

fancyqr

github.com/EagleoutIce/fancyqr

Florian Sihler*

Version v2.5 – 2026/10/18

fancyqr draws QR codes with rounded, colored and decorated modules. `qr` computes the code, so `\fancyqr` takes the options of `\qr` and more. It also encodes text beyond ASCII with every engine.

```
\fancyqr{https://ctan.org}
```



```
\FancyQrLoad{snow}  
\fancyqr[seed=3,r_color=lagoon,  
l_color=deepsea]{https://ctan.org}
```



```
\fancyqr[seed=3,height=4cm,style=castle]{https://ctan.org}
```



1	Getting started	2
2	Styles	2
3	Figures and extras	5
4	Finder and alignment patterns	10
5	Images and holes	12
6	Text beyond ASCII	13
7	Faster runs	13
8	Writing a style	13
9	All options	14
10	History	15
	Index of commands, keys and styles	17

*Released under the [LaTeX Project Public License](https://www.latex-project.org/public-license/), version 1.3c or later.

1 Getting started

- `\fancyqr[⟨options⟩]{⟨text⟩}` draws a QR code for the text. It goes wherever `\qr{code}` goes. All options are listed in Section 9.
- `\fancyqrset{⟨options⟩}` sets options for the following codes in the current group.

The package starts with these settings:

```
\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,persist=true]
```

The examples use a few colors of their own:

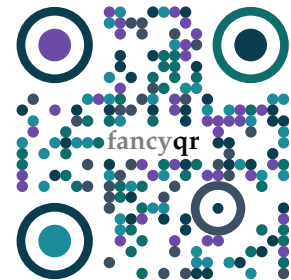
```
\definecolor{ocean}{HTML}{0E6E6B} \definecolor{deepsea}{HTML}{0B3D4F}
\definecolor{lagoon}{HTML}{1C8C9A} \definecolor{slate}{HTML}{3D4F63}
\definecolor{amber}{HTML}{F0A202} \definecolor{plum}{HTML}{6C4AA8}
\definecolor{rose}{HTML}{D8466E}
```

`classic` gives black squares like a plain QR code. Any $\text{\LaTeX}$ content can go in the center (the logo is from `fontawesome`):

```
\fancyqr[classic,
  image=\LARGE\faGithub]{https://ctan.org}
```



```
\FancyQrLoad{dots}
\fancyqr[height=3.6cm,
  image={\textbf{\textcolor{gray}{fancy}qr}},
  random_color={ocean}{deepsea}
    {lagoon}{slate}{plum},
  ]{https://ctan.org}
```



2 Styles

A style decides how the modules are drawn. Figure 1 shows all thirteen.

- `\FancyQrLoad{⟨name⟩}` loads a style for the rest of the current group.
- `style=⟨name⟩` v2.5 loads a style for this code alone. The other options of the code still override it.
- `add={⟨name⟩},...` v2.5 adds the decorations of other styles to the loaded one: `halloween` lends its webs, threads and slime, `christmas` its lights, stars, snowflakes and candles, and `snow` its snow on the tiles. Their tiles, figures and colors stay with them. The added decorations count as extras (Section 3), so `extras` and `no extras` pick from them as well.
- `\FancyQrAdd{⟨name⟩},...` does the same for the rest of the current group, up to the next `\FancyQrLoad`.
- `\FancyQrLoadDefault` goes back to `default`, as does `\FancyQrLoad{default}`.
- `\FancyQrSeed{⟨number⟩}` fixes the random numbers, like `seed`. `blobs`, `glitch`, `goo`, `halloween` and `snow` are random and change on every run without it.

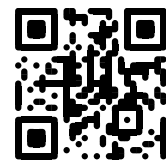


Figure 1: The styles, all with `seed=4`.

2.1 Rounding

- **rounding**= $\langle number \rangle$ (default 0.5) the corner radius of a tile in the **default** style, in modules.
- **inner rounding**= $\langle number \rangle$ (default 0.25) the radius of an inner corner, where two black neighbours meet at a white diagonal. 0 makes them sharp, like the **rounded** style.

```
\fancyqr[color=black,
  inner rounding=0]{https://ctan.org}
```



```
\fancyqr[color=black,rounding=.25,
  inner rounding=.35]{https://ctan.org}
```



2.2 Settings of the styles

Most of these are internal macros: set them between `\makeatletter` and `\makeatother`, before or after loading the style.

