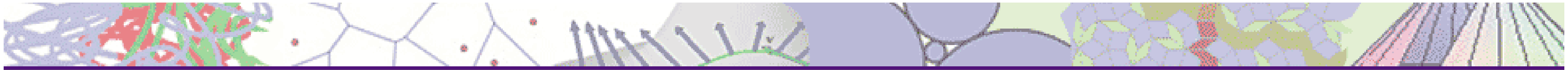


FEATURE COLUMN Monthly essays on mathematical topics

How to Read QR Symbols Without Your Mobile Telephone

How does your phone read the information in a QR code symbol? How is information stored on it? In this column I'll make a start, but really only a start, on answering these questions....

Bill Casselman
University of British Columbia, Vancouver, Canada
Email Bill Casselman

Mail to a friend Print this article

Welcome to the Feature Column!

These web essays are designed for those who have already discovered the joys of mathematics as well as for those who may be uncomfortable with mathematics.
Read more . . .

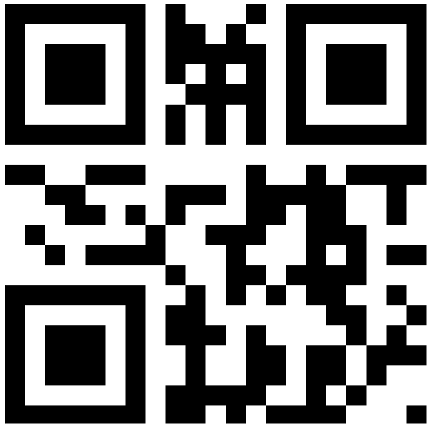
Search Feature Column

Google™ Custom Search

Feature Column at a glance

Introduction

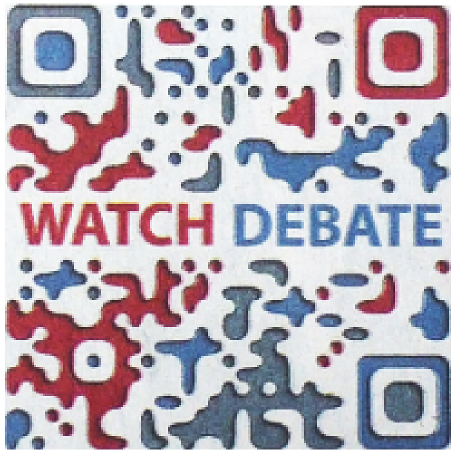
QR code is what you find on those little black and white things that look like



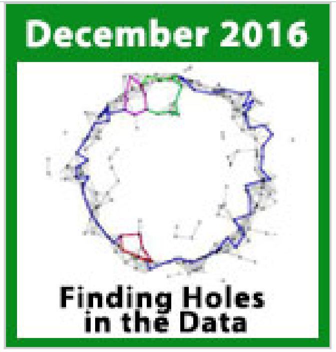
or

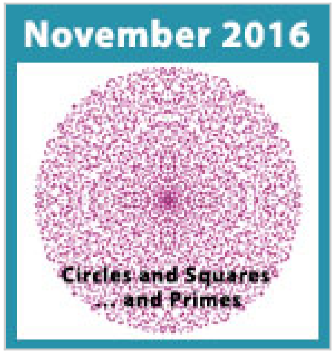
.

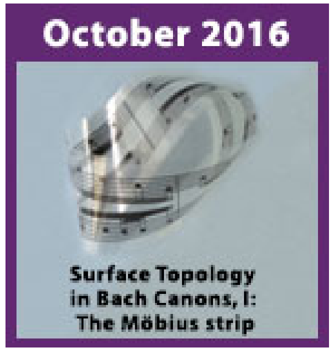
They can be a bit more decorative:



(The QR symbol on the left was an ad for the Canadian Liberal Party leaders' debate. On the right ... well ... it's February.) These do not exactly make much sense to the unaided human eye. They are, however, still quite useful, as this store display from Bonn, Germany shows in a hopeful if unusual hand-painted message about a store for rent:

