Table of Contents

1 Introduction

This document defines the IQRF Code. IQRF Code can effectively store various IQRF specific values in a human-readable alphanumeric text format that can be then transmitted in various ways using various media. For instance, IQRF Code called Smart Connect Code containing concrete IQRF values is used for Smart Connect bonding.

2 IQRF Values

The following IQRF values can be stored at IQRF Code. All numeric values are stored using big-endian style. The order of IQRF values in IQRF Code is not defined.

2.1 MID

ID=1. Module Identification is a 4 bytes long unique identification of IQRF transceiver. MID is stored in the IQRF transceiver during its production process and it cannot be changed later.
This value indispensable for the Smart Connect bonding.

2.2 IBK

ID=2. Individual Bonding Key is a 16 bytes long randomly generated key (list of bytes) used to bond IQRF node during Smart Connect process. IBK is stored in the IQRF transceiver during its production process and it cannot be changed later.
This value indispensable for the Smart Connect bonding.

2.3 HWPID

ID=3. Hardware Profile ID is a 2 bytes long unique identification of a product type using IQRF transceiver. HWPID is provided by the Custom DPA Handler code by the product manufacturer.
This value is highly recommended for the Smart Connect bonding as it allows to identify the type of the product to be bonded.

2.4 Bonding Channel [obsolete]

ID=4. This value is not used anymore.

3 IQRF Code Encoding

IQRF Code encoding i.e. algorithm to convert IQRF values into text consists of 3 steps.

3.1 Conversion to Bytes

In this step, IQRF values are converted into an array of bytes. Because the smallest piece stored in the array of bytes is one nibble it is actually a nibble stream. Every IQRF value is first introduced by its ID stored in a nibble. Then actual bytes (from LSB to MSB) follow. Because a previous nibble can divide byte into halves the lower nibble of the following byte is stored in the higher nibble of the divided byte and the higher nibble is stored in the lower nibble of the following byte. When all IQRF values are stored one after another a final zero nibble is stored to label end of the stream.

Example: HWPID=0xABCD

byte index	0		1		2	
description	low nibble 0xAB	HWPID ID=3	low nibble 0xCD	high nibble 0xAB	End ID=0	high nibble 0xCD
nibbles	B	3	D	A	0	C

Result = B3 DA 0C

3.2 base57 Conversion

The array of bytes from the previous step is converted into alphanumeric text. Bytes are divided into 8 bytes long pieces from its beginning. Because the total number of bytes is not always divisible of 8, the last piece can be long from 1 to 8 bytes. Every piece as 8 bytes long big-endian unsigned integer value is then converted into base 57 number where every 57 base digit corresponds to the character index at the base57 alphabet shown below. In this case least significant base 57 characters are added to the text first. base57 alphabet contains digits and uppercase & lowercase letters except for these 5 ambiguous characters: `0 I O l u`:

`123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstvwxyz`

The following table show number of base 57 characters needed to store a certain number of bytes:

number of base57 characters	*number of bits*	*number of bits rounded*	*number of bytes*
2	11.67	8	1
3	17.50	16	2
5	29.16	24	3
6	35.00	32	4
7	40.83	40	5
9	52.50	48	6
10	58.33	56	7
11	64.16	64	8

Example: B3 DA 0C

Piece = 0xB3DA0C

step	*piece*	*mod 57= base57 character value*	*base57 character*
1	`0xB3DA0C`	19	L
2	`0x0327C1`	46	o
3	`0x000E2B`	36	d
4	`0x00003F`	6	7
5	`0x000001`	1	2

Result = Lod72

3.3 Adding Check Character

In this step, a check character is added at the end of the text to validate it and to protect it against accidental errors. The check character is computed using *Luhn mod N algorithm* (see https://en.wikipedia.org/wiki/Luhn_mod_N_algorithm).

Example: Lod72

character index	*character*	*base57 value*	*factor*	*addend*	*sum digits*	*sum*
4	2	1	2	2	2	2
3	7	6	1	6	6	8
2	d	36	2	72	16	24
1	o	46	1	46	46	70
0	L	19	2	38	38	108

sum = 108
Check character value = (57 - (108 mod 57)) mod 57 = 6
Check character = 7

Result = Lod727

4 IQRF Code Decoding

Decoding algorithm is inverse to encoding. Encoding steps must be executed in a reverse way from the last to the first.