- `glows` `\FancyQrGlow{<width>}{<strength>}{<falloff>}` sets the width of the glow in modules (keep it below 1), how much color it keeps in percent, and how fast it fades. `\FancyQrGlowColor{<from>}{<to>}` sets its colors. `.` is the color of the tile.
- `glitch` `\fancyqr@glitch@detach` is the chance that a shard breaks away from its neighbour (.3).
- `goo` `\fancyqr@goo@drip` is how far the tiles drip (.45). `0` stops the dripping.
- `halloween` v2.5 Figures take the place of blocks of black modules that fit them: reapers, jack-o'-lanterns, ghosts, gravestones and crosses, witch hats, spiders, bats, skulls, bones, candies, and small figures such as eyeballs, candles and potions on single modules. Most of them come in variants, cracks, poses and faces differ from place to place. Slime drips from lower edges, and pale spider webs and threads fill some of the white corners and gaps. The finder cores (`finder_core=halloween`) are pumpkins, jack-o'-lanterns, ghosts or skulls, drawn at random. Every figure is drawn in the color of its tiles and only where modules are black, so the code reads the same, and with the same `seed` a code always gets the same figures. `\fancyqr@<figure>@chance` sets how often a figure appears (for example `\fancyqr@bat@chance`, default .2), `\fancyqr@lime@chance` (.35) how often a tile drips, `\fancyqr@web@chance` (.4), `\fancyqr@bigweb@chance` (.008) and `\fancyqr@thread@chance` (.08) how many webs and threads there are.
- `castle` v2.5 Rows of tiles become walls with battlements, columns towers with roofs and flags, single modules turrets, tents, wells, houses or trees, and larger blocks keeps, gatehouses, halls, castles and citadels, all in masonry whose colors follow the gradient. The stones follow the gradient. With the colors of the style, every region of the code gets its own stone, mixed in waves between a grey granite and a reddish brick, and its roofs and banners lean to blue, orange or green, so the town looks built over time. With colors you set, the stones keep your colors and only their lightness varies, and roofs, banners and trees are darker or lighter shades of them. The finder patterns are fortresses with a moat, a drawbridge mostly at the bottom and a keep, the alignment patterns ponds with a boat. Rivers, training grounds and, rarely, a princess, archer or knight fill the land in between, which lies on grass (`background`). `\fancyqr@<building>@chance` sets how often a building appears (for example `\fancyqr@keep@chance`, default .6), `\fancyqr@princess@chance` (.25) how likely the figures are.
- `christmas` v2.5 Trees, candy canes, presents, baubles, stockings, sledges, trains, toys and nutcrackers take the place of dark modules that fit their shape, the timing patterns carry lights and baubles, and snow lies on top as in `snow`. The finder patterns are wreaths around a core drawn at random: a bauble, a ring of lights, a gift, a wreath, a snowflake or candy canes. `\fancyqr@<figure>@chance` sets how often a figure appears, for example `\fancyqr@tree@chance` (1) or `\fancyqr@present@chance` (.15).
- `snow` v2.5 Snow lies on the finder rings without tiles on top and on most upper edges, in wavy layers that run on across neighbouring tiles. `\fancyqr@snow@chance` (.7) is the chance that a row of tiles gets snow, `\fancyqr@snow@height` (.32) how high it lies and `\fancyqr@snow@wave` (.1) how high its waves are. `\fancyqr@snow@color` and `\fancyqr@snow@line` are its fill and outline.

Some styles also set finder shapes: `frame` draws square finder patterns with round cores, which it needs to scan, `dots` and `blobs` round ones and `snow` rounded ones. `halloween`, `castle` and `christmas` also bring their colors, with pumpkins, fortresses and wreaths as finder patterns. Options given to `\fancyqr` override them, and the next style puts them back.

```
\FancyQrLoad{glows}
\FancyQrGlow{.5}{60}{2}
\FancyQrGlowColor{white}{black}
\fancyqr[seed=2]{https://ctan.org}
```



```
\FancyQrLoad{halloween}
\makeatletter
\def\fancyqr@slime@chance{.9}
\makeatother
\fancyqr[seed=5]{https://ctan.org}
```



3 Figures and extras

halloween, **castle** and **christmas** draw figures on blocks of dark modules that fit their pattern, and extras around them, such as webs, rivers or people, that scanners take for white. The lists below take names separated by commas. A trailing ***** makes a name a prefix: it matches every name that begins with the text before it. So **cane*** matches **cane/R**, **cane/L** and the other canes, **small*** matches **smalltree** and **smallfir**, and **land/*** every extra of the group **land**. It works only at the end of a name.

- **figures**={<name>},..., all or none (default all) v2.5 draws only the figures named.
- **no figures**={<name>},... v2.5 leaves out the figures named.
- **place**=<name>=<row>/<column>,... v2.5 puts a figure with the top left of its block at a module, counted from 1 at the top left, before all others. Where its pattern does not fit, it takes the closest place that fits.
- **faint**=true or false (default false) v2.5 draws the code in a faint grey, but the figures named in **place** in full colors.
- **figure chance**={<name>}=<chance>,... v2.5 sets how often a figure takes a block that fits, from 0 to 1.
- **extras**={<name>},..., all or none (default all), **no extras**={<name>},... v2.5 do the same for the extras.
- **\FancyQrListFigures**, **\FancyQrListExtras** write the names for the loaded style to the log.
- **\FancyQrShowFigure**[<options>]{<name>} draws a figure of the loaded style on its own, at the current position, at a module size of **\FancyQrFigureModule** (3mm), so it also works in running text. The starred form draws the module grid over it. **\FancyQrShowPattern**{<name>} draws the pattern it replaces.
- **\FancyQrTileGallery**[<options>], **\FancyQrFigureGallery**[<options>], **\FancyQrExtraGallery**[<options>] draw the plain tiles on a few shapes (**\FancyQrTileShapes**), every figure with its pattern, name and size, and a code with faint tiles for every extra.