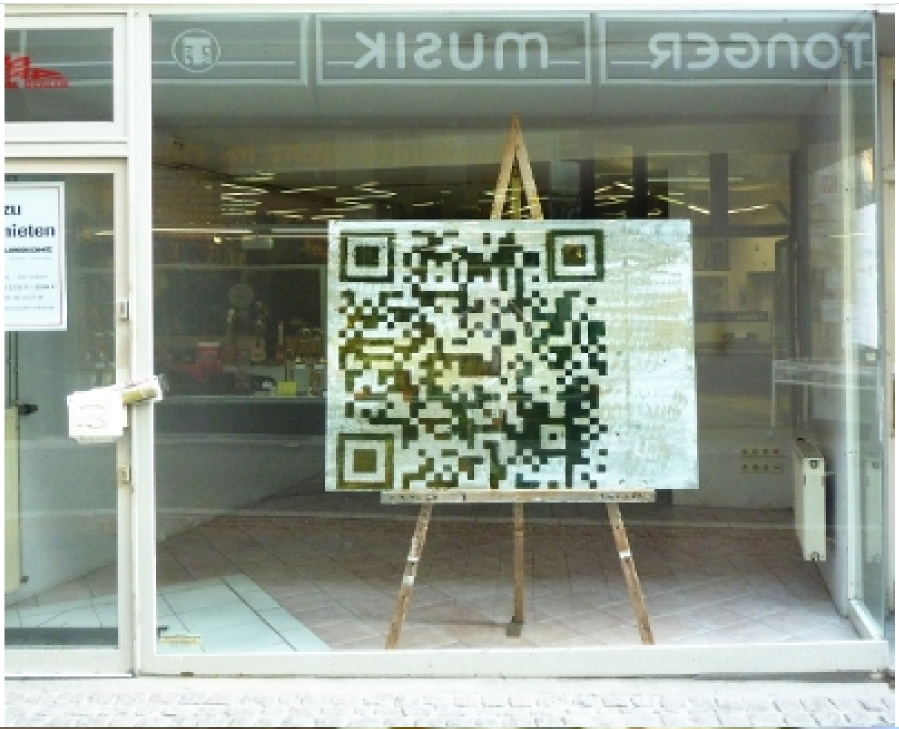


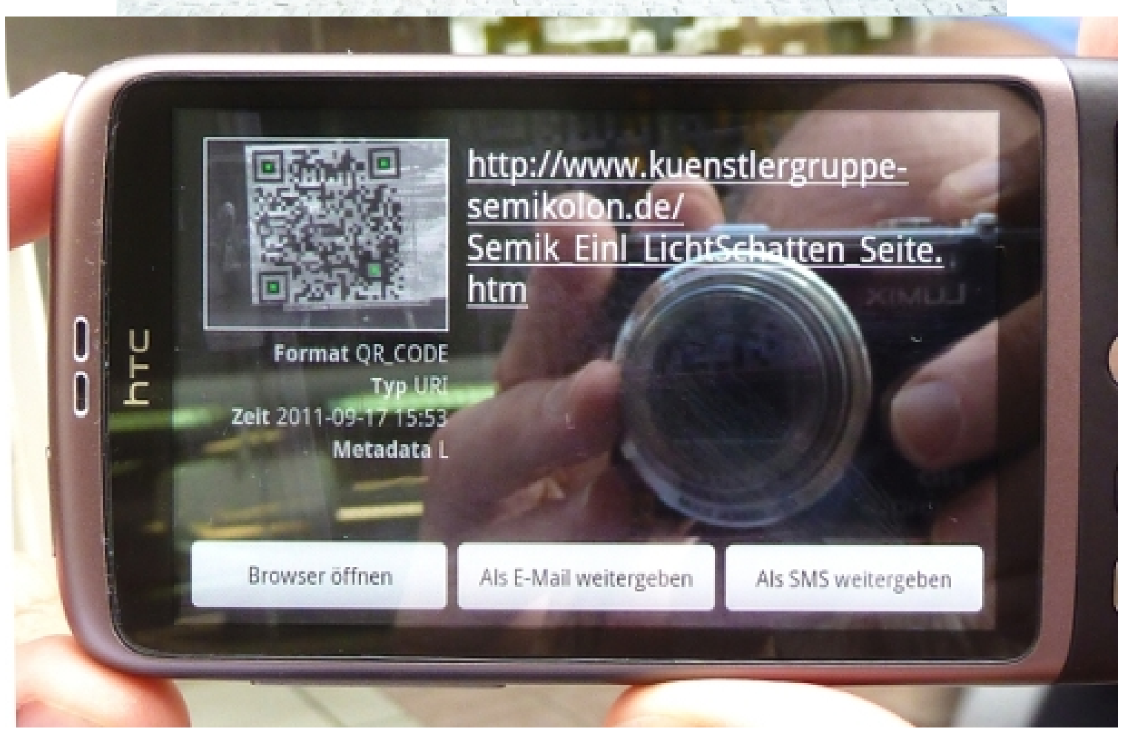




View Full Archive

Browse Subjects





The point is that these symbols are not meant to be read by the unaided human eye, but instead by a computer's aided eye. If your mobile phone has a camera and the right app loaded into it, it can interpret one of these. In fact, this example is perhaps a bit out of date, since many phone apps nowadays will read the symbol and directly take you to a URL embedded in it. Other applications include some very useful ones such as linking your phone to a real-time bus-schedule, paying for downtown parking, recalling where your car was parked, and how to purchase advertised items. As well as some of more doubtful taste, such as learning all about someone's dearly departed.

