File2QR, an Android Application to Encode Files in QR Codes

Francesco Feltrinelli

Email: <francesco.feltrinelli@gmail.com>
Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milan, IT

Abstract

In this paper File2QR is introduced, which is an Android application that encodes any small file to a QR Code, and vice versa, decodes a file previously encoded in a QR. To encode, a file is loaded from the filesystem; to decode, the QR image can be loaded from filesystem or acquired through the built-in camera of the mobile phone. The ZXing library has been used and slightly modified to let it encode binary data instead of only text. The encode procedure is a custom proposal, as there is no standard up to now that encompasses the encoding of binary data.

1 Overview

QR Codes, which are a technology relatively old, are now beginning to spread in several scenarios across the end-user market, thanks to the improvements of mobile phones' cameras and the growth of the computational power of smartphones. Libraries and applications to decode/encode text to QRs with the camera were developed for various mobile platforms, like Android, Symbian, Blackberry and iPhone. None of these however, to the best of our knowledge, allows the encoding of arbitrary binary data to QR Codes, and this is because the standards released till now by ISO (International Organization for Standardization) and JIS (Japanese Industrial Standards) do not plan it [1]. This is the reason why File2QR was developed.

In section 2 QR codes will be introduced, while section 3 is dedicated to ZXing, an Android library to decode/encode them. Eventually in section 4 File2QR will be discussed. Finally, conclusions will be outlined in section 5.

2 QR Codes

A QR Code™ [1] (Quick Response Code) is a type of 2D bar code originally developed by the japanese automotive components manufacturer Denso Wave in 1994. As every 2D codes, it contains information in both the vertical and horizontal directions, whereas a bar code contains data in one direction only: as a consequence, at equal printout area, it holds a considerably greater volume of information than a bar code. It can encode any kind of textual information, which can later be decoded by dedicated scanners or by mobile phone's camera using one of the many existing applications. An example of QR Code is in figure 1.

[1] registered trademark of Denso Wave, Inc.

Figure 1: Example of QR Code

2.1 Structure

A QR code is made of a matrix of black squares, or modules, arranged in a squared pattern on a white background: these modules can be either data codewords (each module is a bit of information) or error correction codewords.

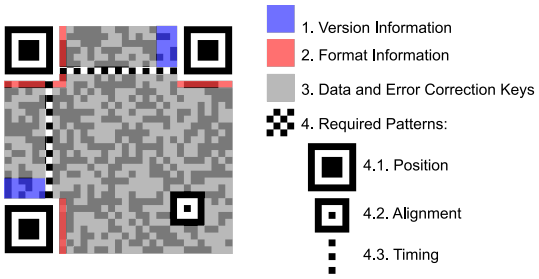

Figure 2: Structure of a QR Code [6]

As you can see in figure 2, Position-detection patterns are placed around three of the four corners to help the detection of the QR (from any direction). Timing patterns, which are lines composed by alternating black and white

modules and connecting two Position patterns, are used to help determine a symbol's coordinate. Alignment patterns are used to correct skewness of the code. Some modules around the Position patterns store Version (see section 2.4) and Format (Error Correction level and Mask Pattern, see section 2.3) information. A white margin, or "clear area", surrounds the code. [1] [2] [4] [5]

2.2 Character Encoding

A QR Code can store only text. Four types of characters can be encoded:

- **numeric**: 10 numeric digits (0-9). Normal data density of 10 bits per 3 characters.
- **alphanumeric**: 10 numeric digits (0-9), 26 alphabetic characters (A-Z), and 9 symbols (SP, $, %, *, +, -, ., /, :). Normal data density of 11 bits per 2 characters.
- **8-bit byte**: the 8-bit Latin/Kana character set. Data density of 8 bits per character.
- **Kanji**: Kanji characters. Each two-byte character value is compacted to a 13 bit binary codeword.

Note that the 8-bit byte mode has a misleading name, because it does not refer to the encoding of arbitrary binary data, but is just another form of character encoding.

2.3 Error Correction

QR codes use an Error Correction Code (ECC) mechanism based on Reed-Solomon code, which allows to restore data if the code is dirty or damaged, up to a certain percentage. Four levels of error correction are defined:

- **Level L**: approx. 7% correction rate
- **Level M**: approx. 15% correction rate
- **Level Q**: approx. 25% correction rate
- **Level H**: approx. 30% correction rate

Note that the percentages above refer to data restoration rate for *total* number of codewords (where "codeword" is a unit of information equal to 8 bits). As Reed-Solomon Code requires twice the amount of codewords to be corrected, the percentage of only data codewords corrected is higher. For example, if it is needed to correct 25 of 100 codewords, there should be $25 \times 2 = 50$ correction codewords, so the correction rate as displayed above would be $25/(100 + 50) \simeq 16.7\%$. [3]

Raising correction level improves reliability, but increases its size (stored information being equal).

