BY 公版三模键盘驱动协议

(版本 v2.22)

版本信息:

版本	日期	作者	更新说明
2.20	2022-05-23	卿	增加备注信息，说明 Report ID 06/09 的差别， 及读取配置的限制说明，用红字提醒。
2.21	2022-06-07	卿	增加读 FW 版本命令，以便 web HID 下获取版本
2.22	2022-06-14	卿	修改读 FW 版本命令

特别说明：USB 模式下 RE ID 为 0x06/0x09，具体以键盘为主

8805 系列，驱动读取仅支持<读取键盘密码或者宏存储空间大小><读配置数据（profile 表）>

有线（USB）模式

驱动与 USB 设备之间的数据传输通道和流程

与驱动相关的的数据传输端口只有 2 个：

1.冒泡数据端口（用于键盘主动通知驱动）

2. 配置数据读写端口

冒泡数据端口：

主动上报一些特定信息，命令 ID 是 0A 时，代表这是键盘通过 EP 端口主动上报的数据，数据格式如下(XX 是冒泡接口的 reportid)

XX 0A idval00 00 00 00

解释说明如下：

RE ID	命令 ID	ID	Val	预留	预留	预留	预留	说明
0x09	0x0A	0x05	Val	XX	00	00	00	功能：上报电池电量及状态 Val：当前电池剩余电量（0-100） XX：（高 4bit）1=充电中 0=未充电 （低 4bit）1=充满 0=未充满 例： 未充电电池电量 75: 09 0A 05 4B 00 00 00 00 充电电池电量 75： 09 0A 05 4B 10 00 00 00 充电电池电量 100： 09 0A 05 64 11 00 00 00
0x09	0x0A	0x06	Val	XX	CRC	00	00	功能：驱动下发屏幕动图时，主动回报当前数据是否处理完成。 Val：1=处理成功 0=处理失败，驱动重发 XX：驱动下发的包序号 CRC:设备计算的 CRC 0xFF&(Byte8+…+ByteN) 例：驱动第一组动图的第二帧图片的第三包数据 驱动发 set_report 命令 09 0c 00 01 02 CRC(计算结果) 00 02

								XX(512byte 的图片数据) 键盘计算 CRC 与接收到 CRC 做比较，如果一样，则发送给屏幕 IC，发送完成之后，冒泡给驱动 09 0a 06 01 01 crc 00 00 发送失败或者 CRC 对比不一样回复： 09 0a 06 00 01 crc 00 00
0x09	0x0A	07	Val	00	00	00	00	功能：键盘切换系统（Win/Mac） Val：1=切换到 Mac 0=切换到 Windows 例： 切换到 Mac： 09 0A 07 01 00 00 00 00 切换到 Windows： 09 0A 07 00 00 00 00 00
0x09	0x0A	08	Val	XX	00	00	00	功能：自定义按键触发（按键矩阵中的自定义键） Val：自定义码 XX:1=按键按下 0=按键释放 例： 按下自定义键,其自定义码为 01： 09 0A 08 01 01 00 00 00 释放自定义键,其自定义码为 01： 09 0A 08 01 00 00 00 00
0x09	0x0A	09	Val	00	00	00	00	功能：键盘切换配置 Val：当前配置（0-2） 例： 从配置 1 切到 2： 09 0A 09 02 00 00 00 00
0x09	0x0A	0A	Val	XX	YY	00	00	功能：设备端有变化，pc 软件收到此命令之后，发起读取设备数据命令。收到命令之后等待 300ms 在读取。 Val：1=profile 表更改 2=自定义灯效更改 XX：被更改配置（0-2） YY：自定义灯效下有效，代表第几组灯效（0-4） 例： 键盘在配置 1 下锁 win： 主动上报：09 0A 0A 01 01 00 00 00 驱动下发读取 profile 表的命令 键盘在配置 1 自定义灯效 0 更改： 主动上报：09 0A 0A 02 01 00 00 00

								驱动下发读取自定义灯效的命令
0x09	0x0A	AA	Val	00	00	00	00	功能：驱动多设备通道验证。 Val：驱动主动下发 例： 驱动 set_report 下发 09 AA 00 00 01 00 01 00 05 00(511) 接收到上报： 09 0A AA 05 00 00 00 00

配置读写端口：

必须支持 setfeature 和 getfeature 命令，端口长度是 512+8 字节，格式如下

Set_Report:

BmRequestType	Request	Value		Index		Length	
		Low	High	Low	High	Low	High
21h	09h	09h	03h	01h	00h	08h	02h

Get_Report:

BmRequestType	Request	Value		Index		Length	
		Low	High	Low	High	Low	High
a1h	01h	09h	03h	01h	00h	08h	02h

数据格式：

BYTE0	BYTE1	BYTE2	BYTE3	BYTE4	BYTE5	BYTE 6	BYTE 7	BYTE8-BYTE519
RE ID	命令 ID	命令参数	预留字节	总包数	当前包序号	数据长度 L	数据长度 H	负载数据

RE ID：USB 模式下 RE ID 为 0x06/0x09，具体以键盘为主
命令 ID：用于指明是哪条命令，根据设备不同而变化。
命令参数：用于补充命令信息，例如读写宏数据时，指明是读写哪个宏缓冲区。
预留字节：若设备支持多板载，且命令 ID 是跟板载相关的，则 bit0-bit2 是板载序号，指明本包数据属于哪个板载。序号范围 0-2。
总包数：指明数据总包数。
当前包序号：指明数据当前包数。
数据长度：指明配置数据 BYTE8-BYTE519 中的有效数据的长度。BYTE6 为低 8 位，BYTE7 为高 8 位。
负载数据：数据格式根据设备不同而不同，例如矩阵数据是每 4 字节对应一个按键，颜色数据是 每 3 字节对应一个颜色。