How does your 'phone read the information in a QR code symbol? How is information stored on it? In this column I'll make a start, but really only a start, on answering these questions.

QR code symbol basics

QR symbols were first developed by the Japanese company DENSO, a subsidiary of Toyota, apparently in order to track manufactured items. The company has released the patent for free public use. In Japan, even now, these symbols are far more common than elsewhere in the world.

The exact definition of QR symbols is contained in an ISO/IEC manual of about 114 pages. The manual is not light reading. One reason is that it contains much necessarily technical information, and the symbols are organized for efficiency rather than elegance and simplicity. Another is that there are several options for QR code symbols, and some correspondingly complicated programming is involved. Each symbol is a square array of smaller squares, which are called **modules** in the ISO manual. Sizes range from 21×21 to 177×177. Each size is indexed by a **version number**, ranging from 1 to 40. The size of version V is $N \times N$ with $N = 17 + 4V$. Every symbol amounts to a layout of bits stored in modules, equal to 0 or 1 according to whether the module is light or dark. The bits are grouped into **bytes** of 8 bits each, but the interpretation of the symbol is through a linear arrangement of bits. The point of the grouping into bytes is to allow for recovery from damaged symbols through what are by now classical and even ubiquitous techniques of error correction.

For a fixed size, there are a fixed number of possible data bytes, but the actual number of bytes which originate in an actual message depends on the level of error correction (EC) implemented. The data bytes are partitioned into three parts: message, padding, and error correction. The length of the message + padding is fixed by the size and level of error correction. There are four levels possible. For example, for version 1, of size 21×21, the total number of data bytes is 26, but in the various levels of error correction only 19, 16, 13, or 9 can be chosen for the message itself, according to the EC level. In these, 2, 4, 6, or 8 data words can be restored (respectively) if damaged or lost. In version 40, of size 177×177, there are $3,706$ data words, of which $2,986$, $2,390$, $1,756$, and $1,396$ can be chosen freely according to EC level.

There are other variations. After a message has been chosen and appropriate bits added to enable error correction, a **mask** is laid over the symbol to make it easier for a device to figure out what it is looking at by randomizing the black and white pattern. Furthermore, there are different **modes** of interpretation. There is one for purely numeric input, one for simple alpha-numeric input, another for an extended form of standard ASCII text that incorporates the Japanese kana alphabet, and yet one more that allows the much larger set of Kanji, the character adopted by the Japanese from what were originally part of the Chinese language.

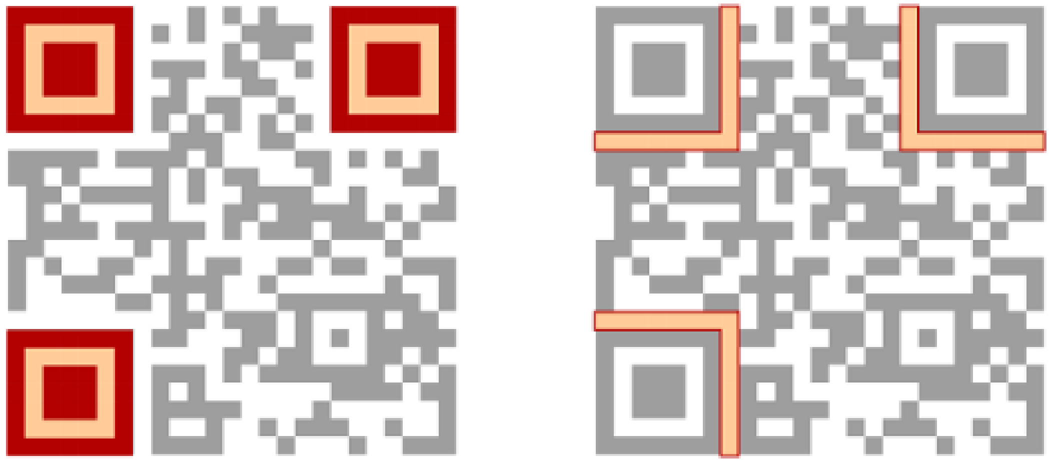
Comments?