2.4 Version

A QR code has a symbol Version ranging from 1 to 40, where each version refers to a different number of modules used, starting with Version 1 (21 x 21 modules) up to Version 40 (177 x 177 modules): each version number has 4 more modules than previous version (see figure 3). Consequently, each version for each error correction level has a different capacity: from the 72 bits (or in characters: 17 numeric, 10 alphanumeric, 7 binary and 4 Kanji) of Version 1 with ECC level H, to the 23648 bits (in characters: 7089 numeric , 4296 alphanumeric, 2953 binary and 1817 Kanji) of Version 40 with ECC level L. [3]

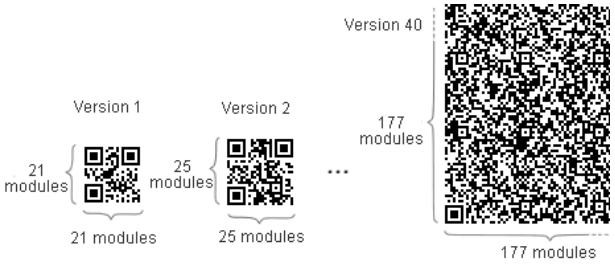

Figure 3: Versions of a QR Code [3]

Once a Version is choosen, the printout size of the QR depends on the module size. The larger the module is, the more stable and easier to read with a QR code scanner it becomes. On the other hand, as the QR gets larger, a larger printing area is required. Other constraints are the printer head density if the QR is printed, or the monitor resolution if it is displayed on a screen, which determine the minimum viewable module size, and the scanner resolution, which determine the minimum module detectable size.

2.5 Standards

There are several standards covering the physical encoding of a QR Code, released mainly by ISO and JIS [1]. At the application layer, there is some variation between implementations. NTT DoCoMo, the predominant mobile phone operator in Japan, has established de facto standards for the encoding of URLs, contact information, bookmarks and several other data types [7]. There exist several applications and libraries for the majority of mobile operating systems, including Android, Symbian, Blackberry and iPhone.

As a QR can encode any kind of textual information, a lot of usage scenarios were proposed, and many of them are gaining more and more popularity: encoding of URLs, contact information, SMS, geographic information, Wifi network configuration, calendar events, and so on. Of course, a QR reader must identify the type of the content, and this is done prefixing the text with known tags,

like for example *http://* or *URLTO:* for URLs, *mailto:* or *MATMSG:* for email addresses, *tel:* for telephone numbers and so on [9]. Instead, as we will see, QRs have never been used to encode arbitrary binary data, because standards always refer to textual information. File2QR proposes a way to do that.

3 ZXing

ZXing is an open-source, multi-format 1D/2D barcode image processing library implemented in Java. The purpose of the library is to use the built-in camera on mobile phones to photograph and decode barcodes on the device. Many are the supported code formats, including QR Codes, and the library is available on a variety of platforms, among which Android. An Android scanner application, Barcode Scanner, is also provided [8].

3.1 Integration

Integration of a custom Android application with Barcode Scanner to decode/encode QR codes is possible through the use of Intents, either manually creating the code to launch the appropriate Intents or exploiting the provided integration class IntentIntegrator, which encapsulates some of the details of launching Intents. Another possibility is to start web-browser navigation to a special type of url, which Barcode Scanner is registered to, and which serves as an hook to start it [10]. Unfortunately, in our case integration through Intents or via URLs was not sufficient.

3.2 Binary Encoding

ZXing strictly complies with QR code standards, so it gives the possibility to encode Strings, which content must belong to one of the four categories: numeric, alphanumeric, 8-bit bytes and Kanji. As stated in section 2.2, the 8-bit byte mode is not really a binary data encoding, but just another form of character encoding, and ZXing respects this standard [11]. Therefore, there is no natural way to encode arbitrary binary data, even if obviously what is put in the QR in the end are raw bytes. But the standards does not plan this, and the encoded bytes are always in some ways internally manipulated, basing on the assumption of textual information.

A solution to this could be to encode the String representation of the bytes, for example in decimal format, or more efficiently in higher bases like hexadecimal, base-64 or base-128. Still this would be inefficient because, (optimistically) assuming to use the default ISO/IEC 8859-1encoding, which encodes each character with a byte, with base-64 6 bits would be represented with each character ($64 = 2^6$) and 7 bits with base-128 ($128 = 2^7$), which is less than the 8 bits used to memorize that character. Moreover, the use of base-256 would not be possible, as in ISO/IEC 8859-1not all of the 256 codes are displayable characters.

