_coordinates = Coordinates(latitude=586000.32, longitude=250954.19,␣
↪altitude=500)

    # Build context for rendering QR codes.
    context = dict(
        mecard_contact=mecard_contact,
        vcard_contact=vcard_contact,
        wifi_config=wifi_config,
        epc_data=epc_data,
        video_id='J9go2nj6b3M',
        google_maps_coordinates=google_maps_coordinates,
        geolocation_coordinates=geolocation_coordinates,
        options_example=QRCodeOptions(size='t', border=6, error_correction='L'),
    )

    # Render the index page.
    return render(request, 'my_app/application_qr_code_demo.html', context=context)
```

Then, in your template, you can render the appropriate QR codes for the given context:

```
<h3>Add contact '{{ mecard_contact.name }}' to phone book</h3>
{% qr_for_mecard mecard=mecard_contact size='S' %}
<p>or:</p>
{% qr_for_contact mecard_contact size='S' %}
<p>or:</p>
{% qr_for_contact mecard_contact options=options_example %}

<h3>Add contact '{{ vcard_contact.name }}' to phone book</h3>
{% qr_for_vcard vcard=vcard_contact size='S' %}
<p>or:</p>
{% qr_for_contact vcard_contact size='S' %}
<p>or:</p>
{% qr_for_contact vcard_contact options=options_example %}

<h3>Configure Wi-Fi connexion to '{{ wifi_config.ssid }}'</h3>
<img src="{% qr_url_for_wifi wifi_config=wifi_config size='T' error_correction='Q' %}
↪">
<p>or:</p>
<img src="{% qr_url_for_wifi wifi_config size='T' error_correction='Q' %}">
<p>or:</p>
<img src="{% qr_url_for_wifi wifi_config options=options_example %}">

<h3>EPC QR Code'</h3>
<img src="{% qr_url_for_epc epc_data=epc_data %}">
<p>or:</p>
<img src="{% qr_url_for_epc epc_data size='H' %}">

<h3>Watch YouTube video '{{ video_id }}'</h3>
{% qr_for_youtube video_id image_format='png' size='T' %}
<p>or:</p>
{% qr_for_youtube video_id options=options_example %}

<h3>Open map at location: ({{ geolocation_coordinates }})</h3>
<img src="{% qr_url_for_geolocation coordinates=geolocation_coordinates %}">
```

(continues on next page)

(continued from previous page)

```
<p>or:</p>
<img src="{% qr_url_for_geolocation latitude=geolocation_coordinates.latitude␣
↪longitude=geolocation_coordinates.longitude altitude=geolocation_coordinates.
↪altitude %}">
<p>or:</p>
<img src="{% qr_url_for_geolocation latitude=geolocation_coordinates.latitude␣
↪longitude=geolocation_coordinates.longitude altitude=geolocation_coordinates.
↪altitude options=options_example %}">

<h3>Open Google Maps App at location: ({{ google_maps_coordinates }})</h3>
{% qr_for_google_maps coordinates=google_maps_coordinates %}
<p>or:</p>
{% qr_for_google_maps latitude=google_maps_coordinates.latitude longitude=google_maps_
↪coordinates.longitude %}
<p>or:</p>
{% qr_for_google_maps latitude=google_maps_coordinates.latitude longitude=google_maps_
↪coordinates.longitude options=options_example %}
```

Please check out the *demo application* to see more examples.

1.3 Notes

1.3.1 Image Formats

The SVG is the default image format. It is a vector image format so it can be scaled up and down without quality loss. However, it has two drawbacks. The size is not given in pixel, which can be problematic if the design of your website relies on a fixed width (in pixels). The format is less compact than PNG and results in a larger HTML content. Note that a base64 PNG is less compressible than a SVG tag, so it might not matter that much of you use HTML compression on your web server.

SVG has broad support now, and it will work properly on any modern web browser.

1.3.2 qr_from_text vs qr_url_from_text

The tag `qr_url_from_text` has several advantages over `qr_from_text`, despite the fact that it requires a bit more of writing:

- the generated HTML code does not contain heavy inline image data (lighter and cleaner content)
- the generated images can be cached (served with a separate HTML request)
- the HTML tag used to render the QR code is always an `<img>` tag, which may simplify CSS handling
- the HTML tag embedding the image is not generated for you, which allows for customization of attributes (*height*, *width*, *alt*)
- the page can be loaded asynchronously, which improves responsiveness
- you can provide links to QR codes instead of displaying them, which is not possible with `qr_from_text`

One disadvantage of `qr_url_from_text` is that it increases the number of requests to the server: one request to serve the page containing the URL and another to request the image.

Be aware that serving image files (which are generated on the fly) from a URL can be abused and lead to (D)DoS attack pretty easily, for instance by requesting very large QR codes from outside your application. This is the reason why the associated setting `ALLOWS_EXTERNAL_REQUESTS_FOR_REGISTERED_USER` in

`QR_CODE_URL_PROTECTION` defaults to completely forbid external access to the API. Be careful when opening external access.

1.3.3 QR Codes Caching

Caching QR codes reduces CPU usage, but the usage of `qr_url_from_text` (which caching depends on) increases the number of requests to the server (one request to serve the page containing the URL and another to request the image).