QR geometry

The modules in a QR symbol can be grouped into several distinct components:

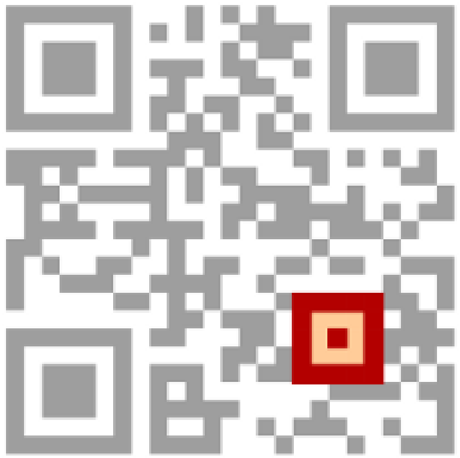
The fixed pattern

- Position symbols and their borders



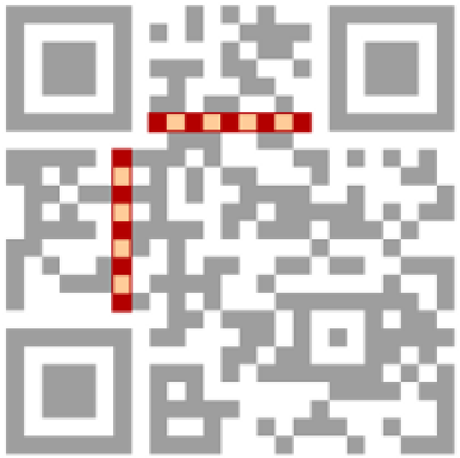
These are used by the device attempting to read the symbol. They fix the rotation and basic dimensions.

- Alignment symbols



The number of these grows with the size of the symbol. For version 40 (of size 177×177), there are 46 of these. I have not seen an explanation of their function, but presumably they help correct for perspective, curvature, and other distortion.

- Timing arrays

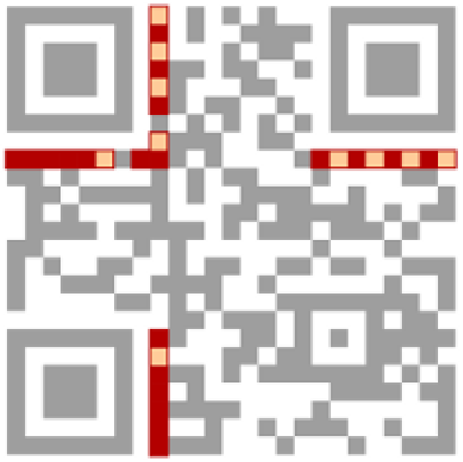


These help to determine the dimensions of the symbol.

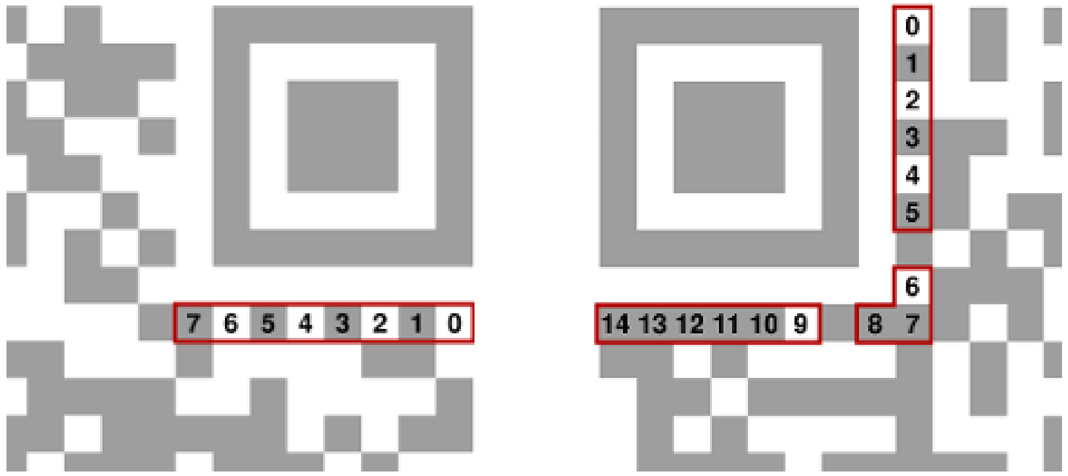
Variable data

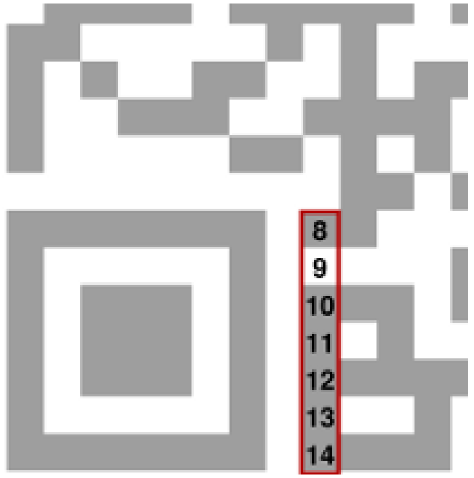
The components so far do not vary with the message, and are not liable to interpretation. They are fixed for each size. The components listed next convey information. Ther first two provide information about choices made that determine interpretation, and the rest are part of the actual message.

- Format information



This contains two pieces of information, the level of error correction chosen, and the index of the mask laid over the original message. Because it is crucial to be able to read this, it is stored in several places.