例：宏数据是 1034 字节，而每次只能发送 512 字节，故总共要发送 1034÷512=2.1 次，，因不能整除 512 所以总共要发送 3 次，则总包数为 3。当前包序号的取值范围就是 0-2。数据长度前面两包为 512（byte6 为 0，

byte7 为 2），第三包的长度为 10（byte6 为 10，byte7 为 0）。Byte8-Byte519 为宏具体内容。

请求获取数据流程：

1. Set_report

21h	09h	Report id 09h	report type 03h	01h	00h	08h	02h

Set_report 数据：

BYTE0	BYTE1	BYTE2	BYTE3	BYTE4	BYTE5	BYTE 6	BYTE 7	BYTE8-BYTE519
Report id (09h)	命令 ID	命令参数	预留字节	总包数	当前包序号	数据长度 L	数据长度 H	负载数据

2. Get_report

a1h	01h	Report id 09h	report type 03h	01h	00h	08h	02h

返回 Get_report 数据：

BYTE0	BYTE1	BYTE2	BYTE3	BYTE4	BYTE5	BYTE 6	BYTE 7	BYTE8-BYTE519
Report id (09h)	命令 ID	命令参数	预留字节	总包数	当前包序号	数据长度 L	数据长度 H	负载数据

请求设置数据流程：

1. Set_report

21h	09h	Report id 09h	report type 03h	01h	00h	08h	02h

Set_report 数据：

BYTE0	BYTE1	BYTE2	BYTE3	BYTE4	BYTE5	BYTE 6	BYTE 7	BYTE8-BYTE519
Report id (09h)	命令 ID	命令参数	预留字节	总包数	当前包序号	数据长度 L	数据长度 H	负载数据

备注：有些命令需要设备冒泡回复

键盘命令定义

协议只规定数据 ID，不规定键盘内部存储地址。

RE ID	命令 ID	命令参数	预留	总包数	当前包序号	数据长度 L	数据长度 H	说明

0x09	0x01	Val	XX	01	00	00	00	功能：设备通过该命令做准备工作,若不需要可忽 略该命令 Val:（bit7）1=驱动准备发送配置数据 （bit6）1=驱动准备读取配置数据 （bit7-bit6）0=驱动发送/读取完毕 Bit0-Bit5 指明驱动后续命令将会发送数据类型 （bit5）1=实时灯效数据（音乐律动） （bit4）1=自定义模式颜色组数据（游戏模式） （bit3）1=灯效颜色 （bit2）1=profile 表 （bit1）1=宏数据 （bit0）1=按键矩阵数据
0x09	0x82	Val	00	01	00	XX	00	功能：读取键盘密码或者宏存储空间大小。 Val：0=宏空间大小 1=读取密码和设备版本号 (驱动决定是否需要读取版本号，不读取版本号则只有 6byte 密码，读取版本号则 6byte 密码 +2byte 版本号<高字节在前低字节在后>) XX：Val=0，XX=04 Val=1，XX=06/08 例：读取密码 09 82 01 00 01 00 06 00 XX (6byte 的密码) 读取密码和版本号 09 82 01 00 01 00 08 00 XX (6byte 的密码 2byte 的版本号码) 读取宏长度，宏预留空间 1K 09 82 00 00 01 00 04 00 00 04 00 00 (4byte 的长度，1K=1024byte，从低位到高位)
0x09	0x03	Val	XX	01	00	F8	01	功能：写按键矩阵数据，一个按键占用 4 个字节，驱动一次写入一个 layer 的数据（504byte=21 列*6 行），矩阵排序按列，第一个键位是 ESC,第二个键位是~，第三个键位 tab。 Val：（bit0-bit1）=0 写 normal layer （bit0-bit1）=1 写 FN1 layer （bit0-bit1）=2 写 FN2 layer （bit0-bit1）=3 写 Tap layer （bit2-bit4）=0 写 win 系统的按键表 （bit2-bit4）=1 写 mac 系统的按键表 XX：（bit0-bit2）指定板载（0-2）

								例：驱动下发板载 1，normal 层的 mac 系统下的按键 驱动发 set_report 命令 09 03 04 01 01 00 F8 01 XX （504 个有效 byte 代表矩阵按键）00（8 个无效 byte）
0x09	0x83	Val	XX	01	00	F8	01	功能：读按键矩阵数据，一个按键占用 4 个字节，驱动一次写入一个 layer 的数据（504byte=21 列*6 行），矩阵排序按列，第一个键位是 ESC,第二个键位是~，第三个键位 tab。 Val：（bit0-bit1）=0 读 normal layer （bit0-bit1）=1 读 FN1 layer （bit0-bit1）=2 读 FN2 layer （bit0-bit1）=3 读 Tap layer （bit2-bit4）=0 读 win 系统的按键表 （bit2-bit4）=1 读 mac 系统的按键表 XX：（bit0-bit2）指定板载（0-2） 例：驱动读板载 1，normal 层的 mac 系统下的按键 驱动先发 set_report 命令 09 83 04 01 01 00 F8 01 00 （512 个 0） 在发 get_report 命令 09 83 04 01 01 00 F8 01 XX （XX 代表矩阵按键）
0x09	0x04	00	Val	01	00	80	00	功能：写配置数据（profile 表）。 Val：（bit0-bit2）指定板载（0-2） 例：驱动设置板载 1 的配置 驱动发 set_report 命令 09 04 00 01 01 00 80 00 XX(128 个有效数据) 00（384 个无效 byte）
0x09	0x84	00	Val	01	00	80	00	功能：读配置数据（profile 表）。 Val：（bit0-bit2）指定板载（0-2） 例：驱动读板载 1 配置 驱动先发 set_report 命令 09 84 00 01 01 00 80 00 00 （512 个 0） 在发 get_report 命令 09 84 00 01 01 00 80 00 XX （128 个 byte，profile 表）
0x09	0x05	00	00	Val	XX	YY	ZZ	功能：写宏数据，数据格式参考可变长度宏格式。 Val：总包数，根据宏的长度变化，一包最长为 512byte。 XX：当前包数，根据宏的长度变化，一包