Moreover, be aware that the largest QR codes, in version 40 with a border of 4 modules and rendered in SVG format, have a size of ~800 KB. Be sure that your cache options are reasonable and can be supported by your server(s), especially for in-memory caching.

Note that even without caching the generated QR codes, the app will return a *HTTP 304 Not Modified* status code whenever the same QR code is requested again. The URL named **`qr_code:serve_qr_code_image`** adds the `ETag` and `Last-Modified` headers to the response if the headers aren't already set, enabling *HTTP 304 Not Modified* response upon following requests.

1.4 Demo Application

If you want to try this app, you may want to use the demo application shipped alongside the source code.

Get the source code from GitHub, follow the *installation instructions* above, and run the `runserver` command of Django:

```
python manage.py runserver
```

The demo application should be running at http://127.0.0.1:8000/qr-code-demo/.

If you have Docker Compose installed, you can simply run the following from a terminal (this will save you the burden of setting up a proper python environment):

```
cd scripts
./run-demo-app.sh
```

The demo application should be running at http://127.0.0.1:8910/qr-code-demo/.

1.5 Testing

Get the source code from GitHub, follow the *installation instructions* above, and run the `test` command of Django:

```
python manage.py test
```

This will run the test suite with the locally installed version of Python and Django.

If you have Docker Compose installed, you can simply run the following from a terminal (this will save you the burden of setting up a proper python environment):

```
cd scripts
./run-tests.sh
```

This will run the test suite with all supported versions of Python and Django. The test results are stored within `tests_result` folder.

1.6 Projects Using this App

This app is used in the following projects:

- MyGym Web: a web platform for managing sports clubs. The QR codes are used for importing members' contact information in a phone book.
- Gymna-Score: a web platform for entering scores during gymnastics competitions organized by the Association Cantonale Jurassienne de Gymnastique (ACJG). The QR codes are used to provide an easy way for the public to follow an ongoing competition. They are also used to authenticate judges that need to enter scores.
- AC-Ju: a website that generates digital vouchers that can be redeemed at affiliate merchants.

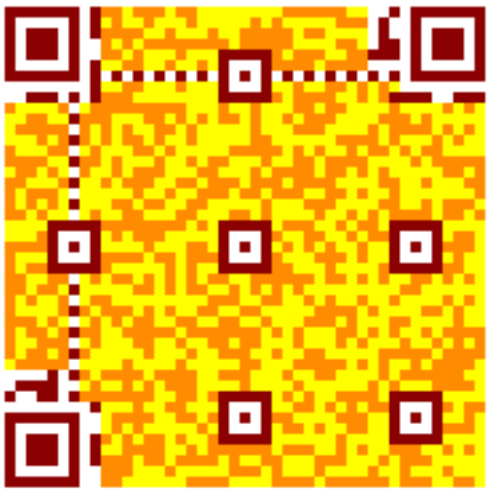
CHAPTER 2

Module Colors

By default the QR codes are rendered in black and white but this project supports individual colors for each QR Code module type, i.e. the alignment and finder patterns may use another color than the data modules.

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').

The value `None` is used to indicate transparency, i.e. `light_color=None` indicates that all light modules should be transparent.



2.1 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.

2.1.1 dark_color

Sets the (default) color of dark modules.

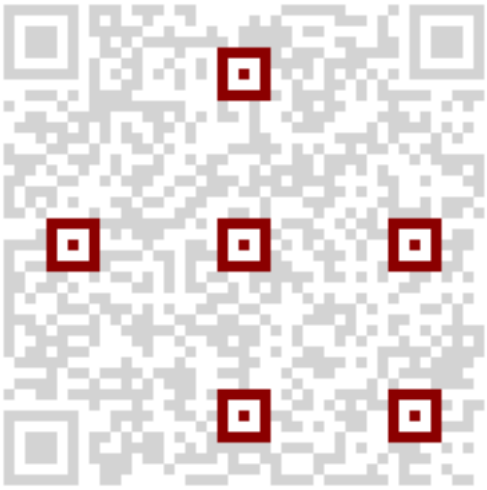
2.1.2 light_color

Sets the (default) color of light modules.

2.1.3 alignment_dark_color

Sets the color of the dark alignment pattern modules.

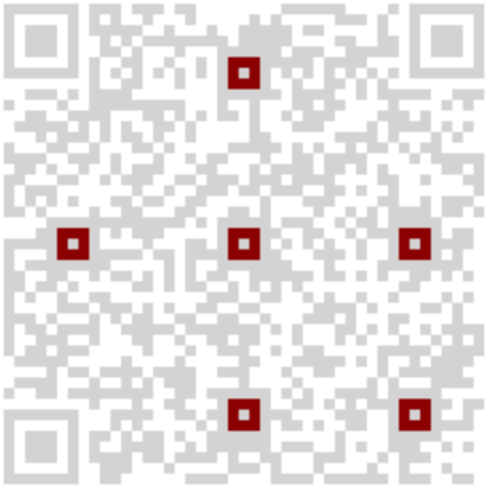
Micro QR Codes don't have alignment patterns.