For these reasons, it was decided to integrate part of the source code of ZXing to adapt it to directly accept bytes, bypassing any preliminary manipulation in encoding phase. Similarly, it was written some code to directly extract raw bytes from a QR, bypassing all post-processing of presumed textual info just decoded.

4 File2QR

File2QR is an Android application that let the user convert an arbitrary file to and from a QR Code, exploiting the built-in camera of the mobile phone. It uses ZXing library and reimplement some parts of it. It is open-source software, released under Apache License version 2.0 as ZXing is, and its source code is freely downloadable from SourceForge [2]; end-users can download it for free from the Android Market. The minimum required Android version is 2.0, while the mobile itself must be provided with a camera with autofocus capability.

4.1 Features

The application has two main parts: the "decode" section and the "encode" one (see figure 4). In the *encode* section the user loads a file from filesystem (through *OpenIntents OI FileManager* [3], which should be installed in the system), and this is encoded to a QR and displayed on screen, to possibly let another user decode it with his mobile. The generated QRs are also saved on external storage and listed in the application window, from which the user can manipulate them, that is, open again, delete, rename and share them (for example, send them via bluetooth or with an email, post them to Facebook or Flickr, and so on), and view some info.

[2] The project page is at `http://sourceforge.net/projects/file2qr/`

[3] *OI File Manager* is an open file manager for Android: `http://www.openintents.org/en/filemanager`

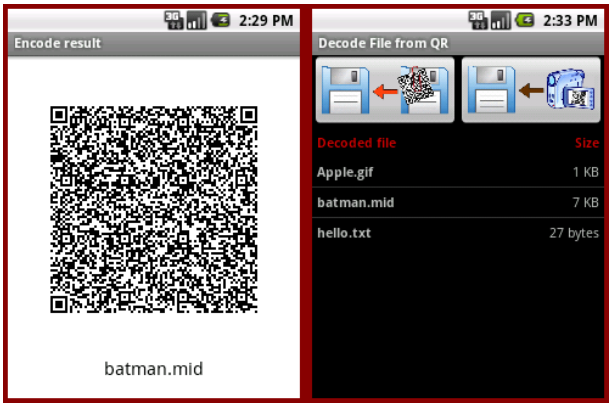

Figure 4: File2QR screenshots. On the left, an encoded QR; on the right, the decode section.

The *decode* section is dedicated to decode a file previously encoded in a QR. This can be done either loading the QR image from the filesystem or through the camera. In the second case the user points the camera lens to the QR, taking care of including it completely in the screen, and waits for it being decoded in real time; meanwhile, the camera periodically autofocus. When the QR is decoded, a dialog alerts the user and the decoded file is saved on the external storage. If the QR is actually a standard textual one, its text is displayed instead. No error messages are displayed if no QR can be decoded correctly: the application simply indefinitely try to decode something, and when the user wants to abort he taps the screen. Like for encoded QRs, also the decoded files are listed and can be manipulated in the same ways.

4.2 Encode Procedure

As a QR can hold nominally a maximum of 23648 bits (about 2.9 KB) with Version 40 and lowest error correction ([3]), files encoded by File2QR are necessarily small. To lessen this problem, during the encode procedure the body of the encoded file is gzip-compressed; because of this, the maximum (uncompressed) file size encodable in a QR depends on how well the file reacts to compression, but anyway should be in the order of a few KB. In the following the encode procedure is detailed (see also figure 5).

Both the filename of an encoded file and its binary content are saved in the QR. A textual part that identifies the file, called header, is followed by the gzip-compression of the bytes it contains, called body. The header is composed by the fixed tag **FILE:** , followed by the filename, followed by the fixed separator **NUL** (the null character, code 0). The filename is just the name of the file, excluding its path and including its extension; to save space, it must have a maximum lenght of 20 characters, so if it is longer than 20 characters only the first 20–1 are kept and the truncating character ˜ (tilde) is put as 20th, followed by the extension which is never truncated. The character encoding used for the header is ISO/IEC 8859-1, so only Latin1 - Western European characters should be included in the filename. The body bytes are simply the gzip-compression of the bytes contained in the file. Then the header bytes and the body bytes are merged in a single array of bytes which is the one to be finally encoded in the QR. The adopted error correction level is L.