								最长为 512byte。 YY-ZZ：数据有效长度 例：驱动设置宏的内容为 508 字节 驱动发 set_report 命令 09 05 00 00 01 00 fc 01 XX(508 个有效数据) 00（4 个无效 byte） 驱动设置宏的内容为 520 字节 驱动发 2 包 set_report 命令 09 05 00 00 02 00 00 02 XX(512 个有效数据) 09 05 00 00 02 01 08 00 XX(8 个有效数据)00（504 个无效 byte）
0x09	0x85	00	Val	XX	YY	ZZ	Val	功能：读宏数据，数据格式参考可变长度宏格式。 Val：总包数，根据宏的长度变化，一包最长为 512byte。 XX：当前包数，根据宏的长度变化，一包最长为 512byte。 YY-ZZ：数据有效长度 例：驱动读取宏的内容为 508 字节 驱动发 set_report 命令 09 85 00 00 01 00 fc 01 00(512 个 0) 在发 get_report 命令 09 85 00 00 01 00 fc 01 XX （508 个 byte 的宏内容） 驱动读取宏的内容为 520 字节 驱动发 set_report 命令 09 85 00 00 02 00 00 02 00 (512 个 0) 在发 get_report 命令 09 85 00 00 02 00 00 02 XX （512 个 byte 的宏内容） 驱动发 set_report 命令 09 85 00 00 02 01 08 00 00 (512 个 0) 在发 get_report 命令 09 85 00 00 02 01 08 00 XX （8 个 byte 的宏内容）
0x09	0x06	Val	XX	01	00	7A	01	功能：写自定义灯效数据（游戏灯效），数据格式红色 126byte，绿色 126bye，蓝色 126byte。排序按列，第一个 byte 是 ESC 键位,第二个 byte 是~键位，第三个 byte 是 tab 键位。 Val：第几组自定义灯效（目前只支持一组，即 Val=0）。 XX：（bit0-bit2）指定板载（0-2）。 例：驱动设置板载 1 的游戏灯效

								驱动发 set_report 命令 09 06 00 01 01 00 7a 01 XX(126byte 红色表) YY（126byte 绿色表）ZZ（126byte 蓝色表）00（134 个无效 byte）
0x09	0x86	Val	XX	01	00	7A	01	功能：读自定义灯效数据（游戏灯效），数据格式红色 126byte，绿色 126bye，蓝色 126byte。排序按列，第一个 byte 是 ESC 键位,第二个 byte 是~键位，第三个 byte 是 tab 键位。 Val：第几组自定义灯效（目前只支持一组，即 Val=0）。 XX：（bit0-bit2）指定板载（0-2）。 例：驱动读取板载 1 的游戏灯效 驱动发 set_report 命令 09 86 00 01 01 00 7a 01 00（512 个无效数据） 在发 get_report 命令 09 86 00 01 01 00 7a 01 XX(126byte 红色表) YY（126byte 绿色表）ZZ（126byte 蓝色表）
0x09	0x87	00	00	01	00	02	00	功能:读取电量以及电池的状态(是否充电) 例：驱动读取电量 驱动发 set_report 命令 09 87 00 00 01 00 02 00 00（512 个无效数据） 在发 get_report 命令 09 87 00 00 01 00 02 00 XX(1byte，电池剩余电量) YY（1byte，高 4bit=1 充电中，高 4bit=0 未充电，低 4bit=1 已充满，低 4bit=0 未充满）
0x09	0x08	00	00	01	00	7A	01	功能：发送音乐灯效数据（实时更新，一次一帧数据，378byte 的有效数据），数据格式第一个键位 RGB,第二个键位 RGB。。。126 个键位。排序按列，第一个是 ESC 键位,第二个是~键位，第三个是 tab 键位。 例：驱动下发音乐律动 驱动发 set_report 命令 09 08 00 00 01 00 7a 01 RGB RGBRGB 。。。(126 个 RGB) 00（134 个无效 byte）
0x09	0x88	00	00	01	00	7A	01	功能：读取前灯光颜色数据（实时更新，一次一帧数据，378byte 的有效数据），数据格式第一个键位 RGB,第二个键位 RGB。。。126 个键位。排序按列，第一个是 ESC 键位,第二个是~键位，第三个是 tab 键位。

