The qrcode package: Quick Response code generation in LaTeX*

Anders Hendrickson
St. Norbert College, De Pere, WI, USA
anders.hendrickson@snc.edu

September 26, 2014

1 Introduction

The proliferation of smartphones and tablets has led to the widespread use of Quick Response (QR) codes, which encode numeric, alphanumeric, kanji, or binary information into a square matrix of black and white pixels called modules. Although QR codes can encode any information up to almost three kilobytes, their most common use is as physical hyperlinks: a mobile device scans a printed QR code, decodes a URL, and automatically points a browser to that location.

It is natural to want to include QR codes in certain LaTeX documents; for example, one may want to direct the reader of a printed page to related interactive content online. Before now, the only LaTeX package for producing QR codes was the immensely flexible `pst-barcode`. As that package relies on `pstricks`, however, it can be difficult to integrate with a pdfLaTeX workflow,[1] and a pdfLaTeX user may not want the extra overhead just to produce a QR code. If one wants to avoid `pstricks`, a LuaTeX solution was proposed at `http://tex.stackexchange.com/questions/89649/`, and a plainTeX solution can be found at `http://ktiml.mff.cuni.cz/~maj/QRcode.TeX`, but until now no LaTeX package had been available that did not call on outside machinery.

The `qrcode` package, in contrast, implements the QR code algorithm using only TeX and LaTeX commands, so it should work with any LaTeX workflow. Because it draws the squares constituting a QR code using the TeX primitive `\rule`, there

*This document corresponds to `qrcode` v1.0, dated 2014/09/26.

[1] The `auto-pst-pdf` or `pstool` packages can make this possible by automatically running LaTeX → `dvips` → `ps2pdf` → `pdfcrop` for each barcode generated in `pstricks`, so long as the user is able and willing to enable `\write18` in `pdflatex` and install Perl. Judging by questions on `tex.stackexchange.com` and `latexcommunity.org`, this is a significant hurdle for some users. Moreover, according to `http://tex.stackexchange.com/questions/72876/` this workflow may have trouble if the QR code is in a header.

is no need to load any graphics package whatsoever. For a user who merely wants a QR code, this is the simplest solution.

2 Usage

`\qrcode` The package provides just one command, `\qrcode`, with the following syntax:

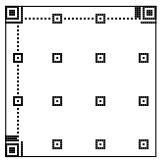
`\qrcode[`⟨*options*⟩`]{`⟨*text to be encoded*⟩`}`

For example, `\qrcode[height=0.5in]{http://www.ctan.org}` produces

Although the most common use of QR codes is as URLs, the ⟨*text to be encoded*⟩ can be almost any typed text. The few exceptions to this are described in section 2.3.

2.1 Package Options

`draft` `final` Creating QR codes for short URLs takes relatively little time.[2] Because TEX was designed for typesetting, not for extensive computations, however, if many small QR codes or a single large one are required, the time spent can be quite noticeable. To save compilation time while working on a large document, calling the `draft` option causes the package not to compute QR codes, but merely to insert placeholder symbols with no data. The `final` option is an antonym to `draft` and is the default.



```
\documentclass{article}
\usepackage[draft]{qrcode}
\begin{document}
  \qrcode[version=15]{Dummy code}
\end{document}
```

The placeholder symbol produced in `draft` mode will have the same size and dimensions as the actual QR code.

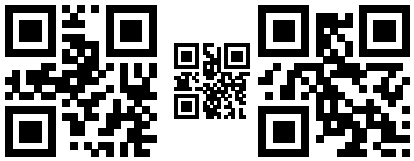
To conserve processing time, when `\qrcode` computes the binary matrix representing a QR code, it saves that binary data as a string of 1's and 0's both in a macro and in the `.aux` file. Thus if the same QR code is desired later in the document, or upon the next run of LATEX, the QR symbol can be redrawn immediately from the saved binary data.

`forget` There may be times when this is not desired; testing of this package is the chief example, but one might also have reason to believe that the `.aux` file contains bad data. Invoking the `forget` package option causes `\qrcode` to calculate every QR code anew, even if a QR code for that ⟨*text to be encoded*⟩, level, and version was read from the `.aux` file or was already computed earlier in the document.

[2] On this author's laptop, even a 60-character URL (version 4, level M) adds only about 0.7 seconds of compilation time.