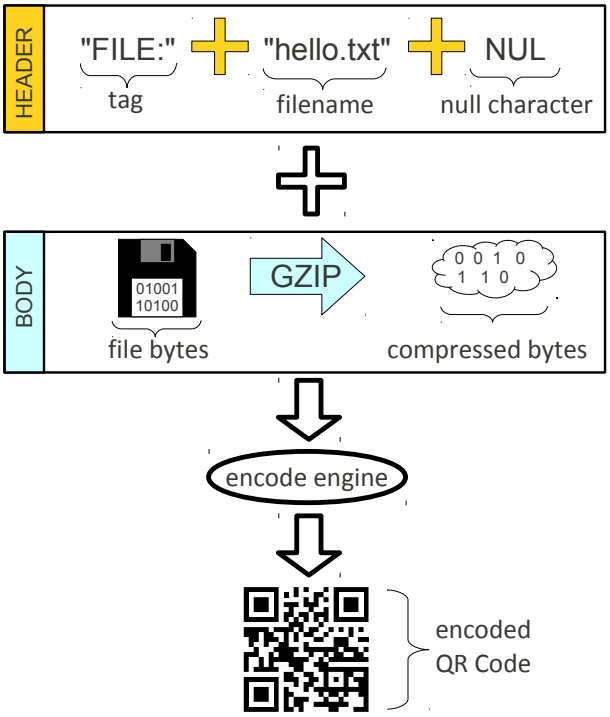

Figure 5: File2QR encode procedure.

To interpret the raw bytes decoded from the QR the reverse procedure is used. First of all the raw bytes are read as a String with ISO/IEC 8859-1. If the file tag is not found at the beginning, the QR does not encode a file and it is probably a standard textual one, so its textual content is simply displayed to user. On the contrary, if the file tag is found, the characters from the end of the tag till the first found occurence of the separator character (exluded) are the filename. The bytes following the separator character are the gzip-compression of the content of the file, which must be uncompressed. Once the filename and the content of the file are collected, the corresponding decoded file can be saved to storage.

4.3 Usage Scenarios

Due to the small size of encodable files, usage scenarios are obviously reduced. Nevertheless, there are various situations in which encoding of a small file instead of merely textual information could be interesting.

A nice example is the encoding of small MIDI files. They could be advertising-related melodies, used for marketing purposes and displayed on websites or printed on walls, buildings, public transports and potentially everywhere ads are already placed on. They could remind the user a nice tune already listened on television/radio spots, making him remember about the sponsored product. They could also be ringtones for mobile phones. Or they could be used for musical television games where the interaction with audience at home is desired. As a last example, they could be printed on gadgets or business cards as "musical brand" of a company. This kind of encoding should be particularly attractive from the user's point of view because it would implicitly map the information from one sensorial domain to another, that is, from sight to hearing and vice versa.

Another reason to encode binary files in a QR instead of text could be because a binary format for a custom application already exists, and it is used by an already established legacy infrastructure. If the existing scenario could be profitably extended with the use of QR codes, than probably it would be easier to use directly the binary format instead of translating it in an intermediary textual representation. Moreover, the binary format would probably be more compact than the textual equivalent. An example could be the usage of small binary configuration files for some kind of machinery, maybe where the machine itself can decode on the fly its configuration reading the QR code.

4.4 Source Code

Every Android application is made of Java classes, of XML configuration files, of resource files like texts, images and sounds, and optionally of .jar libraries linked to the application. The Java source code is contained in standard Java packages inside the `src/` directory, while resource/configuration files are inside `res/` folder: there is a subfolder for each kind of resource files, like `res/drawables/` for images, `res/layout/` for XML files describing the layout of each Activity, `res/values/` for texts and other values, and so on. The application general configuration file, `AndroidManifest.xml`, is placed in the top folder. Included libraries could be placed anywhere, for example in a `lib/` folder.

The Java source code of File2QR is organized in four packages (the prefix *feltrinelli.project.android.file2qr* is omitted). In the following some details about each package is given.

4.4.1 *activity* package

It contains all the Activity classes, which are responsible for the interaction with the user, display of UI elements and exploitation of the decode/encode engine classes. Its classes are:

- ***MainActivity***: the first started, it lets the user choose whether to go to the decode section (*DecodedListActivity*) or the encode one (*EncodedListActivity*)
- ***DecodedListActivity***: displays a list of the decoded files, which the user can manipulate in several ways. It lets the user start the decoding of a file from a QR either by loading an image from filesystem (*OI FileManager* activities) or through the camera (*CameraDecodeActivity*). The decoded file is added to the list.
- ***CameraDecodeActivity***: it lets the user decode a QR acquiring it from the camera input, which is displayed on screen so that the user can direct camera lens towards it. If a file is decoded, it is reported back to (*DecodedListActivity*).
- ***EncodedListActivity***: displays a list of the encoded QR images, which the user can manipulate in several ways. It lets the user load a file from filesystem (*OI FileManager* activities) and start the encoding to QR (*EncodeActivity*). The encode QR image is added to the list.
- ***EncodeActivity***: encodes the given file to a QR. If the encoding is successful, the QR is shown and also saved to storage.