								例：驱动读取当前灯光颜色 驱动发 set_report 命令 09 88 00 00 01 00 7a 01 00（512 个无效数据） 在发 get_report 命令 09 88 00 00 01 00 7a 01 RGB RGBRGB 。。。(126 个 RGB) 00（134 个无效 byte
0x09	0x0a	00	Val	01	00	e3	01	功能：写灯效对应颜色数据（每个灯效下有 7 组 RGB 支持修改，即切换当前灯效颜色循环时候显示的颜色），数据格式灯效 0 对应的颜色 RGB,RGB,RGB...(共 7 组),灯效 1 对应的颜色 RGB,RGB,RGB...(共 7 组) 共灯效 0-22，总数据 23*7*3=483byte。 Val：（bit0-bit2）指定板载（0-2）。 例：驱动设置板载 1 的灯效 2 下的第二个颜色为红色 驱动发 set_report 命令 09 0a 00 01 01 00 e3 01 XX(42 个 byte，灯效 0 的 21byte，灯效 1 的 21byte) YY（3 个 byte，灯效 2 的第一个颜色）ff 00 00，ZZ（15 个 byte，灯效 2 的后面 5 个颜色）NN（420 个 byte，后面 20 个灯效的颜色）00（29 个无效 Byte）
0x09	0x8a	00	Val	01	00	e3	01	功能：读灯效对应颜色数据（每个灯效下有 7 组 RGB，即切换当前灯效颜色循环时候显示的颜色），数据格式灯效 0 对应的颜色 RGB,RGB,RGB...(共 7 组)，灯效 1 对应的颜色 RGB,RGB,RGB...(共 7 组) 共灯效 0-22，总数据 23*7*3=483byte。 Val：（bit0-bit2）指定板载（0-2）。 例：读取板载 1 的颜色 驱动发 set_report 命令 09 8a 00 01 01 00 e3 01 00（512 个无效 Byte） 在发 get_report 命令 09 8a 00 01 01 00 e3 01 XX(483 个 byte 的颜色数据)
0x09	0x0b	00	00	01	00	0e	00	功能：发送给屏显示的数据 Byte8：系统音量 Byte9：CPU 使用率 Byte10：内存使用率 Byte11：年份的低字节 Byte12：年份的高字节 Byte13：月份

								Byte14：日 Byte15：时 Byte16：分 Byte17：秒 Byte18：星期 X Byte19- Byte21：预留 例：驱动发送音量 20，cpu 使用率 50，内存 50，2024 年 3 月 20 日 19 点 20 分 30 秒星期三 驱动发 set_report 命令 09 0b 00 00 01 00 0b 00 14 32 32 18 14 03 14 13 14 1e 03 XX(3byte 预留) 00(498byte 的无效数据)
0x09	0x0c	Val	XX	YY	CRC	NN	MM	功能：发送给屏显示动图的数据（主控不做保存，直接发送给刷屏的 IC 做处理），一组动图会有多组帧数据（帧数量来自于动图的图片的数量），一帧数据有多组包数据（包数量取决于图片的大小）。每发送一个数据包需要等待键盘的冒泡数据来告知驱动是处理成功与否。处理成功以及 CRC 正确，继续发送下一组，否则重发当前数据。 Val：动图组序号，指明本包数据属于那组动图(0-N) XX：当前帧序号（0-N） YY：当前包序号，代表的当前帧的序号 (0-N) CRC：0xFF&（Byte8+..+ByteN） NN：有效数据的低字节 MM：有效数据的高字节 例：发送第一组动图的第二帧图片的第三包数据 驱动发 set_report 命令 09 0c 00 01 02 CRC(计算结果) 00 02 XX(512byte 的图片数据) 键盘计算 CRC 与接收到 CRC 做比较，如果一样，则发送给屏幕 IC，发送完成之后，冒泡给驱动 09 0a 06 01 01 crc 00 00 发送失败或者 CRC 对比不一样回复： 09 0a 06 00 01 crc 00 00
0x09	0x0d	00	00	01	00	05	00	功能：发送 0x0c 命令之前发送，该数据处理设备主控需要处理，同时也需要发送给控制屏幕的 IC。 Byte8：（bit7-bit6）0=结束传输 （bit7-bit6）1=开始传输

								（bit7-bit6）2=取消传输 （bit5-bit0）第几组动图 Byte9：当前动图共有多少帧 Byte11-Byte12：当前动图每帧播放间隔，单位毫秒 例：开始传输第一组动图，该动图共 5 帧，每帧间隔 10 毫秒 驱动发 set_report 命令 09 0d 00 00 01 00 05 00 40 05 00 0a 00 00（507 个 byte 无效数据）
0x09	0x0e	00	00	01	00	NN	MM	功能：发送边灯音乐灯效数据（实时更新，一次一帧数据，有效数据个数取决于有多少个灯 N <N*3>），数据格式第一个灯 RGB，第二个灯 RGB。。。N 个键位。 NN-MM:有效数据长度 例：驱动下发灯条共 7 个灯音乐律动 驱动发 set_report 命令 09 0e 00 00 01 00 15 00 RGB RGBRGB 。。。(7 个 RGB) 00（491 个无效 byte）
0x09	0x10	00	00	01	00	01	00	功能：设置当前板载 Byte8：表明当前板载（0-2） 例：切换到到板载 2 驱动发 set_report 命令 09 10 00 00 01 00 01 00 02 00（511 个无效 byte）
0x09	0x90	00	00	01	00	01	00	功能：读当前设备的板载。 Byte8：表明设备当前板载（0-2） 例：读取设备当前板载（当前板载 2） 驱动发 set_report 命令 09 90 00 00 01 00 01 00 00（512 个无效 Byte） 在发 get_report 命令 09 90 00 00 01 00 01 00 02
0x09	0x11	00	00	01	00	01	00	功能：键盘复位 Byte8：=0：整机复位（profile，按键，灯光） =1：按键复位 =2：灯光复位 例：整机复位 驱动发 set_report 命令 09 11 00 00 01 00 01 00 00 00（511 个无效 byte）