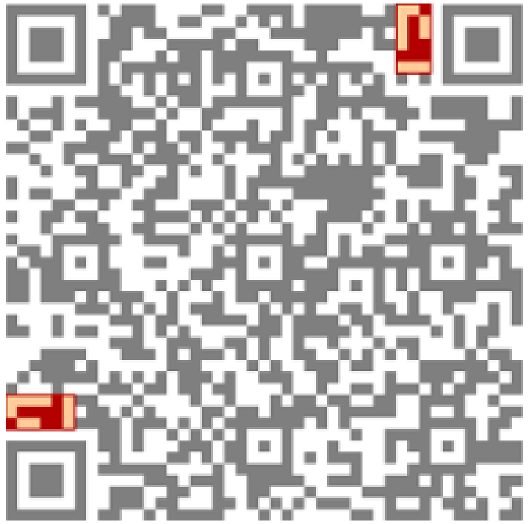


The basic format information is a sequence of bits, 0 through 14, of which 5 are information and 10 are for error correction. The information bits are in the segement 10-14. Bits $(14,13)$ hold the error level and $(12,11,10)$ the mask ID. But these cannot be read directly from the symbol, because the format information is overlaid by a mask, which is the bit sequence 101010000010010. In the symbol the high bits are 11 and 111. But to unmask we do some arithmetic modulo 2: $$ \matrix { 11111 \cr 10101 \cr 01010 \cr } $$ so that after unmasking these become 01 and 010. We shall see later exactly what this means for reading the symbol.

But what I want to do right here is say something about the 10 bits of error correction, because it is a simpler version of what the data does, and is somewhat more comprehensible. The best way to think of the 15 bits in the format region is as the coefficients of a polynomial, all of whose coefficients are either 0 or 1. For the example, the polynomial is $$ P(x) = x^{14} + x^{13} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{3} + x . $$ The actual information to be conveyed by this is contained in the terms of degree 10 through 14, so here in $$ M(x) = x^{14} + x^{13} + x^{12} + x^{11} + x^{10} . $$ The coefficients are to be taken as integers modulo 2. With this interpretation, the terms of degree less than 10 make up a polynomial $R(x)$ with the property that the entire polynomial $M(x) + R(x)$ is divisible by a somewhat arbitrarily chosen **generator polynomial**, which is in fact $$ G(x) = x^{10} + x^{8} + x^{5} + x^{4} + x^{2} + x + 1 . $$ In other words, taking into account that $1 + 1 = 0$ in arithmetic modulo 2, we get the error correction $R(x)$ from $M(x)$ by taking the remainder after division of $M(x)$ by $G(x)$, since if $$ M(x) = Q(x) G(x) + R(x) $$ then also $$ M(x) + R(x) = Q(x)G(x) . $$ In other words, the acceptable format strings are precisely those that represent polynomials divisible by the generator polynomial $G(x)$.

I'll not say anything here about how the error correction bits can be used to restore damage. Although that is the most interesting mathematical point, it is a bit too complicated to discuss here.

- Version information



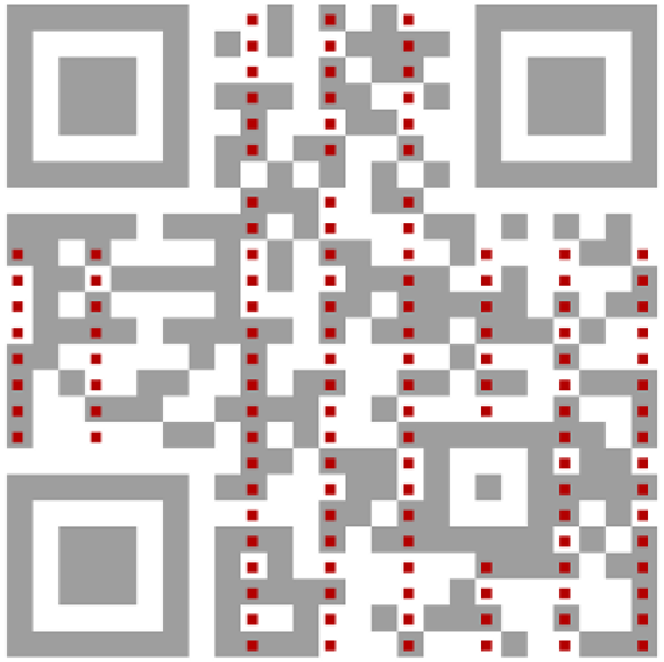
This appears only for large sizes, versions 7 and higher. It specifies the version number. Presumably having this saves time and is more accurate for these sizes than attempting to deduce the same information from the symbol itself. This also is stored in two places. The version bits are not masked, and the version number itself is stored in the 6 highest bits. Thus in this example the version pattern is $$ \matrix { 001 \cr 010 \cr 010 \cr 011 \cr 111 \cr 000 \cr } . $$ Reading backwards, this becomes the number expressed in binary notation as 000111110010010100, which tells us (as we could have already figured out from its size 29×29) that this symbol is version $4 + 2 + 1 = 7$.