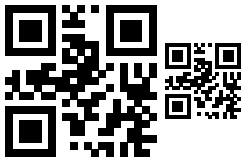
2.2 Options

`\qrset` Several options affect the appearance and encoding of the QR code; `qrcode` uses the `xkeyval` package to handle the setting and processing of key-value pairs. The following options may either be given as optional arguments to `\qrcode` or changed within a TEX-grouping using the macro `\qrset`.



```
\qrcode{ABCD}
{\qrset{height=1cm}%
 \qrcode{EFGH}}
\qrcode{IJKL}
```

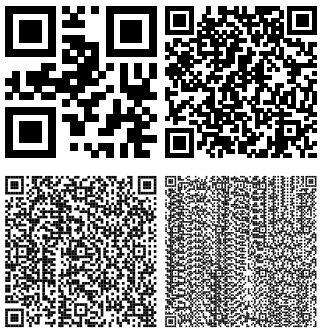
`height` The `height=`⟨*dimen*⟩ key sets the printed height (and width) of the QR code. The default value is `2cm`.



```
\qrcode{ABCD}
\qrcode[height=1cm]{ABCD}
```

`level` The QR code specification (ISO 18004:2006) includes four levels of encoding: Low, Medium, Quality, and High, in increasing order of error-correction capabaility. In general, for a given text a higher error-correction level requires more bits of information in the QR code. The key `level=`⟨*level specification*⟩ selects the minimum acceptable level. The ⟨*level specification*⟩ may be `L`, `M`, `Q`, or `H`; the default is `M`. It may happen that the smallest QR code able to encode the specified text at the desired level is in fact large enough to provide a higher level of error-correction. If so, `qrcode` automatically upgrades to the higher error-correction level, and a message is printed in the log file.

`version` QR codes range in size from 21×21 modules ("version 1") to 177×177 modules ("version 40"), in steps of 4 modules. The package automatically selects the smallest version large enough to encode the specified text at the desired error-correction level. Nevertheless, there might be occasions when a specific version is required; for example, perhaps a set of QR codes should have the same dimensions for aesthetic reasons, even though some encode shorter texts than others. For this reason, the key `version=`⟨*version specification*⟩ allows the user to specify a minimum version number, from 1 through 40, for the QR code. Setting `version=0` means "as small as possible"; this is the default. If the desired version is not large enough to encode the text, the version will automatically be increased to accommodate the text, and a message will be placed in the log file.



```
\qrcode{ABCD}
\qrcode[version=5]{ABCD}
\medskip \\
\qrcode[version=10]{ABCD}
\qrcode[version=20]{ABCD}
```

`tight` `padding`

The QR specification states that a QR code should be surrounded by white-space of a width equal to that of four modules. In many applications, a document author is likely to provide sufficient spacing anyway (e.g., by placing the QR code in a `center` environment, header, or `\marginpar`), so by default the `qrcode` package adds no spacing. If the option `padding` is specified, however, the QR code will automatically be surrounded with 4 modules' worth of whitespace. The key `tight` is an antonym of `padding`; the default is `tight`.

2.3 Special characters

Many URLs can be processed by TEX with no hiccups, but not infrequently a URL may contain the symbols `%`, `#`, `~`, `_`, and `&`. Moreover, QR codes need not just contain URL's, so a user may wish to encode text containing `^`, `$`, or spaces. The `qrcode` package offers two ways of coping with these special characters.

First, the `\qrcode` command itself processes its ⟨*text to be encoded*⟩ in a limited verbatim mode. The following characters will be encoded into the QR code as typed:

`# $ & ^ _ ~ % ␣`

Conspicuously absent from this list are `\`, `{`, and `}`. This is intentional, so that macros may be used within `\qrcode` to generate the ⟨*text to be encoded*⟩ automatically. If these characters are desired, they may be obtained by "escaping" them with an extra backslash:

```
\qrset{height=1.5cm}%
\qrcode{We can include #$&^_~%.}
\def\foo{bar}%
\qrcode{Set the \foo\ high.}
\qrcode{We must escape \\emph\{this\}.}
```

As with all verbatim modes, however, because TEX irrevocably sets catcodes when it first encounters characters, this will not work if the `\qrcode` macro is contained in another macro. If you call `\qrcode` inside an `\fbox` or a `\marginpar`, for example, and if your URL contains one of those special characters, you will either encounter error messages or (worse, because it is undetectable to the naked eye) have the wrong QR code typeset. In this scenario, you can still include any

of the characters `#$&^_~%␣\{}` by escaping them with an extra backslash; so long as they eventually pass unexpanded to `\qrcode`, they will produce the correct QR code.

```
\fbox{qrcode[height=1cm]{\#\$\&\^\_\~\%\ \\\{\}}}
```

3 Limitations and Cautions

- The QR specification includes modes for encoding numeric, alphanumeric, or Kanji data more efficiently. This package does not (yet) offer those options.
- The QR specification offers ways to string lengthy data across multiple QR codes. This package does not implement that possibility.