0x09	0x71	00	00	01	00	00	01	功能：下发用户自定义数据，用于设备的区别（产品序列号，产品名称等，这部分数据升程序后也不允许修改）。 例：设置产品序列号 03 05 08 驱动发 set_report 命令 09 71 00 00 01 00 00 01 03 05 08 00 （253 个 byte 的 0） 00（256 个无效 byte）
0x09	0xf1	00	00	01	00	00	01	功能：读取用户自定义数据，用于设备的区别（产品序列号，产品名称等，这部分数据升程序后也不允许修改）。 例：读产品序列号 驱动发 set_report 命令 09 f1 00 00 01 00 00 01 00（512 个无效 Byte） 在发 get_report 命令 09 f1 00 00 01 00 00 01 XX(256byte 的有效数据)
0x09	0xAA	00	00	01	00	01	00	功能：驱动多设备通道验证。 Val：驱动主动下发 例： 驱动 set_report 下发 09 AA 00 00 01 00 01 00 05 00(511) 接收到主动上报： 09 0A AA 05 00 00 00 00

可变长宏格式（宏的最大长度由设备上报）：

1. 宏数据结构

为节省空间，宏须支持可变长度，宏存储区由“宏地址表+宏数据区”组成，格式如下：

宏地址表：

前 N*4 个字节是宏地址表，地址表每 4 个自己存一个宏的位置信息，前 2 字节表示存储偏移地址（相对于地址表起始处），后 2 字节表示该宏的数据长度（单位字节）。、

地址表的长度是可变的，N 是宏的数量。

地址表的信息格式：

宏 AddrL	宏 AddrH	宏 LenL	宏 LenH

宏数据区：

第一个字节代表宏名字的长度，第 2 个字节起代表宏名字符串数据(宏名字设备只做记录，驱动读取解析显示)，宏名后跟宏按键动作数据。

◆宏长度：宏名长度字节+宏名+宏按键动作

◆宏偏移地址和长度存储在宏内容的最前面

◆宏按键动作：

每个按键动作由 4 个 byte 组成，动作格式如下：

字节 1：（bit7）1=按键弹起

（bit7）0=按键按下

（bit6-bit4）0=键盘普通键（字节 4：按键 HID 键值）

（bit6-bit4）1=键盘 modify 键（字节 4：L_Ctrl = 0xE0，L_Shift = 0xE1 L_Alt = 0xE2，L_Win = 0xE3，R_Ctrl = 0xE4，R_Shift = 0xE5 R_Alt = 0xE6，R_Win = 0xE7）

（bit6-bit4）2=鼠标按键(字节 4：左键=0x01，右键=0x02，中键=0x04，前进键=0x08，后退键=0x10）

（bit6-bit4）3=鼠标光标 X 轴（字节 4：位移数据，不做处理直接上传）

（bit6-bit4）4=鼠标光标 Y 轴（字节 4：位移数据，不做处理直接上传）

（bit6-bit4）5=鼠标滚轮（字节 4：有数值（bit7）1=下滚（bit7）0=上滚）

（bit3-bit0）动作中间延时时间的高 4bit

字节 2：动作中间延时时间的中 8bit

字节 3：动作中间延时时间的低 8bit（延时时间的最小单位 1ms）

字节 4：意义跟随字节 1 里的种类有关系

例：存储了 2 组宏，第一组宏名称 123，按键动作 6 个。第二组宏名称 AB，按键动作 2 个，动作为 A 按下 10ms，A 松开

宏 1 偏移地址 L 08h	宏 1 偏移地址 H 00h	宏 1 长度 L 1ch	宏 1 长度 H 00h	宏 2 偏移地址 L 24h	宏 2 偏移地址 H 00h	宏 2 长度 L 0Bh	宏 2 长度 H 00h
宏 1 名字长度 03h	宏 1 名字字符	宏 1 名字 字符	宏 1 名字字符	宏 1 按键动作 1 XX(4byte)	宏 1 按键动作 2 XX(4byte)	宏 1 按键动作 3 XX(4byte)	宏 1 按键动作 4 XX(4byte)
宏 2 名字长度 02h	宏 2 名字字符	宏 2 名字 字符	宏 2 按键动作 1 00	宏 2 按键动作 1 00	宏 2 按键动作 1 0A	宏 2 按键动作 1 04	宏 2 按键动作 2 80
宏 2 按键动作 2 00	宏 2 按键动作 2 00	宏 2 按键动作 2 04					