2.1.4 alignment_light_color

Sets the color of the light alignment pattern modules.

Micro QR Codes don't have alignment patterns.



2.1.5 dark_module_color

Sets the color of the dark module.

Micro QR Codes don't have a dark module.

2.1.6 data_dark_color

Sets the color of the dark data modules.

2.1.7 data_light_color

Sets the color of the light data modules.

2.1.8 finder_dark_color

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

2.1.9 finder_light_color

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

2.1.10 format_dark_color

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

2.1.11 format_light_color

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

2.1.12 quiet_zone_color

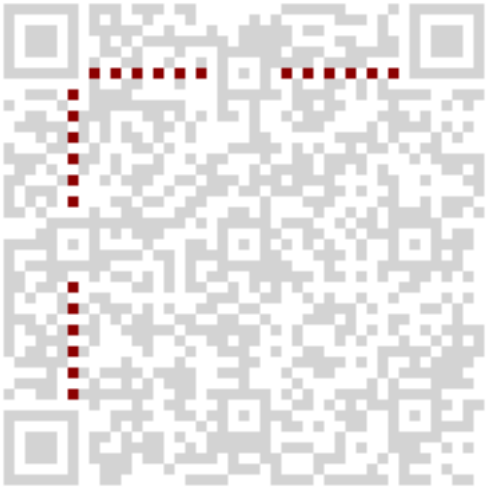
Sets the color of the quiet zone.

2.1.13 separator_color

Sets the color of the separator.

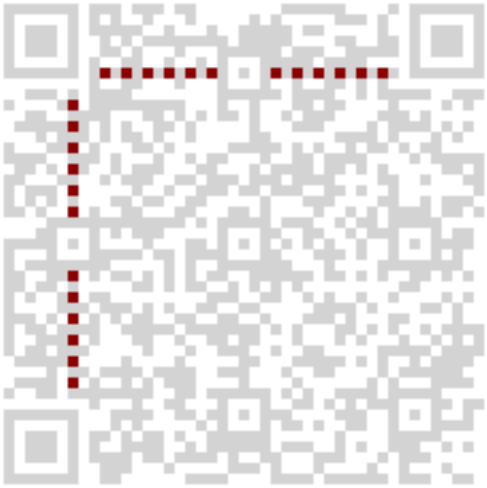
2.1.14 timing_dark_color

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



2.1.15 timing_light_color

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



2.1.16 version_dark_color

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.

2.1.17 version_light_color

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 3

API Reference

3.1 qrcode.maker

Tools for generating QR codes. This module depends on the Segno library.

qr_code.qrcode.maker.**make_embedded_qr_code**(*data: Any*, *qr_code_options: qr_code.qrcode.utils.QRCodeOptions*, *force_text: bool = True*) → str

Generates a <svg> or <img> tag representing the QR code for the given *data*. This tag can be embedded into an HTML document.

qr_code.qrcode.maker.**make_qr**(*data: Any*, *qr_code_options: qr_code.qrcode.utils.QRCodeOptions*, *force_text: bool = True*)

Creates a QR code that encodes the given *data* with the given *qr_code_options*.

Parameters

- **data** (*str*) – The data to encode
- **qr_code_options** – Options to create and serialize the QR code.
- **force_text** (*bool*) – Tells whether we want to force the *data* to be considered as text string and encoded in byte mode.

Return type segno.QRCode

qr_code.qrcode.maker.**make_qr_code_image**(*data: Any*, *qr_code_options: qr_code.qrcode.utils.QRCodeOptions*, *force_text: bool = True*) → bytes

Creates a bytes object representing a QR code image for the provided *data*.

Parameters

- **data** (*str*) – The data to encode
- **qr_code_options** – Options to create and serialize the QR code.
- **force_text** (*bool*) – Tells whether we want to force the *data* to be considered as text string and encoded in byte mode.

Return type bytes

3.2 qrcode.serve

qr_code.qrcode.serve.**get_qr_url_protection_signed_token**(*qr_code_options: qr_code.qrcode.utils.QRCodeOptions*)

Generate a signed token to handle view protection.

qr_code.qrcode.serve.**get_qr_url_protection_token**(*qr_code_options*, *random_token*)

Generate a random token for the QR code image.

The token contains image attributes so that a user cannot use a token provided somewhere on a website to generate bigger QR codes. The random_token part ensures that the signed token is not predictable.

qr_code.qrcode.serve.**make_qr_code_url**(*data: Any*, *qr_code_options: Optional[qr_code.qrcode.utils.QRCodeOptions] = None*, *force_text: bool = True*, *cache_enabled: Optional[bool] = None*, *url_signature_enabled: Optional[bool] = None*) → str

Build an URL to a view that handle serving QR code image from the given parameters.

Any invalid argument related to the size or the format of the image is silently converted into the default value for that argument.

Parameters

- **data** (*str*) – Data to encode into a QR code.
- **qr_code_options** (QRCodeOptions) – The rendering options for the QR code.
- **force_text** (*bool*) – Tells whether we want to force the *data* to be considered as text string and encoded in byte mode.
- **cache_enabled** (*bool*) – Allows to skip caching the QR code (when set to *False*) when caching has been enabled.