4.4.2 *engine* package

It contains the decode/encode engine classes. Its main classes are:

- ***FileFromQrDecoder***: gets the raw bytes from a QR found in an image and interprets them according to the procedure described in 4.2 to get the file encoded in it.
- ***FileToQrEncoder***: reads the given file and returns the Bitmap of the QR in which it is encoded, according to the procedure described in 4.2. It uses a *QrByteWriter* to get the *ByteMatrix* containing the bytes of the bitmap of the QR, and convert this matrix to a *Bitmap*.
- ***QrByteWriter***: returns a *ByteMatrix* of greyscale values which are the bytes of the bitmap of the QR. It uses a *QrByteEncoder* to get a *QRCode* object; the *QRCode* contains a *ByteMatrix* in which the color of each module has value 0-1 (0=white, 1=black), so it must be converted to a *ByteMatrix* in which color range is 0-255 as in greyscale (0=black, 255=white), scaling also the QR with a factor of 4 to fit the screen, and adding the white margins.
- ***QrByteEncoder***: returns a *ByteMatrix* of black/white values. It works out the encoding as described in QR standards.

4.4.3 *manager* package

Managers are a way to unify common operations done by activities within a single module. Currently there are two implemented managers, *FileManager* and *NotificationManager*:

- ***FileManager***: used by every Activity to operate on files.
- ***NotificationManager***: used by every Activity to show various kind of notifications to the user.

4.4.4 *utility* package

It contains various utility classes, like for example:

- ***ByteUtilities***: it provides methods to compress, decompress and merge arrays of bytes.
- ***FolderCursor***: it extends *Cursor* to provide the abstraction of a database-like access to the files of a specified folder.

5 Conclusions

In this paper File2QR, an Android application that lets the user decode/encode any small file from/to QR Codes, was introduced. This is, to the best of our knowledge, the first public application that encodes arbitrary binary data in a QR Code, because there are no standards for this. File2QR proposes an encoding procedure.

The QR Code world is quickly developing, making closer and closer to reality the concept of World of Things, or Object Hyperlinking. Usage scenarios will be more and more common in the next future, and hopefully File2QR will have given its small contribution to this.

References

[1] ISO/IEC: *Information technology - Automatic identification and data capture techniques - QR Code 2005 bar code symbology specification*, ISO/IEC 18004:2006

[2] Denso Wave: *About QR Code*, Retrieved 21 Sep. 2010, <http://www.denso-wave.com/qrcode/aboutqr-e.html>

[3] Denso Wave: *Symbol Version*, Retrieved 21 Sep. 2010, <http://www.denso-wave.com/qrcode/qrgene2-e.html>

[4] Barcode Reader Resource Center: *What is QR Code?*, Retrieved 21 Sep. 2010, <http://www.keyence.com/barcode/technology/barcode_2d_qr_code.php>

[5] Y.Swetake: *How to create QRcode*, Retrieved 21 Sep. 2010, <http://www.swetake.com/qr/qr1_en.html>

[6] Richard Wheeler: *Example of QR code, highlighting functional elements*, Retrieved 21 Sep. 2010, <http://en.wikipedia.org/wiki/File:QR_Code_Structure_Example.svg>

[7] NTT DoCoMo: *Bar Code, Outline of Functions*, Retrieved 21 Sep. 2010, <http://www.nttdocomo.co.jp/english/service/imode/make/content/barcode/function/index.html>

[8] ZXing: *ZXing ("Zebra Crossing")*, Retrieved 21 Sep. 2010, <http://code.google.com/p/zxing/>

[9] ZXing: *Barcode Contents*, Retrieved 21 Sep. 2010, <http://code.google.com/p/zxing/wiki/BarcodeContents>

[10] ZXing: *How to scan a barcode from another Android application via Intents*, Retrieved 21 Sep. 2010, <http://code.google.com/p/zxing/wiki/ScanningViaIntent>

[11] ZXing: *QRCode for binaries data*, Retrieved 21 Sep. 2010, <http://code.google.com/p/zxing/issues/detail?id=382>