

The fancyqr package



Florian Sihler

Version v2.4 – 2026/08/17

LPPL 1.3c or later

github.com/EagleoutIce/fancyqr

fancyqr is a simple package to create fancy qr-codes with the help of the *qrcode*-package. You can use the `\fancyqr`-macro as a drop-in replacement for the normal `\qrcode`.¹

If you do want to hide a center square (e.g. because you want to embed an image) you can use `\FancyQrDoNotPrintSquare{<x>}{<y>}` to hide a rectangle with radius x and y set from the center (with `\FancyQrDoNotPrintRadius{<factor>}` you can apply a *rounding* to this!). If you choose this option, the default `\FancyQrRoundCut` that rounds cut corners can be changed with `\FancyQrHardCut`. The styles that draw randomized tiles use the random number generator of $\LaTeX$; `\FancyQrSeed{<n>}` (or the seed option) fixes its seed and makes them reproducible.

All of the extra qr-options (you can set all of them with `\fancyqrset{<keys>}`) are showcased in Table 1. The defaults are set like this:

```
\fancyqrset{image padding=0,gradient=true,
            gradient angle=135,
            r color=red!68!black!88!white!90!white,
            l color=purple!40!red!20!black!90!white}
```

Consider the following examples (uses *fontawesome*, but you can use include images, ...):

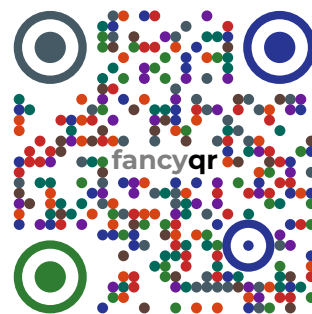
```
\fancyqr[
  classic,image=\LARGE\faGithub,
]{https://github.com/EagleoutIce/fancyqr}
```



```
\FancyQrLoad{goo}%
\fancyqr[
  seed=3,content color=black
]{https://github.com/EagleoutIce/fancyqr}%
```



```
% needs \usepackage[prefix=@]{xcolor-material}
\FancyQrLoad{dots}%
\fancyqr[
  height=4cm,
  image={\textbf{\textcolor{gray}{fancy}qr}},
  random color={@Red800}{@Purple800}{@Indigo800}{@Teal800}{@Green800}
             {@DeepOrange800}{@Brown700}{@BlueGrey700}
]{https://github.com/EagleoutIce/fancyqr}%
```



¹`\fancyqr[<qr-options>]{<url>}`

1 Styles

Besides the `default` style there are eight more, all loaded (locally) with `\FancyQrLoad{<name>}`; `\FancyQrLoad{default}` and `\FancyQrLoadDefault` reset to the default. Figure 1 shows every one of them with the same payload and the call that produced it.



Figure 1: Every style that `fancyqr` ships with. `blobs`, `glitch` and `goo` are randomized and therefore shown with a fixed seed. Every one of them but `frame` is read by a scanner as it is; `frame` leaves the center of a module white, which is where a scanner samples, so it needs `finder=square` to be found at all.

The `default` style rounds the free corners of a tile with the `rounding` radius. A corner that is enclosed by two black neighbours but not by the diagonal one is an *inner* corner; it is softened with a fillet of the (smaller) `inner rounding` radius instead of staying a hard 90 degree edge. Setting `inner rounding=0`, or loading the `rounded` style, brings back the sharp notches of earlier versions:

```
\fancyqr[
  color=black,inner rounding=0
]{https://github.com/EagleoutIce/fancyqr}
```



```
\fancyqr[
  color=black,rounding=.25,inner rounding=.35
]{https://github.com/EagleoutIce/fancyqr}
```



Every style picks the pattern shape that suits it: `dots` and `blobs` a circle, everything else `inherit`, which leaves the patterns to the tiles of the style itself. The `glows` style draws every module once per halo ring and once for the tile itself; `\FancyQrPasses{<n>}` is what a style uses to ask for that. It is tuned with `\FancyQrGlow{<width>}{<strength>}{<falloff>}` – the width of the widest ring in modules, the percentage of color it keeps, and how quickly the rings gain color towards

the tile – and with `\FancyQrGlowColor{<from>}{<to>}`, where `.` is the color of the tile the halo surrounds: `\FancyQrGlowColor{white}{black}` turns it into a white backglow. Keep the width below one module, or the halos of two modules with a white one between them run into each other. `glitch` uses the passes as well, for the two displaced ghosts it puts behind its shards; its `\fancyqr@glitch@detach` is the chance that a shard tears off a neighbour instead of merging with it. `goo` hangs its tiles downwards by `\fancyqr@goo@drip`; `0` makes it symmetric again. `frame` outlines the shape the modules form – only the sides facing white are stroked, so a run of modules becomes one pill – and puts a dot inside every module, because a scanner samples the center of a module and a bare outline would leave it white. `dots` and `blobs` additionally draw the position and alignment patterns as solid shapes, because a scanner first has to find those; with `finder=inherit` they are made of tiles as well and the code no longer reads. A style declares with `\FancyQrInnerRoundings` (or `\FancyQrNoInnerRoundings`) whether its tiles understand inner corners at all; only `default` and `glows` do, so no other style pays for the extra lookups.