A hat and a jack-o'-lantern in a line of text:

```
\FancyQrLoad{halloween}
\FancyQrFigureModule=1.2ex
Beware the \FancyQrShowFigure{hat} and
the \FancyQrShowFigure{bigjack}!
```

Beware the 
and the !

Only keeps, towers and huts, a hut on every lone module:

```
\FancyQrLoad{castle}
\fancyqr[seed=2,figures={keep,tower,hut},
figure chance={hut=1}]{https://ctan.org}
```



All figures but trees and tents, without people and training grounds:

```
\FancyQrLoad{castle}
\fancyqr[seed=2,no figures={trees,tent},
no extras={people/*,land/training}]{https://ctan.org}
```



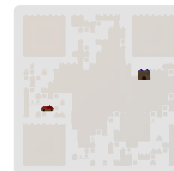
Just one figure, in one variant: only lollipops with the stick to the left, wherever they fit, and one more at a place of its own:

```
\FancyQrLoad{halloween}
\fancyqr[seed=2,figures={lollipop/left},
  figure_chance={lollipop/left=1},
  place={lollipop/left=10/10}]{https://ctan.org}
```



A keep near the middle and a house at the top, the rest faint:

```
\FancyQrLoad{castle}
\fancyqr[seed=2,faint,place={keep=12/12,
  house=9/10}]{https://ctan.org}
```



No figures, and of the extras only the water:

```
\FancyQrLoad{castle}
\fancyqr[seed=2,figures=none,
  extras=water/*]{https://ctan.org}
```



Castle walls with the spider webs of halloween, without its slime:

```
\fancyqr[seed=2,style=castle,add=halloween,
  no_extras=slime]{https://ctan.org}
```



Castles in winter:

```
\fancyqr[seed=2,style=castle,
  add={snow,christmas}]{https://ctan.org}
```



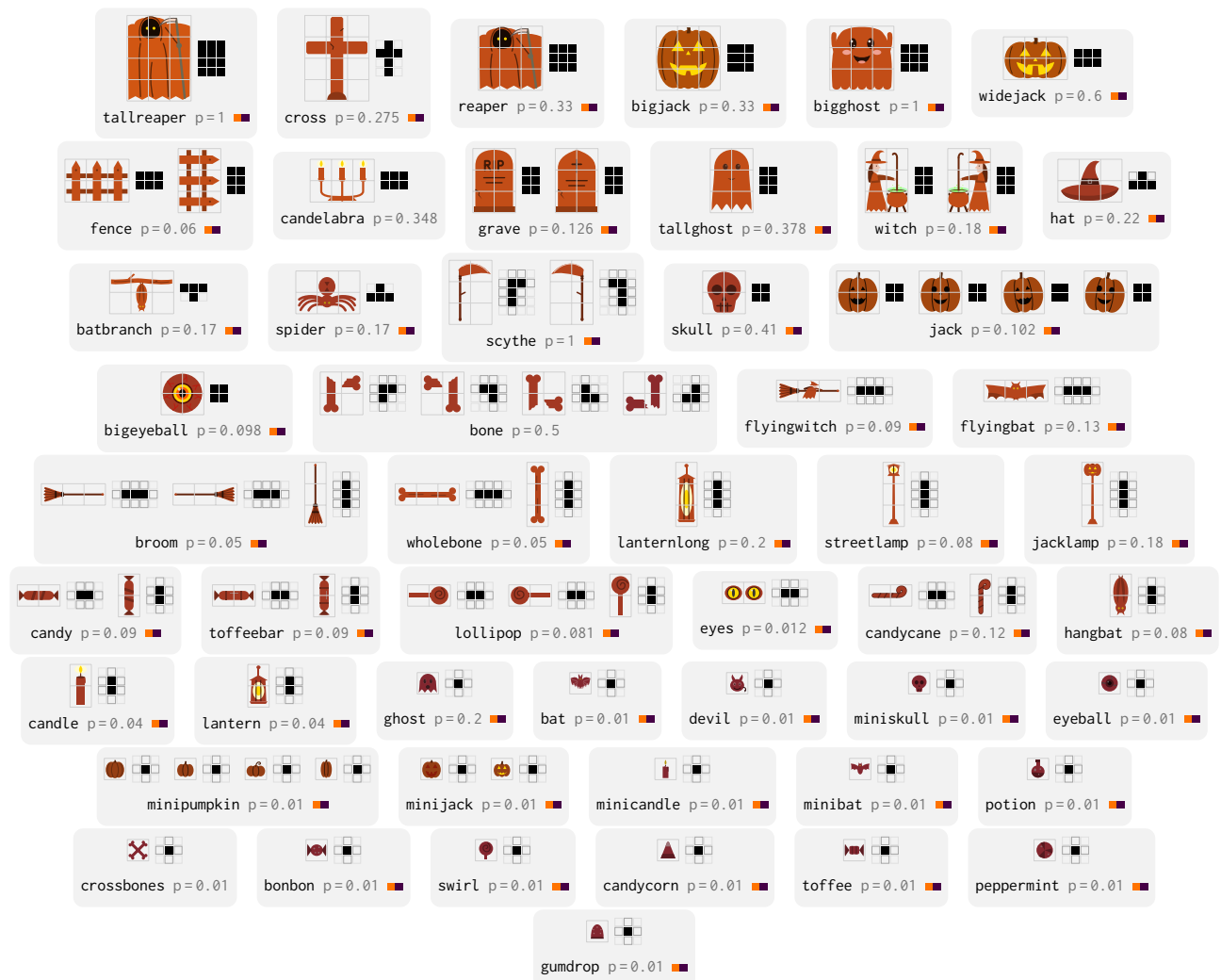
The sections below are drawn from the styles themselves. In a pattern, black is a dark module the figure takes and draws over, grey a dark module it needs next to it (a hat sits on one), an outline a module that must be white, so the figure has room and does not merge into a neighbour, and a faint outline a module that may be anything. A code draws its figures in variants that change with their place. A small swatch in the colors of the gradient marks the figures that take their colors from the code. The others keep their own.