按键矩阵表

矩阵表存放每个按键的设定，每 4byte 存储一个按键，按键格式如下：

按键类型	Byte4	Byte3	Byte2	Byte1
禁用按键	00	00	00	00
键盘按键	00	Modify（bit0）1=L_Ctrl （bit1）1=L_Shift （bit2）1=L_Alt （bit3）1=L_Win （bit4）1= R_Ctrl （bit5）1=R_Shift （bit6）1=R_Alt （bit7）1=R_Win	Hid1(普通按键 hid 值)	Hid2(普通按键 hid 值)
鼠标按键	01	鼠标键码 1=左键，2=右键，3=中键 4=前进键，5=后退键 6=左摆，7=右摆 8=滚轮上，9=滚轮下 10=X 轴左，11=X 轴右 12=Y 轴上，13=Y 轴下	单击次数 0xff 火力键（按键按下一直自动做击打重复，直到按键释放） 0 单击（正常触发） 1 双击 2 三连发 数值为 N，连发 N+1 次	单击间隔 单位毫秒 若为 0，则使用设备的默认间隔时间
多媒体键	02	00	Meida_H	Meida_L
宏按键	03	宏循环方式 1=指定循环次数 2=循环直到任意键按下 4=循环直到该按键松开	循环次数 若循环方式为 1，指定具体的循环次数（从 1 开始）	宏序号 ID 指定到具体的第几个宏
自定义键	04	00	00	自定义码 此类型按键通过冒泡端口上传，详见前面冒泡端口数据处理
DPI 按键（鼠标有效）	05	DPI 码 1=dpi+，2=dpi- 4=dpi 循环	00	00
切换配置	06	切换码 1=配置+，2=配置- 4=配置循环	00	00
特殊功能	07	保留	保留	=0：取反 WASD 和方向 =1：win 开关锁 =2：全键盘开关锁 =4：整键盘复位 =5：切换到蓝牙模式 0 =6：切换到蓝牙模式 1 =7：切换到蓝牙模式 2 =8：切换到 2.4G 模式

				=9：切换到 USB 模式 =10：进入无线测试 （此功能会根据规格不断增加，具体数据参考 keyboard.H 文件）
灯效/亮度/速度/颜色切换键	08	=0：灯效切换 =1：灯效方向 =2：颜色切换 =3：亮度切换 =4：速度切换 =5：启动自定义灯效录制 =6：保存自定义灯效 =7：启动录制与保存自定义灯效 =8：复位灯效 =9：直接切换到特定灯效	=0：循环 =1：+/左 =2：-/右 Byte3 如果是 9，该字节为直接切换到的灯效模式	=0：主键区灯效 =1：边灯 =2：logo 灯
回报率键	09	=0：直接设定档位 =1：回报率+ =2：回报率- =3：回报率循环	Byte3 如果是 0，该字节为直接切换到的回报率档位	=0：125 =1：250 =2：500 =3：1000
狙击键 (鼠标有效)	0a	Dpi 锁定档位 当狙击键按下时，鼠标将 dpi 切换到该档位，0 代表第一档	00	00
压枪键 (鼠标有效)	0b	压枪参数	射速	Y 轴压枪幅度
FN 键	0d	=0：fn1 =1：fn2	00	00
Lod 键 (鼠标有效)	0f	静默高度参数	00	00

附：自定义描述符

```
0x06, 0x02, 0xFF,       // usage page(3)
0x09, 0x02,             // usage(2)
0xA1, 0x01,             // Collection(Application)
0x85, 0x09,             // REPORT_ID (9)
0x15,0x00,
0x26,0xFF,0x00,
0x75,0x08,
0x95,0x07,
0x09,0x02,
0x81,0x00,
0x15, 0x00,                 // Logical Minimum (0)
0x26, 0xFF, 0x00,       // Logical Maximum (255)
0x19, 0x01,                 // Usage Minimum(0x01)
```

```
0x29, 0x02,                 // Usage Maximum(0x05)
0x75, 0x08,                 // REPORT SIZE (8)
0x96, 0x07, 0x02,           // REPORT COUNT(512+8-1)
0xB1, 0x02,                 // FEATURE(DATA,VARIABLE,ABSOLUTE)
0xC0,                       // END COLLECTION
```

无线（2.4G）模式

驱动与 USB 设备之间的数据传输通道和流程

与驱动相关的的数据传输端口只有 2 个：

1.自定义数据端口（用于键盘主动通知驱动以及驱动下传数据时的回复 IN 数据）

2. 配置数据写端口（Set_Report Output 下传）

自定义数据端口：

通过 EP 的自定义上传数据。

命令 ID 是 0A 时，代表这是 dongle 通过 EP 端口主动上报的数据，数据格式如下：

13 0A 0100 04 id val00 00 00 00 00 00 00 00 00 00 00crc（crc =0xFF&（byte0+..+byte18）

Byte0:13 的位置）

主动上报说明如下:

id	val
02	上报 dongle 和设备的连接状态 =1：连接成功 =0：失去连接 例：dongle 和设备连接成功时，dongle 需主动上报 13 0A 01 00 04 02 01 00 00 00 00 00 00 00 00 00 00 00 crc dongle 和设备断开连接时，dongle 需主动上报 13 0A 01 00 04 02 00 00 00 00 00 00 00 00 00 00 00 00 crc
03	上报鼠标当前 DPI 挡位的改变（鼠标有效） Val：当前 DPI 挡位（0~7） 例：鼠标 DPI 切换到档位 3，dongle 主动上报 13 0A 01 00 04 03 03 00 00 00 00 00 00 00 00 00 00 00 crc
05	上报电量状态 Val：当前剩余电量（值范围 0~100） Val+1byte：（高 4bit）=1：正在充电 （高 4bit）=0：未充电 （低 4bit）=1：已充满 （低 4bit）=0：未充满 例：设备充电中，电池电量 15 13 0A 01 00 04 050f 10 00 00 00 00 00 00 00 00 00 00 00 crc
07	上报设备当前使用系统 Val：=1 键盘切换到 MAC 系统 =0 键盘切换到 Windows 系统 例：键盘切换到 MAC 时，键盘上报 13 0A 01 00 04 07 01 00 00 00 00 00 00 00 00 00 00 00 crc