- **url_signature_enabled** (*bool*) – Tells whether the random token for protecting the URL against external requests is added to the returned URL. It defaults to *True*.

3.3 qrcode.utils

Utility classes and functions for configuring and setting up the content and the look of a QR code.

class qr_code.qrcode.utils.**ContactDetail**(*first_name: Optional[str] = None*, *last_name: Optional[str] = None*, *first_name_reading: Optional[str] = None*, *last_name_reading: Optional[str] = None*, *tel: Optional[str] = None*, *tel_av: Optional[str] = None*, *email: Optional[str] = None*, *memo: Optional[str] = None*, *birthday: Optional[datetime.date] = None*, *address: Optional[str] = None*, *url: Optional[str] = None*, *nickname: Optional[str] = None*, *org: Optional[str] = None*)

Represents the detail of a contact for MeCARD encoding.

Note: This is a legacy class. Please use `MeCard` instead for new projects.

Fields meaning:

- first_name
- last_name
- first_name_reading: the sound of the first name.
- last_name_reading: the sound of the last name.
- tel: the phone number, it can appear multiple times.
- tel_av: the video-phone number, it can appear multiple times.
- email: the email address, it can appear multiple times.
- memo: notes.
- birthday: the birth date (Python date).
- address: the fields divided by commas (,) denote PO box, room number, house number, city, prefecture, zip code and country, in order.
- url: homepage URL.
- nickname: display name.
- org: organization or company name (non-standard,but often recognized, ORG field).

__init__(*first_name: Optional[str] = None*, *last_name: Optional[str] = None*, *first_name_reading: Optional[str] = None*, *last_name_reading: Optional[str] = None*, *tel: Optional[str] = None*, *tel_av: Optional[str] = None*, *email: Optional[str] = None*, *memo: Optional[str] = None*, *birthday: Optional[datetime.date] = None*, *address: Optional[str] = None*, *url: Optional[str] = None*, *nickname: Optional[str] = None*, *org: Optional[str] = None*)
Initialize self. See help(type(self)) for accurate signature.

__weakref__
list of weak references to the object (if defined)

make_qr_code_data() → str
Make a text for configuring a contact in a phone book. The MeCARD format is used, with an optional, non-standard (but often recognized) ORG field.

See this archive of the format specifications: https://web.archive.org/web/20160304025131/https://www.nttdocomo.co.jp/english/service/developer/make/content/barcode/function/application/addressbook/index.html

Returns the MeCARD contact description.

class qr_code.qrcode.utils.**Coordinates**(*latitude: float*, *longitude: float*, *altitude: Optional[float] = None*)
Represents a set of coordinates with an optional altitude.

Fields meaning:

- latitude: The latitude.
- latitude: The longitude.
- latitude: The optional altitude.

__str__() → str
Return str(self).

__weakref__
list of weak references to the object (if defined)

class qr_code.qrcode.utils.**Email**(*to: Union[str, Sequence[str]], cc: Union[str, Sequence[str], None] = None, bcc: Union[str, Sequence[str], None] = None, subject: Optional[str] = None, body: Optional[str] = None*)
Represents the data of an e-mail.

Fields meaning:

- to: The email address (recipient). Multiple values are allowed.
- cc: The carbon copy recipient. Multiple values are allowed.
- bcc: The blind carbon copy recipient. Multiple values are allowed.
- subject: The subject.
- body: The message body.

__weakref__
list of weak references to the object (if defined)

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

Return type str

class qr_code.qrcode.utils.**EpcData**(*name: str, iban: str, amount: Union[int, float, decimal.Decimal], text: Optional[str] = None, reference: Optional[str] = None, bic: Optional[str] = None, purpose: Optional[str] = None*)
Data for representing an European Payments Council Quick Response Code (EPC QR Code) version 002.

You must always use 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).

Fields meaning:

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

__weakref__
list of weak references to the object (if defined)

make_qr_code_data() → str
Validates the input and creates the data for an European Payments Council Quick Response Code (EPC QR Code) version 002.

This is a wrapper for `segno.helpers._make_epc_qr_data()` with no choice for encoding.

Return type str

class qr_code.qrcode.utils.**MeCard**(*name: str, reading: Optional[str] = None, email: Union[str, Sequence[str], None] = None, phone: Union[str, Sequence[str], None] = None, videophone: Union[str, Sequence[str], None] = None, memo: Optional[str] = None, nickname: Optional[str] = None, birthday: Union[str, datetime.date, None] = None, url: Union[str, Sequence[str], None] = None, pobox: Optional[str] = None, roomno: Optional[str] = None, houseno: Optional[str] = None, city: Optional[str] = None, prefecture: Optional[str] = None, zipcode: Union[int, str, None] = None, country: Optional[str] = None, org: Optional[str] = None*)
Represents the detail of a contact for MeCARD encoding.

Fields meaning:

- name: Name. If it contains a comma, the first part is treated as lastname and the second part is treated as forename.
- reading: Designates a text string to be set as the kana name in the phonebook
- email: E-mail address. Multiple values are allowed.
- phone: Phone number. Multiple values are allowed.