3.1 halloween v2.5

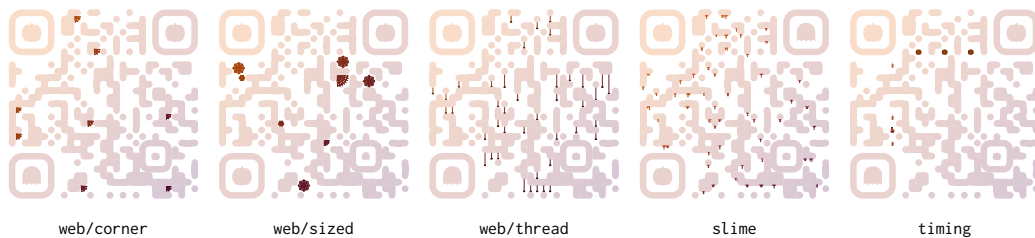
Tiles



Figures

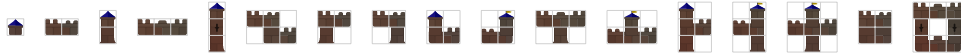


Decorations

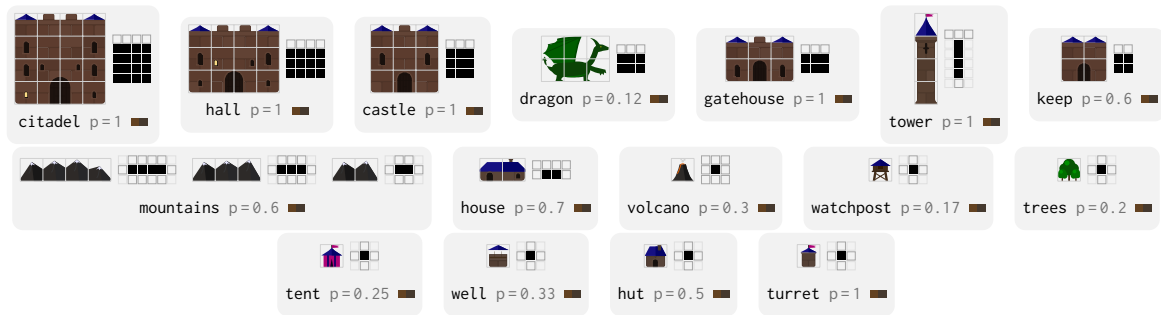


3.2 castle v2.5

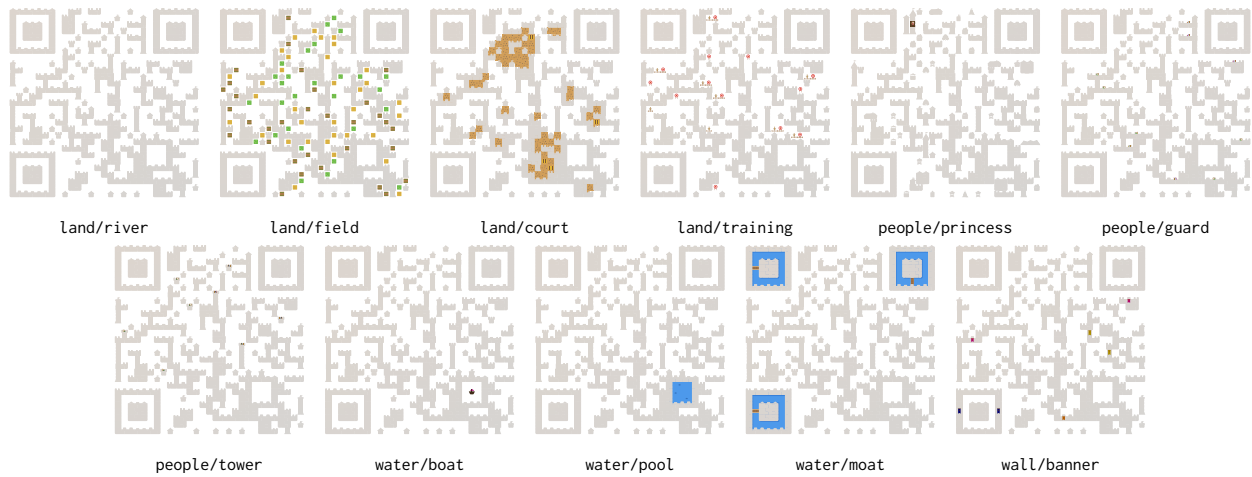
Tiles



Figures



Decorations

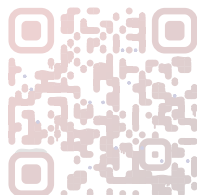


3.3 snow v2.5

Tiles



Decorations



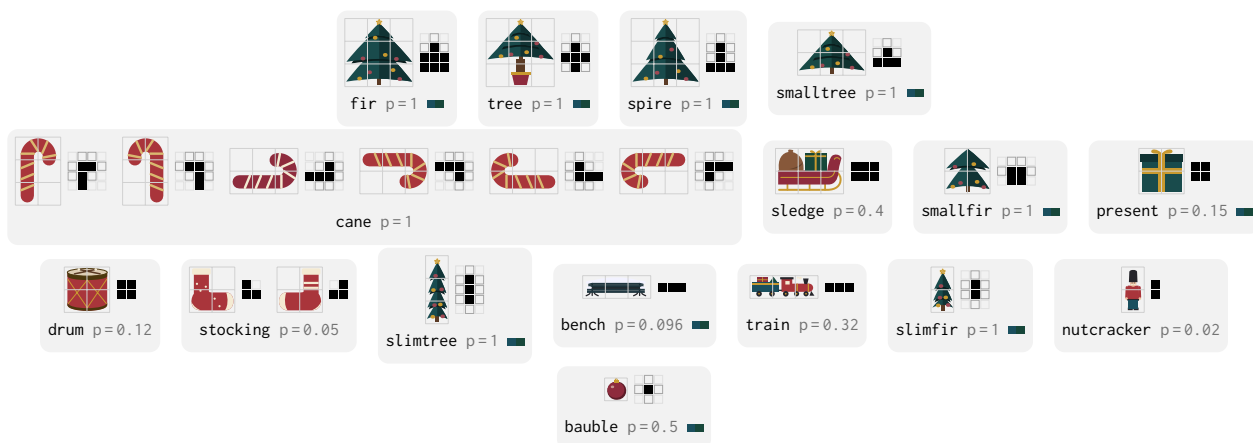
flakes

3.4 christmas v2.5

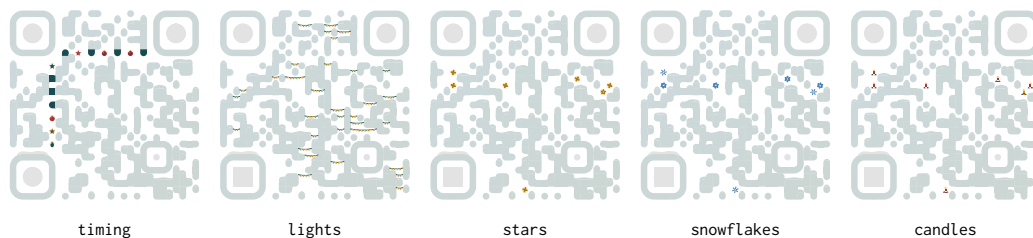
Tiles



Figures



Decorations



4 Finder and alignment patterns

Scanners find a QR code by the three big squares in its corners, the *finder patterns*. Codes from version 2 on also have smaller squares, the *alignment patterns*, which help with tilted or bent codes. Each pattern is a ring around a solid *core*. Unless you set the keys below, they are drawn like the data (Figure 2).

