Table of Contents

1 Introduction 1
2 IQRF Values 1
2.1 MID 1
2.2 IBK 1
2.3 HWPID 1
2.4 Bonding Channel 1
3 IQRF Code Encoding 1
3.1 Conversion to Bytes 1
3.2 base57 Conversion 2
3.3 Adding Check Character 2
4 IQRF Code Decoding 3
5 IQRFcode.exe 3

1 Introduction

This document defines 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 value 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.

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.

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.

2.4 Bonding Channel

ID=4. Bonding Channel is a 1 byte value storing RF channel value used during bonding IQRF node. The value is stored at the transceiver configuration memory and it can be configured by the product manufacturer.

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 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 byte 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 these 5 characters: `0 I O l u`:

```
123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstvwxyz
```

The following table show number of base 57 characters needed to store 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 reverse way from the last to the first.

5 IQRFcode.exe

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

```
IQRFcode.exe encode -MID:12345678 -IBK:00112233445566778899AABBCCDDEEFF -HwpId:AABB -BondChan:10
```

Output: 42rfRrBCHc7zLq2SZrdcCBsUv4wwaHbNevm1L

```
IQRFcode.exe decode -Code:42rfRrBCHc7zLq2SZrdcCBsUv4wwaHbNevm1L
```

Output: -MID:12345678 -IBK:00112233445566778899AABBCCDDEEFF -HWPID:AABB -BondChan:10