	键盘切换到 WIN 时，键盘上报 13 0A 01 00 04 07 00 00 00 00 00 00 00 00 00 00 00 00 crc
08	自定义按键 Val：自定义编码（该自定义编码包含在按键矩阵的键码中） Val+1byte:1=按键按下 0=按键释放 例：按下自定义键,其自定义码为 01： 13 0A 01 00 04 08 01 01 00 00 00 00 00 00 00 00 00 00 crc 释放自定义键,其自定义码为 01： 13 0A 01 00 04 08 01 00 00 00 00 00 00 00 00 00 00 00 crc
09	键盘切换配置 Val：当前配置（0-2） 例：从配置 1 切到 2： 13 0A 01 00 04 09 02 00 00 00 00 00 00 00 00 00 00 00 crc
0A	功能：设备端有变化，pc 软件收到此命令之后，发起读取设备数据命令。收到命令之后等待 300ms 在读取。 Val：1=profile 表更改 2=自定义灯效更改 Val+1byte：被更改配置（0-2） Val+2byte:：自定义灯效下有效，代表第几组灯效（0-4） 例： 键盘在配置 1 下锁 win： 主动上报：13 0A 01 00 04 0A 01 01 00 00 00 00 00 00 00 00 00 00 crc 驱动下发读取 profile 表的命令 键盘在配置 1 自定义灯效 0 更改： 主动上报：13 0A 01 00 04 0A 02 01 00 00 00 00 00 00 00 00 00 00 crc 驱动下发读取自定义灯效的命令

配置读写端口：

必须支持 setfeature 命令，端口长度是 20 字节，格式如下

Set_Report:

BmRequestType	Request	Value		Index		Length	
		Low	High	Low	High	Low	High
21h	09h	13h	02h	01h	00h	14h	00h

数据格式：

BYTE0	BYTE1	BYTE2	BYTE3	BYTE4	BYTE5-BYTE18	BYTE19
RE ID（13）	数据标识	总包数	当前包序号	命令参数	负载数据	CRC

RE ID：report id 为 13H。
数据标识：（bit7）=1：当前数据不需要通过设备主动上传数据校验
（bit7）=0：当前数据需要通过设备主动上传数据，用于校验
（bit6-bit0）：命令 ID
总包数：（bit7）=1：用于主动上传数据时告知驱动，当前 set 发送下来的数据失败

（bit7）=0：用于主动上传数据时告知驱动，当前 set 发送下来的数据成功
（bit6-bit0）：数据总包数
当前包序号：（bit7）=1：Mac 系统
（bit7）=0：Windows 系统

（bit6-bit0）：取值范围为 0-（总包数-1）

命令参数：（低 4bit）：byte5-byte18 中的有效长度
（bit4-bit5）：命令参数
（bit6-bit7）：板载序号（0-2）
负载数据：数据格式根据设备不同而不同，例如矩阵数据是每 4 字节对应一个按键，颜色数据是 每 3 字节对应一个颜色。

CRC：0xFF&（byte0+..+byte18）

例：profile 数据是 128 字节，而每次只能发送 14 字节有效数据，故总共要发送 128÷14=9.1 次，，因不能整除 14 所以总共要发送 10 次，则总包数为 10。当前包序号的取值范围就是 0-9。数据有效长度前面 9 包为 14（byte4 为低 4bit=e），第 10 包的有效长度为 2（byte4 为低 4bit=2）。Byte5-Byte18 为具体内容。

请求获取数据流程：

3. Set_report

21h	09h	Report id 13h	report type 02h	01h	00h	14h	00h

Set_report 数据：

BYTE0	BYTE1	BYTE2	BYTE3	BYTE4	BYTE5-BYTE18	BYTE19
RE ID（13）	数据标识	总包数	当前包序号（0-N）	命令参数	负载数据	CRC

4. IN 返回数据

BYTE0	BYTE1	BYTE2	BYTE3	BYTE4	BYTE5-BYTE18	BYTE19
RE ID（13）	数据标识	总包数	当前包序号（0-N）	命令参数	负载数据	CRC

备注：返回数据有可能是连续多包，需要根据具体的命令决定

请求设置数据流程：

2. Set_report

21h	09h	Report id 13h	report type 02h	01h	00h	14h	00h

Set_report 数据：

BYTE0	BYTE1	BYTE2	BYTE3	BYTE4	BYTE5-BYTE18	BYTE19
RE ID（13）	数据标识	总包数	当前包序号（0-N）	命令参数	负载数据	CRC

3. IN 返回数据

BYTE0	BYTE1	BYTE2	BYTE3	BYTE4	BYTE5-BYTE18	BYTE19
RE ID（13）	数据标识	总包数	当前包序号（0-N）	命令参数	负载数据	CRC