- videophone: Phone number for video calls. Multiple values are allowed.
- memo: A notice for the contact.
- nickname: Nickname.
- birthday: Birthday. If a string is provided, it should encode the date as YYYYMMDD value.
- url: Homepage. Multiple values are allowed.
- pobox: P.O. box (address information).
- roomno: Room number (address information).
- houseno: House number (address information).
- city: City (address information).
- prefecture: Prefecture (address information).
- zipcode: Zip code (address information).
- country: Country (address information).
- org: organization or company name (ORG field, non-standard,but often recognized by readers).

__weakref__
list of weak references to the object (if defined)

make_qr_code_data() → str
Creates a string encoding the contact information as MeCARD.

Return type str

class qr_code.qrcode.utils.**QRCodeOptions**(*size: Union[int, str] = 'm', border: int = 4, version: Union[int, str, None] = None, image_format: str = 'svg', error_correction: str = 'm', encoding: Optional[str] = 'utf-8', boost_error: bool = True, micro: bool = False, eci: bool = False, dark_color: Union[tuple, str] = '#000', light_color: Union[tuple, str] = '#fff', finder_dark_color: bool = False, finder_light_color: bool = False, data_dark_color: bool = False, data_light_color: bool = False, version_dark_color: bool = False, version_light_color: bool = False, format_dark_color: bool = False, format_light_color: bool = False, alignment_dark_color: bool = False, alignment_light_color: bool = False, timing_dark_color: bool = False, timing_light_color: bool = False, separator_color: bool = False, dark_module_color: bool = False, quiet_zone_color: bool = False*)

Represents the options used to create and draw a QR code.

__init__(*size: Union[int, str] = 'm', border: int = 4, version: Union[int, str, None] = None, image_format: str = 'svg', error_correction: str = 'm', encoding: Optional[str] = 'utf-8', boost_error: bool = True, micro: bool = False, eci: bool = False, dark_color: Union[tuple, str] = '#000', light_color: Union[tuple, str] = '#fff', finder_dark_color: bool = False, finder_light_color: bool = False, data_dark_color: bool = False, data_light_color: bool = False, version_dark_color: bool = False, version_light_color: bool = False, format_dark_color: bool = False, format_light_color: bool = False, alignment_dark_color: bool = False, alignment_light_color: bool = False, timing_dark_color: bool = False, timing_light_color: bool = False, separator_color: bool = False, dark_module_color: bool = False, quiet_zone_color: bool = False*) → None

Parameters

- **size** – The size of the QR code as an integer or a string. Default is *'m'*.
- **border** (*int*) – The size of the border (blank space around the code).
- **version** (*int, str or None*) – The version of the QR code gives the size of the matrix. Default is *None* which mean automatic in order to avoid data overflow.
- **version** – 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.
- **image_format** (*str*) – The graphics format used to render the QR code. It can be either *'svg'* or *'png'*. Default is *'svg'*.
- **error_correction** (*str*) – How much error correction that might be required to read the code. It can be either *'L'*, *'M'*, *'Q'*, or *'H'*. Default is *'M'*.
- **boost_error** (*bool*) – Tells whether the QR code encoding engine tries to increase the error correction level if it does not affect the version. Error correction level is not increased when it impacts the version of the code.
- **micro** (*bool*) – Indicates if a Micro QR Code should be created. Default: False

- **encoding** (*str or None*) – Indicates the encoding in mode "byte". By default *encoding* is `UTF-8`. When set to `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 **:paramref:'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.
- **dark_color** – Color of the dark modules (default: black). The color can be provided as `(R, G, B)` tuple, as hexadecimal format (`#RGB`, `#RRGGBB` `RRGGBBAA`), or web color name (i.e. `red`).
- **light_color** – Color of the light modules (default: white). See *color* for valid values. If light is set to `None` the light modules will be transparent.
- **finder_dark_color** – Color of the dark finder modules (default: same as `dark_color`)
- **finder_light_color** – Color of the light finder modules (default: same as `light_color`)
- **data_dark_color** – Color of the dark data modules (default: same as `dark_color`)
- **data_light_color** – Color of the light data modules (default: same as `light_color`)
- **version_dark_color** – Color of the dark version modules (default: same as `dark_color`)
- **version_light_color** – Color of the light version modules (default: same as `light_color`)
- **format_dark_color** – Color of the dark format modules (default: same as `dark_color`)
- **format_light_color** – Color of the light format modules (default: same as `light_color`)
- **alignment_dark_color** – Color of the dark alignment modules (default: same as `dark_color`)
- **alignment_light_color** – Color of the light alignment modules (default: same as `light_color`)
- **timing_dark_color** – Color of the dark timing pattern modules (default: same as `dark_color`)
- **timing_light_color** – Color of the light timing pattern modules (default: same as `light_color`)
- **separator_color** – Color of the separator (default: same as `light_color`)
- **dark_module_color** – Color of the dark module (default: same as `dark_color`)
- **quiet_zone_color** – Color of the quiet zone modules (default: same as `light_color`)

Type str or int

The *size* parameter gives the size of each module of the QR code matrix. It can be either a positive integer or one of t

- t or T: tiny (value: 6)
- s or S: small (value: 12)
- m or M: medium (value: 18)
- l or L: large (value: 30)
- h or H: huge (value: 48)

For PNG image format the size unit is in pixels, while the unit is 0.1 mm for SVG format.