- Message bits
  These contain the actual message. I'll say more about this component and the other parts of the data bytes a bit later on.
- Padding
  For a given choice of size and error correction level, the number of available words for a message is fixed. But since the message might be somewhat shorter than what is allowed, it is padded with a more or less fixed pattern of bits to fill up space.
- Error correction bits
  These implement one of the standard BCH coding schemes.

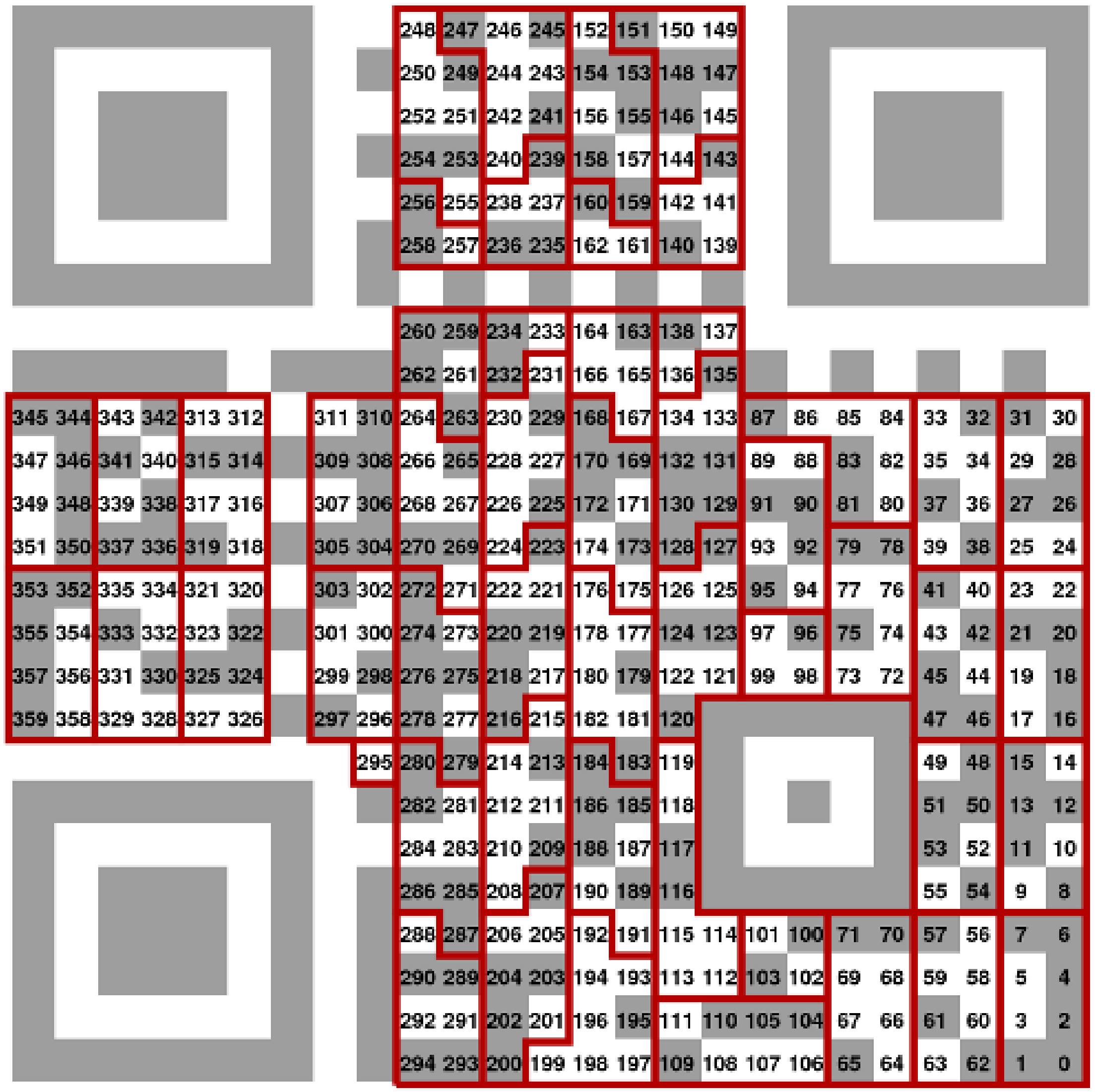
Comments?