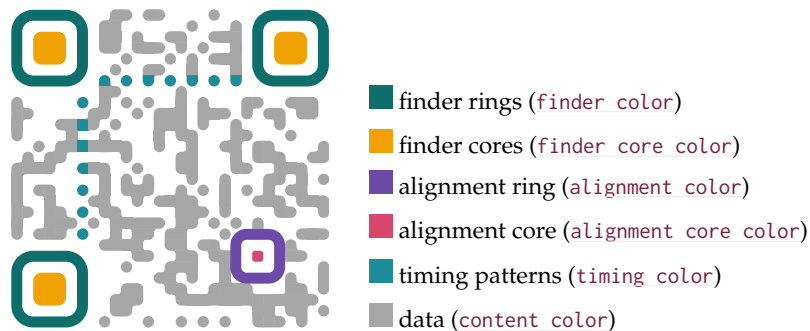


Figure 2: The parts of a QR code.

4.1 Shapes

- **finder**=`inherit`, `square`, `rounded` or `circle` (default `inherit`) the shape of the rings. `inherit` (or `none`) keeps the tiles of the style. Data tiles next to a straight part of a ring merge into it.
- **finder core**=`auto`, `inherit`, `square`, `rounded`, `circle`, `star` or `pumpkin` (default `auto`) the shape of the cores. `auto` takes the shape of the ring. A core shape also works when the ring keeps its tiles.
- **finder radius**=`0` to `0.5` (default `0.5`) the corner radius of `rounded`, relative to the size of the pattern.
- **finder thickness**=`<modules>` (default `1`) the width of the ring. Other values change the proportions scanners look for.