The *border* parameter controls how many modules thick the border should be (blank space around the code). The default is 4, which is the minimum according to the specs.

The *version* parameter is an integer from 1 to 40 that controls the size of the QR code matrix. Set to None to determine this automatically. The smallest, version 1, is a 21 x 21 matrix. The biggest, version 40, is 177 x 177 matrix. The size grows by 4 modules/side. For Micro QR codes, valid values are "M1", "M2", "M3", "M4".

There are 4 error correction levels used for QR codes, with each one adding different amounts of "backup" data depending on how much damage the QR code is expected to suffer in its intended environment, and hence how much error correction may be required. The correction level can be configured with the *error_correction* parameter as follow:

- l or L: error correction level L – up to 7% damage
- m or M: error correction level M – up to 15% damage
- q or Q: error correction level Q – up to 25% damage
- h or H: error correction level H – up to 30% damage

You may enforce the creation of a Micro QR Code with *micro=True*. The *micro* option defaults to *False*.

The *encoding* option controls the text encoding used in mode "byte" (used for any general text content). By default *encoding* is `UTF-8`. When set to `None`, the implementation (based on Segno) 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* option). The *encoding* parameter is case-insensitive.

The *boost_error* indicates whether the QR code encoding engine (Segno) tries to increase the error correction level if it does not affect the version. Error correction level is not increased when it impacts the version of the code.

The *eci* option 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.

Raises TypeError in case an unknown argument is given.

`__weakref__`
list of weak references to the object (if defined)

`color_mapping``()`
Internal method which returns the color mapping.

Only non-default values are returned.

Return type dict

kw_make()
Internal method which returns a dict of parameters to create a QR code.

Return type dict

kw_save()
Internal method which returns a dict of parameters to save a QR code.

Return type dict

class qr_code.qrcode.utils.**VCard**(*name: str, displayname: Optional[str] = None, email: Union[str, Sequence[str], None] = None, phone: Union[str, Sequence[str], None] = None, fax: Union[str, Sequence[str], None] = None, videophone: Union[str, Sequence[str], None] = None, memo: Optional[str] = None, nickname: Optional[str] = None, birthday: Union[str, datetime.date, None] = None, url: Union[str, Sequence[str], None] = None, pobox: Optional[str] = None, street: Optional[str] = None, city: Optional[str] = None, region: Optional[str] = None, zipcode: Union[int, str, None] = None, country: Optional[str] = None, org: Optional[str] = None, lat: Optional[float] = None, lng: Optional[float] = None, source: Optional[str] = None, rev: Union[str, datetime.date, None] = None, title: Union[str, Sequence[str], None] = None, photo_uri: Union[str, Sequence[str], None] = None*)
Represents the detail of a contact for vCard encoding.

Only a subset of available *vCard 3.0 properties <https://tools.ietf.org/html/rfc2426>*

Fields meaning:

- name: The name. If it contains a semicolon, the first part is treated as lastname and the second part is treated as forename.
- displayname: Common name. Defaults to *name* without the semicolon if `None`.
- email: E-mail address. Multiple values are allowed.
- phone: Phone number. Multiple values are allowed.
- fax: Fax number. Multiple values are allowed.
- videophone: Phone number for video calls. Multiple values are allowed.
- memo: A notice for the contact.
- nickname: Nickname.
- birthday: Birthday. If a string is provided, it should encode the date as `YYYY-MM-DD` value.
- url: Homepage. Multiple values are allowed.
- pobox: P.O. box (address information).
- street: Street address.
- city: City (address information).
- region: Region (address information).
- zipcode: Zip code (address information).

- country: Country (address information).
- org: Company / organization name.
- lat: Latitude.
- lng: Longitude.
- source: URL where to obtain the vCard.
- rev: Revision of the vCard / last modification date.
- title: Job Title. Multiple values are allowed.
- photo_uri: Photo URI. Multiple values are allowed.

__weakref__
list of weak references to the object (if defined)

make_qr_code_data() → str
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.

Return type str

class qr_code.qrcode.utils.**WifiConfig**(*ssid: str = ''*, *authentication: int = 0*, *password: str = ''*, *hidden: bool = False*)
Represents a WIFI configuration.

Fields meaning:

- ssid: the name of the SSID
- authentication: the authentication type for the SSID; can be AUTHENTICATION.wep or AUTHENTICATION.wpa, or AUTHENTICATION.nopass for no password. Or, omit for no password.
- password: the password, ignored if "authentication" is 'nopass' (in which case it may be omitted).
- hidden: tells whether the SSID is hidden or not; can be True or False.

__init__(*ssid: str = ''*, *authentication: int = 0*, *password: str = ''*, *hidden: bool = False*) → None
Initialize self. See help(type(self)) for accurate signature.

__weakref__
list of weak references to the object (if defined)

make_qr_code_data() → str
Make a text for configuring a Wi-Fi connexion. The syntax is inspired by the MeCARD format used for contacts.

Returns the WIFI configuration text that can be translated to a QR code.