Taking the mask off

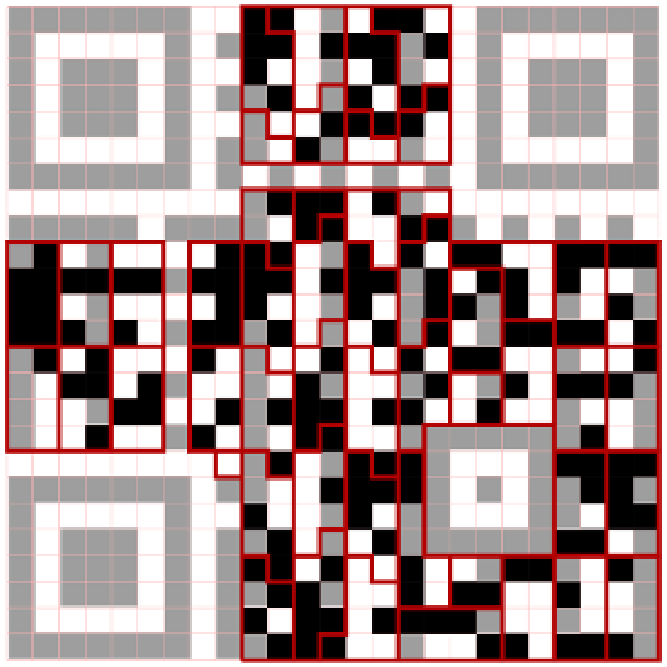
The data bits in the symbol must be unmasked to see the original data bits. The mask id here, as we have seen, is 010, and this mask is indicated in the following picture by red marks. In other words, the mask amounts to columns modulo 3. The fixed pattern is not masked, nor are the format bits (which have their own mask).



The modules are now numbered. The modules belonging to the fixed pattern and the ones belonging to the format region and the version are not numbered, so the numbering scheme has to avoid them. Basically, the numbering starts at the lower right, chooses next its left neighbour, and proceeds alternately up and down columns of width 2, choosing as its next module the first available according to the avoidance rules:



Groups of eight modules are arranged into bytes. This grouping plays a role only in case error correction is necessary, which it is not here. (The bytes are interpreted as elements in the Galois field of order 256 in the BCH codes.) In the following diagram, the original data bits are black and grouped in bytes.



Finally, we see that the original data string is equal to $$ \matrix { 01000001 & 00110111 & 00000110 & 10010011 \cr 11010011 & 00110010 & 11100011 & 00010011 \cr 01000011 & 00010011 & 01010011 & 10010011 \cr 00100011 & 01100011 & 01010011 & 00110011 \cr 01010011 & 10000011 & 10010011 & 01110011 \cr 10010000 & 11101100 & 00010001 & 11101100 \cr 00010001 & 11101100 & 00010001 & 11101100 \cr 00010001 & 11101100 & 00010001 & 11101100 \cr 00010001 & 11101100 & 00010001 & 00001101 \cr 11001100 & 11000011 & 11011100 & 01100010 \cr 01011001 & 00011100 & 10011001 & 01111111 \cr } $$ I have arranged the bits into bytes for ease of reading, even though it is only the uninterrupted bit string that is relevant. **What do we do with these data bits?**

Comments?

Interpreting the data string

The first few bits of any QR data tell you what mode to be used for data interpretation. Here these first few bits are 0100, which tells us that we should interpret according to the Japanese version of ASCII, also known as JIS8. According to this convention, the characters are groups of 8 bits. In this mode, the next 8 bits tell us the length of the true message. Here those 8 are 00010011, which is the binary expression for 19. This means that the next 19 groups of 8 bits make up the message. Thus we should see the initial part of the message, not in terms of the bytes of the symbol, but as $$ \matrix { 0100 & 00010011 \cr & & 01110000 & 01101001 \cr 00111101 & 00110011 & 00101110 & 00110001 \cr 00110100 & 00110001 & 00110101 & 00111001 \cr 00110010 & 00110110 & 00110101 & 00110011 \cr 00110101 & 00111000 & 00111001 & 00110111 \cr 00111001 & \cr } $$ The data then next has some terminal padding to fill out a symbol byte, and then several more very simple padding bytes to fill up the designated message allotment: $$ \matrix {

0000 & 11101100 & 00010001 & 11101100 \cr 00010001 & 11101100 & 00010001 & 11101100 \cr 00010001 & 11101100 & 00010001 & 11101100 \cr 00010001 & 11101100 \cr } $$ The rest of the data bits are for error correction: $$ \matrix { & & 00010001 & 00001101 \cr 11001100 & 11000011 & 11011100 & 01100010 \cr 01011001 & 00011100 & 10011001 & 01111111 \cr } $$ The real message is therefore the sequence of bytes $$ \matrix { & 01110000 & 01101001 & 00111101 & 00110011 \cr & 00101110 & 00110001 & 00110100 & 00110001 \cr & 00110101 & 00111001 & 00110010 & 00110110 \cr & 00110101 & 00110011 & 00110101 & 00111000 \cr & 00111001 & 00110111 & 00111001 & \cr } $$ This is ASCII. Thus the first character represents 'p', the second 'i', and so on. Eventually you can see that the (admittingly unexciting) message is $$ {\rm pi} = 3.14159265358979 $$ The basic idea for interpreting any QR symbol is the same, except that large ones are broken up into interleaved blocks for purposes of error correction.