Loading a style resets `finder` and `finder core`.

4.2 Colors

- **finder color**=`<color>` the color of the patterns.
- **finder core color**=`<color>` v2.5 the color of the cores, by default `finder color`.
- **content color**=`<color>` the color of everything but the patterns.
- **alignment color**=`<color>` v2.5 the color of the alignment patterns, by default `finder color`.
- **alignment core color**=`<color>` v2.5 the color of their cores, by default `alignment color` if that is set, else `finder core color`.
- **timing color**=`<color>` v2.5 the color of the timing patterns, the dotted lines between the finder patterns.
- **background**=`<color>` v2.5 a background behind the whole code, the `padding` included. `None` by default, the `castle` style puts its castles on grass. Keep it light, scanners read it as white.
- **background radius**=`<modules>` (default `1`) v2.5 the rounding of the background's corners, `0` for square ones.
- **background margin**=`<length>` (default `3.5pt`) v2.5 how far the background reaches beyond the code on every side. The code takes that much more room.

Without them, the patterns keep the usual colors, normally the gradient. They work with every shape and with the tiles of a style (Figure 4). Light colors may lack contrast.



Figure 3: Shapes of the finder and alignment patterns.



Figure 4: Colors of the finder and alignment patterns.

4.3 Alignment patterns

- **alignment**=finder, inherit or hide (default finder) v2.5 how the alignment patterns are drawn. **finder** draws them like the finder patterns, **inherit** like the data, and **hide** replaces them with modules that look like the data around them (Figure 5).

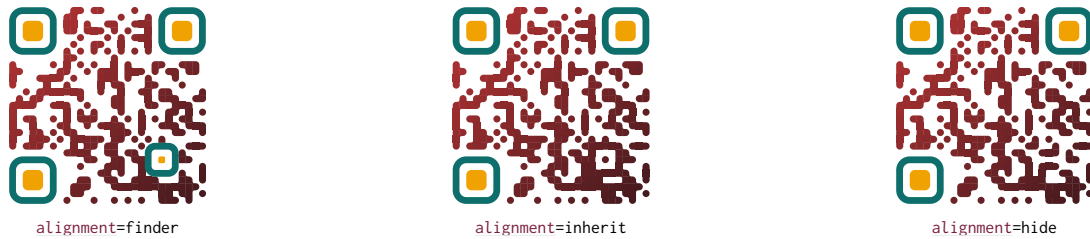


Figure 5: The settings of **alignment**, with **finder**=rounded, **finder_color**=ocean and **finder_core_color**=amber.

Test codes with **alignment=hide**. The alignment patterns are not part of the data, so error correction cannot replace them. Flat codes usually scan without them, but ZBar, a common scanner library, failed on most of our test codes.

Put settings you want for every code into **\fancyqrset**:

```
\fancyqrset{content_color=black,
  finder=rounded,
  finder_color=ocean,
  finder_core_color=amber,
  alignment_color=plum,
  alignment_core_color=rose}
\fancyqr{https://ctan.org}
```



5 Images and holes

- **image**=<content> puts the content in the center and hides the modules behind it. It keeps its own size and does not scale with the code.
- **image padding**=<modules> (default 0) hides this many more modules around the image.
- **image x padding**=<modules> (default 0), **image y padding**=<modules> (default 0) the same, left and right or above and below only.
- **\FancyQrDoNotPrintSquare**{<x>}{<y>} hides a rectangle without an image, <x> modules left and right of the center and <y> up and down.
- **\FancyQrDoNotPrintRadius**{<factor>} rounds the corners of the rectangle (0 to 1).
- **\FancyQrHardCut**, **\FancyQrRoundCut** cut the modules at the edge straight, or give them round ends (the default).

Hidden modules are lost data. If the code no longer scans, use a higher **level** or a smaller image. Finder and alignment shapes that reach into the hole are left out.

```
\FancyQrDoNotPrintSquare{3}{3}
\FancyQrDoNotPrintRadius{.6}
\fancyqr[level=H]{https://ctan.org}
```



6 Text beyond ASCII

Umlauts, accents, other scripts and emoji work with pdf $\LaTeX$, Lua $\LaTeX$ and Xe $\LaTeX$. The text is stored as UTF-8.

- `eci=true` or `false` (default `true`) v2.5 starts such a code with a header (an *ECI*) that says the text is UTF-8. It costs two bytes.

```
\fancyqr{Grüße aus Ulm}
```



Without the header, scanners guess the character set. Most phone apps guess right. ZBar shows GrüBe as Japanese. Google's ML Kit, on the other hand, reportedly ignores codes with the header. If an Android phone does not react to a code, try `eci=false`.

7 Faster runs

Styles with figures take a while to draw.