Return type str

3.4 qrcode.views

qr_code.views.**cache_qr_code**()
Decorator that caches the requested page if a settings named 'QR_CODE_CACHE_ALIAS' exists and is not empty or None.

qr_code.views.**check_image_access_permission**(*request*, *qr_code_options*) → None
Handle image access protection (we do not allow external requests for anyone).

qr_code.views.**serve_qr_code_image**(*request*) → django.http.response.HttpResponse
Serve an image that represents the requested QR code.

IMPORTANT NOTE: Each boolean option mentioned below (value True/False) must be passed as *<option name>=1* for *True*, respectively *<option name>=0* for *False*.

You may pass any argument available for *qr_code.qrcode.utils.QRCodeOptions.__init__()* to adjust the appearance of the returned QR code. The arguments must be passed as query string arguments in the URL. Additionally, the following arguments are available:

- cache_enabled: boolean – Tells the generator to work around the caching mechanism if it is empty (default: True/undefined).
- **token: str – By default, the application only serves QR code images for authenticated URLs (requests generated fron**
  The authentication uses a HMAC to sign the request query arguments. The authentication code is passed as a query argument named *token* which is automatically generated by *qr_url_from_text* or *qr_url_from_data*.

3.5 templatetags.qr_code

Tags for Django template system that help generating QR codes.

qr_code.templatetags.qr_code.**qr_for_geolocation**(***kwargs*) → str
Accepts a *'coordinates'* keyword argument or a triplet *'latitude'*, *'longitude'*, and *'altitude'*.

qr_code.templatetags.qr_code.**qr_for_google_maps**(***kwargs*) → str
Accepts a *'coordinates'* keyword argument or a pair *'latitude'* and *'longitude'*.

qr_code.templatetags.qr_code.**qr_url_for_geolocation**(***kwargs*) → str
Accepts a *'coordinates'* keyword argument or a triplet *'latitude'*, *'longitude'*, and *'altitude'*.

qr_code.templatetags.qr_code.**qr_url_for_google_maps**(***kwargs*) → str
Accepts a *'coordinates'* keyword argument or a pair *'latitude'* and *'longitude'*.

3.6 tests

CHAPTER 4

Change Log

4.1 3.0.0 (2021-11-27)

- Add support for European Payments Council Quick Response Code (EPC QR Code) version 002.
- Add support for vCard v3 QR code.
- Revamp support for MeCARD QR code to provide a cleaner API (old API remains available for compatibility).
- Introduce `qr_from_data` and `qr_url_from_data` to allow optimized encoding of alphanumeric, numeric, and byte data (adopt appropriate encoding mode depending on data content).
- Introduce support for `boost_error` flag.
- Introduce support for `encoding` parameter.
- Several breaking changes in API and generated QR codes:
  - `text` parameters renamed to `data`;
  - class methods `make_qr_text` renamed to `make_qr_data`;
  - uses UTF-8 encoding by default for application and text QR codes (you must set the new `encoding` option or use the new `qr_from_data` and `qr_url_from_data` template tags to emulate old behavior);
  - encoded geolocations always contain a decimal separator.
- Improve API documentation.
- Add support for Python 3.10
- Drop support for Python 3.6.
- Drop support for Django 3.1.

4.2 2.3.0 (2021-11-07)

- Add support for `ECI mode` to control bytes encoding.

- Fix handling of `micro` QR code option when using URLs to serve image via `{% qr_url_from_text %}`.
- Fix handling of `cache_enabled` option when using URLs to serve image via `{% qr_url_from_text %}`.
- Fix handling of `url_signature_enabled` option when using URLs to serve image via `{% qr_url_from_text %}`.

4.3 2.2.0 (2021-06-03)

- Change encoding from URL-safe Base64 to standard Base64 for `text` query argument (used for serving QR code images).
- Fix #31 by passing the border parameter for segno.QRCode.save.
- Ensure compatibility with Django 3.2.
- Drop support for Django 3.0.

4.4 2.1.0 (2021-01-23)

- Change encoding from URL-safe Base64 to standard Base64 for `text` query argument (used for serving QR code images).
- Introduce setting `SERVE_QR_CODE_IMAGE_PATH` to configure the path under which QR Code images are served.
- Reorganize and improve documentation.
- Fix #23
- Introduce usage of type hints.

4.5 2.0.1 (2020-11-24)

- Update the install_requires after the move from qrcode to Segno.

4.6 2.0.0 (2020-11-22)

- Remove dependency on Pillow / qrcode
- Switch to Segno for generating QR Codes
- Add support for QR Codes with multiple colors
- Add support for Micro QR Codes
- Stable SVG format for QR code between 32-bit and 64-bit architecture (#19)
- Use hyphens in URLs (#16)
- Add support for Python 3.9

4.7 1.3.1 (2020-09-07)

- Fix local testing script.
- Fix date of release 1.3.0 in readme.
- Code cleanup.

4.8 1.3.0 (2020-09-05)

- Drop support for Django 2.1.
- Ensure compatibility with Django 3.1.

4.9 1.2.0 (2020-04-26)

- Ensure compatibility with Django 3.0.
- Upgrade Pillow requirement to 7.1.