2 Position and alignment patterns

The three big position (finder) patterns and all alignment patterns can be drawn as one shape instead of as single modules, and in a color of their own. `finder` picks the shape of their ring, `finder core` the shape of their center block (auto follows the ring), `finder radius` and `finder thickness` its corner radius and its ring width, and `finder color` paints both. Figure 2 shows the combinations together with the keys that produced them. `finder color` paints the patterns and `content color` everything else, so `content color=black` keeps the gradient on the patterns alone. The modules of a pattern stay black in the matrix and only their drawing is replaced, so neighbouring tiles still merge into the shape wherever it allows it. A `finder thickness` other than 1 leaves the 1:1:3:1 run a scanner looks for, so check that the code still reads. `finder=inherit` is the default and leaves the patterns to the tiles of the loaded style, which is what `fancyqr` did before the shapes existed; `none` is accepted as a synonym. Loading a style resets `finder` and `finder core`, because a style owns its shapes.



Figure 2: Styling the position and alignment patterns.

Option	Type	Default	Explanation
cache	boolean	true	Keep the module matrix of a code, so that drawing it again – in a gallery, a running header, ... – only has to redraw it.
classic	boolean	false	Use the classic qr-code style (black with flat rectangles, this loads the flat style).
color	color		Disables the gradient and sets the color accordingly.
compensate	length		Overlap between neighbouring tiles as an absolute length; setting it switches off the module-relative overlap.
finder color	color		Color of the position and alignment patterns. Without finder they keep their normal tiles and are only recolored.
content color	color		Color of everything that is not a position or alignment pattern, so that the patterns can keep the gradient while the data stays flat.
finder	shape	inherit	Draw the three position (finder) patterns and all alignment patterns as one shape (square, rounded, or circle) instead of as single modules; inherit (synonym none) leaves them to the tiles of the style (section 2).
finder core	shape	auto	Shape of the center block of every pattern (auto, none, square, rounded, circle, or star); auto follows finder, which has to be set to a shape for this to take effect.
finder radius	number	0.5	Corner radius of the rounded pattern shape, relative to the pattern radius.
finder thickness	number	1	Width of the pattern ring, in modules.
gradient	boolean	true	Toggle the color gradient
gradient angle	angle	135	Change the gradient angle, in degrees; 0 runs from left to right.
image	L ^A T _E X		Automatically center an image (you have to care for the size and maybe adjust the version and level to keep the qr-code readable). ²
image padding	number		Additionally hide blocks (x & y) around the image.
image x padding	number	0	Additionally hide blocks (x) around the image.
image y padding	number	0	Additionally hide blocks (y) around the image.
inner rounding	number	0.25	Radius of the negative rounding: a corner enclosed by two black neighbours but not by the diagonal one gets a fillet of this size, 0 disables it (section 1).
overlap	number	0.045	Overlap between neighbouring tiles, as a fraction of a module. Two tile edges that cross the same pixel each cover only a part of it, and a renderer then leaves a sliver of the background showing through; what decides whether that is visible is how many device pixels the overlap is, which is why it scales with the module.
l color	color	see above	Color at the left end of the gradient axis, which the default angle puts at the bottom right.
left color level	color L/M/Q/H	M	Alias for l color. qrcode option affecting error correction (low, medium, quartile, high).
padding	flag		qrcode option adding sufficient additional space around the qr-code.
r color	color	see above	Color at the right end of the gradient axis, which the default angle puts at the top left.
random color	colors		Allow to set a random color pool to pick from.
right color	color		Alias for r color.
rounding	number	0.5	Corner radius of the rounded tiles of the default style.
seed	integer		Fix the random seed, so that the randomized styles (blobs, glitch, goo) render reproducibly.
size	length		Alias for qrcode's height option.

3 History

Version v2.4 2026-08-17

<https://github.com/EagleoutIce/fancyqr/releases/tag/v2.4>

- Weakly rounded (negative) inner corners for the `default` style
- New `rounded`, `glows` and `goo` styles
- Style the position and alignment patterns (`finder`, `finder core`, `finder color`, `finder radius`, `finder thickness`) and color them apart from the data with `content color`
- Configurable `rounding` and `inner rounding`, reproducible randomness with `seed`, a less red default gradient
- Roughly five times faster (the module matrix of a code is kept, see `cache`), an `l3build` test suite
- Tile overlap is relative to the module size (`overlap`), which keeps renderers from leaving seams between the tiles
- Bug fixes: `frame` is readable by a scanner again, the center module is no longer dropped without a cut-out, `gradient angle` is read in degrees again, and an embedded image is centered exactly

Version v2.3 2026-02-27

<https://github.com/EagleoutIce/fancyqr/releases/tag/v2.3>

- Relicense under the LPPL 1.3c

Version v2.2 2024-11-27

<https://github.com/EagleoutIce/fancyqr/releases/tag/v2.2>

- Configurable tile overlap
- Circular cut-out support
- New `compensate` option

Version v2.1 2024-10-05

<https://github.com/EagleoutIce/fancyqr/releases/tag/v2.1>

- Add `classic` option to easily typeset classic QR codes
- Simplify picture usage for QR codes
- More documentation and Bug-Fixes

Version v2.0 2024-04-13

<https://github.com/EagleoutIce/fancyqr/releases/tag/v2.0>

- No longer require `TikZ`!
- Random Colors
- Support `width` and `height`

Version v1.0 2022-08-18

<https://github.com/EagleoutIce/fancyqr/releases/tag/v1.0>

- Initial release with basic drop-in functionality
- Using `TikZ` to create the QR code

²The package will automatically calculate the required `\FancyQrDoNotPrintSquare` (you have to make sure that the qr-code still has enough information to be readable). Therefore, the image will not scale with the qr-code.