- `persist=true` or `false` (default `true`) v2.5 records every code in a file of its own, `<prefix><hash>.fqr`. Later runs replay the record instead of drawing the code again, so only the first run is slow. A record is read only when its code comes, so a long document needs no more memory than one code. A record is used only as long as the code, its options, the style, its settings and all codes before it stay the same, and a code is drawn the same either way. If you change the internal settings of a style (Section 2.2), delete the file.
- `externalize=true` or `false` (default `false`) v2.5 also turns every record into a pdf of its own, `<prefix><hash>.pdf`, and includes that. The pdf is made right away when the document is compiled with `-shell-escape`. Otherwise a warning tells you which file to compile. Codes with an image are only recorded. This manual is made that way.
- `externalize prefix=<prefix>` (default `<jobname>-fqr-`) v2.5 where the records and pdfs go. Old ones are never read again and can be deleted.

8 Writing a style

A style is a file `fancyqr-style-<name>.code`, read by `\FancyQrLoad` with `@` as a letter. It says how to draw a tile for each of the 16 combinations of black neighbours, with the commands of `pic2e`, in a picture where the module spans (0, 0) to (1, 1).

- `\newpattern{<up>}{<left>}{<right>}{<down>}{<code>}` sets the code for one combination. Each of the first four is 1 for a black neighbour.
- `\FancyQrRoundedPatterns{<macro>}` sets all 16. The macro gets one flag per corner (bottom right, top right, top left, bottom left) that is 1 where the corner is free and should be round.
- `\FancyQrSidePatterns{<macro>}` does the same with one flag per neighbour (up, left, right, down).
- `\FancyQrPasses{<n>}` draws the code in `<n>` layers, one after the other: every pass walks all modules again, and `\fancyqr@pass` tells a tile which layer it draws. What a later pass draws lies on top of the earlier ones, so a tile can put a glow, a shadow or water in the first pass and its stone or body in the second, without a neighbour's glow covering it. `glows` draws a ring of its halo per pass and the tiles in the last one, `snow` the tiles in the first and the snow in the second. `glows` draws its glow in the first passes, `snow` its snow in the last.
- `\FancyQrInnerRoundings` says that the style draws inner corners (`inner rounding`), `\FancyQrNoInnerRoundings` that it does not.

- `\FancyQrStyleSet{<options>}` sets options such as colors and finder shapes, and undoes them when the next style loads.

Inside a tile, `\@up`, `\@left`, `\@right` and `\@down` hold the neighbours as well.

- `\FancyQrDiagonals` also gives the tiles `\@upleft`, `\@upright`, `\@downleft` and `\@downright`.
- `\FancyQrFigure{<name>}{<chance>}{<pattern>}{<code>}` replaces blocks of dark modules with a figure. The pattern lists the rows of the block and of the modules around it, separated by `/`: `x` is a dark module of the figure, `1` any dark module, `0` a white one and `.` anything. Bigger figures are placed first, from the top left, wherever the pattern fits and a hash of the position is below the chance, so a code always gets the same figures. Figures never cover a finder or alignment pattern, a hidden center or another figure. The starred form only uses positions on a grid of the figure's size.
- `\FancyQrFigureTile{<tile>}` wraps the tile code of a style: it draws the figure at its first module, with the block from $(0, 1 - h)$ to $(w, 1)$, and nothing at its other modules. Tiles next to a figure see its modules as white, so they end against it like against white space.
- `\FancyQrFigureAt{<row>}{<column>}` expands to the name of the figure at that module, or to nothing.
- `\FancyQrIfTiming{<true>}{<false>}` tells a tile whether it belongs to a timing pattern, the dotted lines between the finder patterns, which every code has in the same place.

A candy cane on a lone vertical bar of three, in a style with rounded tiles:

```
\FancyQrFigure{cane}{1}{0x0/0x0/0x0}{\linethickness{.5\unitlength}%
\moveto(.6,-1.8)\lineto(.6,.6)\circlearc{.35}{.6}{.25}{0}{180}\strokepath}
\def\fancyqr@cane@tile#1#2#3#4{\FancyQrFigureTile{\fancyqr@rounded@rect#1#2#3#4}}
\FancyQrRoundedPatterns\fancyqr@cane@tile
```

```
% fancyqr-style-flat.code
\def\fancyqr@flat@rect{%
\moveto(0,0)\lineto(0,1)\lineto(1,1)\lineto(1,0)\fillpath}
\newpattern0000{\fancyqr@flat@rect}
\newpattern1000{\fancyqr@flat@rect}
% ... and the other 14 combinations
```

9 All options

Besides the keys of the previous sections, `\fancyqr` and `\fancyqrset` take these:

- **size**=<length> the size of the code, the height of **qr**code. **width**=<length> is the same.
- **level**=L, M, Q or H (default M) the error correction. Higher levels survive more damage, such as an image in the center.
- **version**=0 to 40 (default 0) the size of the module grid. 0 takes the smallest that fits.
- **padding**, **tight** a white border around the code, or none.
- **color**=<color> one color for the whole code, without gradient.
- **gradient**=true or false (default true) the color gradient.
- **gradient angle**=<degrees> (default 135) its direction. 0 runs from left to right.
- **l color**=<color>, **r color**=<color> the colors at the left and right end of the gradient. With the default angle, the bottom right and the top left. **left color**=<color> and **right color**=<color> are the same.
- **random color**=<{<color>}{<color>}. . . gives every module a random color from the list. Needs **pgf**.
- **classic**=true or false (default false) black squares, the **flat** style.
- **seed**=<number> fixes the random numbers of the random styles.