- Drop support for Python 3.5.
- Drop support for Django <2.2.
- More modern build environment and configuration for ReadTheDocs.

4.10 1.1.0 (2019-11-16)

- Ensure compatibility with Django 2.1.
- Ensure compatibility with Django 2.2.
- Upgrade qr_code library from 5.3 to 6.1 (several fixes).
- Drop support for Python 3.4.
- Fixed error when generating qr code from lazy text. (#1)
- Add support for customizing usage of URL signature token via template tags (allows to generate URLs for serving QR code images that do not include a signature token). (#4)
- The caching of QR codes images could allow to bypass checking the user when external access is active.
- Upgrade Pillow requirement to 6.2.0 (CVE-2019-16865).
- Adopt a dedicated logger, and move message "Pillow is not installed. No support available for PNG format." from info to debug. (#6)

4.11 1.0.0 (2018-03-23)

- BREAKING CHANGES:
  - QR code options have been factorized and now use the `QRCodeOptions` class.

  - The context for rendering a QR code encoding a Wi-Fi configuration uses the dedicated `WifiConfig` class.
  - The context for rendering a QR code encoding a contact detail uses the dedicated `ContactDetail` class.
  - `qr_for_contact` and `qr_url_for_contact` keyword arg has been renamed from `contact_dict` to `contact_detail`.
  - `qr_for_wifi` and `qr_url_for_wifi` keyword arg has been renamed from `wifi_dict` to `wifi_config`.
  - Reorganize code and split qr_code.py into several modules.

The changes mentioned above might break the compatibility with code using qr_code.py's API directly, but template tags are not impacted, except for `qr_for_contact`, `qr_url_for_contact`, `qr_for_wifi`, and `qr_url_for_wifi` if they were using a keyword argument.

- Other changes:
  - Added support for `error_correction` parameter when generating a QR code.
  - Added support for `coordinates` keyword argument to `qr_for_geolocation`, `qr_for_google_maps`, `qr_url_for_geolocation`, and `qr_url_for_google_maps`.
  - Additions to documentation.
  - Added ability to use a `QRCodeOptions` instance with `options` keyword argument in all tags.
- Bug fixes:
  - Fixed non-closed tag when generating embedded PNG image.
  - Escape colon char (':') if it appears within a contact detail or a wifi configuration.
  - Add a second terminal semi-colon at the end of the text representing a wifi configuration, as recommended in some sources.

4.12 0.4.1 (2018-03-10)

- Fixed unescaped chars when generating QR code for a contact.
- Simplify handling of default values for QR code options.
- Add documentation about what a QR code can encode.

4.13 0.4.0 (2018-03-09)

- Added support for multiple new tags:
  - `qr_for_email` and `qr_url_for_email`
  - `qr_for_tel` and `qr_url_for_tel`
  - `qr_for_sms` and `qr_url_for_sms`
  - `qr_for_geolocation` and `qr_url_for_geolocation`
  - `qr_for_google_maps` and `qr_url_for_google_maps`
  - `qr_for_youtube` and `qr_url_for_youtube`
  - `qr_for_google_play` and `qr_url_for_google_play`

  - `qr_for_contact` and `qr_url_for_contact`
  - `qr_for_wifi` and `qr_url_for_wifi`
- Reformat documentation on the demo site for better readability.
- Drop support for Django <1.11.

4.14 0.3.3 (2017-08-16)

- Added `app_name` namespace to `qr_code.urls` (better compatibility with `include()` function provided with Django >= 1.9).
- Update documentation regarding the inclusion of `qr_code.urls` for distinct versions of Django.
- Minor improvements to the documentation.

4.15 0.3.2 (2017-08-13)

- Allows optional installation of Pillow (PNG format unavailable, fallback to SVG).
- Fixed caching of images (not working due protection against external queries).
- Fixed conditional view processing (HTTP 304) for rendered QR codes (not working due protection against external queries).

4.16 0.3.1 (2017-08-12)

- Added a mention about Pillow library requirement in documentation.
- Minor improvements to the documentation and the demo application.

4.17 0.3.0 (2017-08-12)

- Added new tag qr_url_from_text:
  - Separate image from the page displaying the image
  - Handle caching of images
  - Conditional view processing (HTTP 304) for rendered QR codes
  - Protection against external requests
  - Settings to configure URLs accesses as a service for generating QR code images
  - Add documentation for new features
  - Add tests for new features
  - Add examples to demo site
- More robust testing of make_embedded_qr_code's arguments.
- Improved documentation.
- Demo site is compatible with Django 1.8.

- Added support for Docker Compose for running the demo application and running the tests.

4.18 0.2.1 (2017-08-05)

- Added support for Django 1.8.
- Fixed version specifiers for Django requirement so that it wont force the installation of Django 1.11.
- Added badges for PyPi, Read the Docs and Travis CI to readme file.
- Several additions to the documentation.

4.19 0.2.0 (2017-08-04)

- Add support for PNG image format via an `img` tag.
- Add documentation for users and developers.
- Improve examples in demo app.

4.20 0.1.1 (2017-08-02)

First public release.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

q

Index

Symbols

C

E

G

K

M

Q

S

V

W