5 IQRFcode.exe

IQRFcode.exe command line utility provides IQRF Code encoding and decoding. The C# source code is available, thus the above algorithms can be easily migrated to different programming languages and operating systems.

- *Encoding example*

```
IQRFcode.exe encode -MID:12345678 -IBK:00112233445566778899AABBCCDDEEFF -HWPID:AABB
```

Output: `42rfRrBCHc7zLq2SZrdcCBkTv4wwaHbNeP`

- *Decoding example*

```
IQRFcode.exe decode -Code:42rfRrBCHc7zLq2SZrdcCBkTv4wwaHbNeP
```

Output: `-MID:12345678 -IBK:00112233445566778899AABBCCDDEEFF -HWPID:AABB`

6 IQRF_IDE_Command.exe

IQRF_IDE_Command.exe is a command line version of the IQRF IDE that can be used for IQRF Code encoding/decoding and also for generating QR Code picture file.

- *Encoding example*

```
IQRF_IDE_Command.exe IQRFencode /IBK=00112233445566778899AABBCCDDEEFF /HwpId=AABB
/MID=12345678 /QRcode=QRcode.png
```

Output:

```
 IQRF Code: 42rfRrBCHc7zLq2SZrdcCBkTv4wwaHbNeP

 QR code file: QRcode_12345678.png
 Size:         37 x 37 (29 x 29) modules
 Total size:   37 x 37 pixels
```

- *Decoding example*

```
IQRF_IDE_Command.exe IQRFdecode /Code=42rfRrBCHc7zLq2SZrdcCBkTv4wwaHbNeP
```

Output:

```
 MID:    12345678
 IBK:    00112233445566778899AABBCCDDEEFF
 HWPID:  AABB
```

7 Generating Smart Connect QR Code

QR Code is a common medium to store Smart Connect Code used for Smart Connect bonding. The QR Code is typically stuck on the back of the product's body and later processed by the mobile APP. The process of generating the QR Code consists of three steps.

7.1 Obtaining IQRF Values

There are three IQRF values needed to generate Smart Connect QR Code. Two of them (MID and IBK) are unique for every IQRF transceiver while HWPID is fixed for the specified product and is independent on the contained physical IQRF transceiver. So the task is to read out the MID and IBK from the IQRF transceiver that is part of the product.

If the SPI signals of the IQRF transceiver are accessible then the best way is to use SPI bus for reading out the values. If the IQRF transceiver can be physically inserted into CK-USB-04A then IQRF IDE can readout the values. Either use *USB Device/Show TR Module Info* (Ctrl+M) or *Tools/IQRF Code Tools* (Ctrl+Alt+S). When IQRF transceiver cannot be inserted into CK-SUB-04A then it is often possible to connect SPI bus with CK-USB-04A by separate wiring. Another way is to read out the value by a custom device according to the IQRF SPI specification or to use IQRF_IDE_Command.exe.

If the SPI bus is not available then DPA Service Mode at IQRF IDE can be used to read out the values supposed DPA plug-in is already uploaded at the IQRF transceiver. Please follow the IQRF IDE documentation and then go to *Tools/CATS Service Tools/DPA Service Mode.*

7.2 Encoding IQRF Code

When MID, IBK, and HWPID values are ready then the process IQRF Code Encoding generates the IQRF Code. Either use IQRF IDE via *Tools/IQRF Code Tools* or use command line utilities described above.

7.3 Generating QR Code Image

The last step is to generate the QR Code image containing the encoded IQRF Code. Again, the *Tools/IQRF Code Tools* (Ctrl+Alt+S) or IQRF_IDE_Command.exe will do the job. Alternatively, there are dozens of either online or command line tools to generate QR Codes for the given text (IQRF Code in our case). Please pay attention to the sufficient image size, resolution, print quality, print material quality, its adhesiveness, and the location of the QR Code at the product.