- **overlap**= $\langle fraction \rangle$ (default 0.045) how much neighbouring tiles overlap, against thin white lines between tiles in some PDF viewers. **compensate**= $\langle length \rangle$ sets it as a length.
- **cache**=true or false (default true) remember computed codes, so the same code is fast the next time.

10 History

Version v2.5 2026-10-18

- Text beyond ASCII with every engine, marked as UTF-8 (**eci**)
- The styles **halloween**, **snow**, **castle** and **christmas**, and the core shape **pumpkin**
- Figures that replace blocks of dark modules (**\FancyQrFigure**, **\FancyQrFigureTile**, **\FancyQrFigureAt**) and diagonal neighbours (**\FancyQrDiagonals**)
- **finder core color**, **alignment color**, **alignment core color**, and **alignment** to draw the alignment patterns like the data or to hide them
- **finder core** also works when the rings keep the tiles of the style
- Data tiles merge into the straight sides of a finder ring
- **frame** draws square finder patterns with round cores, so it scans as it is
- Loading a style undoes the colors and finder shapes of the previous one (**\FancyQrStyleSet**)
- Faster, especially for large codes and the **snow** style
- Bug fixes: alignment patterns are styled on the first run, **\FancyQrLoadDefault** resets the drawing passes, **\FancyQrLoad** adds no spaces in running text, and a hole in the center (**\FancyQrDoNotPrintSquare**) no longer breaks finder shapes or the next code with the same text

Version v2.4 2026-08-17

- Weakly rounded (negative) inner corners for the **default** style
- New **rounded**, **glows** and **goo** styles
- Style the position and alignment patterns with **finder**, **finder core**, **finder color**, **finder radius** and **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

- Relicense under the LPPL 1.3c

Version v2.2 2024-11-27

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

Version v2.1 2024-10-05

- 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

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

Version v1.0 2022-08-18

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

Index of commands, keys and styles

A

add (key)	2
alignment (key)	12
alignment color (key)	10
alignment core color (key)	10

B

background (key)	10
background margin (key)	10
background radius (key)	10
blobs (style)	2, 4

C

cache (key)	15
castle (style)	4, 5, 10, 15
christmas (style)	2, 4, 5, 15
classic (key)	14
color (key)	14
compensate (key)	15
content color (key)	10

D

default (style)	2, 3, 15
dots (style)	4

E

eci (key)	13
externalize (key)	13
externalize prefix (key)	13
extras (key)	5

F

faint (key)	5
\fancyqr	2
\FancyQrAdd	2
\FancyQrDiagonals	14
\FancyQrDoNotPrintRadius	12
\FancyQrDoNotPrintSquare	12
\FancyQrExtraGallery	5
\FancyQrFigure	14
\FancyQrFigureAt	14
\FancyQrFigureGallery	5
\FancyQrFigureTile	14
\FancyQrGlow	4
\FancyQrGlowColor	4
\FancyQrHardCut	12
\FancyQrIfTiming	14
\FancyQrInnerRoundings	13
\FancyQrListExtras	5
\FancyQrListFigures	5
\FancyQrLoad	2
\FancyQrLoadDefault	2
\FancyQrNoInnerRoundings	13
\FancyQrPasses	13
\FancyQrRoundCut	12
\FancyQrRoundedPatterns	13

\FancyQrSeed	2
\fancyqrset	2
\FancyQrShowFigure	5
\FancyQrShowPattern	5
\FancyQrSidePatterns	13
\FancyQrStyleSet	14
\FancyQrTileGallery	5
figure chance (key)	5
figures (key)	5
finder (key)	10
finder color (key)	10
finder core (key)	10
finder core color (key)	10
finder radius (key)	10
finder thickness (key)	10
flat (style)	14
frame (style)	4, 15

G

glitch (style)	2, 4
glows (style)	4, 13, 15
goo (style)	2, 4, 15
gradient (key)	14
gradient angle (key)	14

H

halloween (style)	2, 4-6, 15
--------------------------------	------------

I

image (key)	12
image padding (key)	12
image x padding (key)	12
image y padding (key)	12
inner rounding (key)	3

L

l color (key)	14
left color (key)	14
level (key)	14

N

\newpattern	13
no extras (key)	5
no figures (key)	5

O

overlap (key)	15
----------------------------	----

P

padding (key)	14
persist (key)	13
place (key)	5

R

r_color (key)	14
random_color (key)	14
right_color (key)	14
rounded (style)	3, 15
rounding (key)	3

S

seed (key)	14
size (key)	14
snow (style)	2, 4, 13, 15

style (key)	2
--------------------------	---

T

tight (key)	14
timing_color (key)	10

V

version (key)	14
----------------------------	----

W

width (key)	14
--------------------------	----