Where to read more

Web sites about the applications of QR code

- QR for bus schedules
- QR on tombstones
- QR for parking meters
- More QR parking
- QR can find your parked car
- QR on menus
- Douglas Coupland's QR art
- Other stupid uses of QR

Technical information

- The official web site for QR
  Not a technical specification.

- The Wikipedia article on QR code
  Also not a full technical specification, although some technical information is presented in some rather obscure images.

- An earlier Feature Column on data matrices.
  The format of the 2D code used in the scheme discussed in this earlier feature is designed to be read in a controlled environment, such as scanning postage stamps in a post office. They are used much by postal authorities in Western countries, such as the United States, Canada, and Germany. Most have data that is itself truly encrypted, in order to avoid counterfeiting. QR code symbols are more robust, and seem to be rarely encrypted.

- The ISO specification document on QR code, available unfortunately only for a sizeable fee of 210 Swiss francs. From time to time someone posts a copy for which you do not have to pay, but this is surely a violation of copyright law.

- Carolyn Eby's web site on QR code.
  This is an extremely valuable resource. It will tell you lots of details I have not had space to cover, including an extensive if occasionally tedious discussion of error correction in QR code.

  It has also a QR code generator, which will produce a QR code symbol from input you provide. I have used this feature extensively in producing this column.
- W. Wesley Peterson and E. J. Weldon, Jr., **Error-correcting Codes**, Second edition, M.I.T. Press, 1972.
  I haven't said how to use the redundant EC components of the data to correct damaged symbols. It is not elementary mathematics, but it is standard. The literature is huge, and whether any given reference is satisfactory or not depends on your mathematical level. I have found this book both clear and succinct, if a bit advanced. The very code used in the format word of length 15 is covered in the example on p. 210 (section 8.1), section 9.2 (especially p. 276), and in section 9.4 (especially pp. 286-288).

Bill Casselman
University of British Columbia, Vancouver, Canada
Email Bill Casselman

Comments (10)

Cool

hey this is really cool. It's really complicated though.

#1 - :) - 01/12/2015 - 17:08

WOW!

Thank you. This was just what I was hoping for -- well, one of two things. The other is a site where I can upload a QR code and find out what it says.

What an amazingly complicated system.

#2 - Sean L - 07/14/2015 - 16:30

Wow it is amazing . Superb

#3 - Massih - 08/07/2015 - 09:15

I'm a bit confused by the Interpreting section in how the bytes are relatin b to the full sequence. (Sorry I'm new at this). I think I've figured it out, some four bit sequences are being flipped? If so why?

#4 - Curious - 08/11/2015 - 11:56

I need a bit more information about what you are asking.

#5 - Bill Casselman - 08/12/2015 - 14:52

This is really awesome. There is not much info on the Internet about learning how to read these QR codes. This article is so cool. Would love to see further about it. Thanks a lot !!

#6 - Enric - 01/29/2016 - 05:40

Great

This is what I wanted to see. Thank you, editor, for making this great article!

#7 - Param Siddharth - 03/15/2016 - 05:44

Unique

I was finding these answers for a very long time and this site has given it all. Thanks for the wonderful explanation it's a bit complicated but it is the only one understandable.

#8 - Neelansh - 08/12/2016 - 23:46

Arigato

Ohhh it's so... Cooooool!
So pretty!
??

#9 - Leppryt - 11/15/2016 - 03:03

Appreciative

TY for this. Small error though... I believe. The last 8 or 9 blocks maybe wrong. The last block is the remainder bc it should have 7 bits. The pixel starting from lower left (9,8) should always be dark... I think. Called the dark module?

#10 - Randy - 12/23/2016 - 15:43

Name

E-mail (Will not appear online)

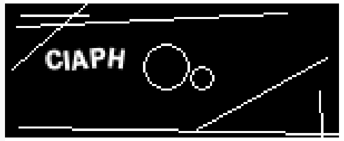
Homepage

http://

Title

Comment

To prevent automated Bots from spamming, please enter the text you see in the image below in the appropriate input box. Your comment will only be submitted if the strings match. Please ensure that your browser supports and accepts cookies, or your comment cannot be verified correctly.



»

Submit Comment

This comment form is powered by **GentleSource** **Comment Script**. It can be included in PHP or HTML files and allows visitors to leave comments on the website.