备注：返回数据有可能当前设置下来的原始数据（用于校验），有可能是需要读取的一些数据。

键盘命令定义

协议只规定数据 ID，不规定键盘内部存储地址。

RE ID	数据标识	总包数	当前包序号	命令参数	负载数据	CRC	说明
13	01	VAL	XX	YY	ZZ	CRC	功能：写按键矩阵数据，一个按键占用 4 个字节，驱动一次写入一个 layer 的数据（504byte=21 列*6 行），矩阵排序按列，第一个键位是 ESC,第二个键位是~，第三个键位 tab。 VAL：（bit7 用于上报）=0：下发数据成功 （bit7 用于上报）=1：下发数据失败 （bit6-bit0）：该命令会下发包总数 XX：（bit7)=0：Win 系统 （bit7)=1：Mac 系统 （bit6-bit0）：当前包序号，从 0 开始至包总数减 1 YY:（bit7- bit 6）板载序号（0-2） （bit5- bit4）=0:写 normal layer （bit5-bit4）=1:写 FN1 layer （bit5-bit4）=2:写 FN2 layer （bit5-bit4）=3:写 Tap layer （bit3-bit0）:负载数据区的有效数据 ZZ:负载数据 CRC: 0xFF&（byte0+..+byte18） 例：设置板载 1 mac 系统下 fn1 层的按键 Set_report: 13 01 24 80 5e xx(14byte 的按键数据) crc IN：（成功） 13 01 24 80 5e xx(14byte 的按键数据) crc Set_report: 13 01 24 81 5e xx(14byte 的按键数据) crc IN：（失败，驱动重发当前笔） 13 01 a4 81 5e xx(14byte 的按键数据) crc

							Set_report: 13 01 24 81 5e xx(14byte 的按键数据) crc IN：(成功) 13 01 24 81 5e xx(14byte 的按键数据) crc 。。。 Set_report: 13 01 24 a3 5e xx(14byte 的按键数据) crc IN：(成功) 13 01 24 a3 5e xx(14byte 的按键数据) crc 若同一笔失败 10 次，则认为设备出问题驱动报错，不在下传。若发送数据没有等到回复，则过 30ms 再次重发，重发 10 次，一直没回复，驱动报错，不在下发。
13	41	VAL	XX	YY	ZZ	CRC	功能：读按键矩阵数据，一个按键占用 4 个字节，驱动一次写入一个 layer 的数据（504byte=21 列*6 行)，矩阵排序按列，第一个键位是 ESC,第二个键位是~，第三个键位 tab。 VAL：(bit6-bit0)：该命令会下发包总数 XX：(bit7)=0：Win 系统 (bit7)=1：Mac 系统 (bit6-bit0)：当前包序号，从 0 开始至包总数减 1 YY:（bit7- bit 6）板载序号（0-2） （bit5- bit4）=0:读 normal layer (bit5-bit4）=1:读 FN1 layer (bit5-bit4）=2:读 FN2 layer (bit5-bit4）=3:读 Tap layer (bit3-bit0）:负载数据区的有效数据 ZZ:负载数据 CRC: 0xFF&（byte0+..+byte18） 例：读取板载 1 mac 系统下 fn1 层的按键 Set_report: 13 4101 80 50 00(14byte 的 0) crc IN： 13 41 24 80 5e xx(14byte 的按键数据) crc IN： 13 41 24 81 5e xx(14byte 的按键数据) crc 。。。 IN： 13 41 24 a3 5e xx(14byte 的按键数据) crc Set_report 命令里面的总包数为 1，上传的总包数由设备计算，驱动对数据连续性做校验，如果不连续，等数据上传之后，在重新读取，重复十次失败，驱动报错。

13	02	VAL	XX	YY	ZZ	CRC	功能：写自定义灯效数据（游戏灯效），数据格式红色 126byte，绿色 126bye，蓝色 126byte。排序按列，第一个 byte 是 ESC 键位,第二个 byte 是~键位,第三个 byte 是 tab 键位。 VAL：（bit7 用于上报）=0：下发数据成功 （bit7 用于上报）=1：下发数据失败 （bit6-bit0）：该命令会下发包总数 XX：（bit6-bit0）：当前包序号，从 0 开始至包总数减 1 YY:（bit7- bit6）：板载序号（0-2） （bit5- bit4）：第几组游戏灯效（0-4） （bit3-bit0）:负载数据区的有效数据 ZZ：负载数据 CRC: 0xFF&（byte0+..+byte18） 例：写板载 1 第 1 组游戏灯效 Set_report: 13 021b00 4e xx(14byte 的有效数据) crc IN：（成功） 13 021b00 4e xx(14byte 的有效数据) crc Set_report: 13 021b01 4e xx(14byte 的有效数据) crc IN：（失败，驱动重发当前笔） 13 029b01 4e xx(14byte 的有效数据) crc Set_report: 13 021b01 4e xx(14byte 的有效数据) crc IN：（成功） 13 021b01 4e xx(14byte 的有效数据) crc 。。。 Set_report: 13 021b1a4e xx(14byte 的有效数据) crc IN：（成功） 13 021b1a4e xx(14byte 的按键数据) crc 若同一笔失败 10 次，则认为设备出问题驱动报错，不在下传。若发送数据没有等到回复，则过 30ms 再次重发，重发 10 次，一直没回复，驱动报错，不在下发。
13	42	VAL	XX	YY	ZZ	CRC	功能：读自定义灯效数据（游戏灯效），数据格式红色 126byte，绿色 126bye，蓝色 126byte。排序按列，第一个 byte 是 ESC 键位,第二个 byte 是~键位,第三个 byte 是 tab 键位。 VAL：（bit6-bit0）：该命令会下发包总数 XX：（bit6-bit0）：当前包序号，从 0 开始至包总数减 1

							YY: （bit7- bit6）：板载序号（0-2） （bit5- bit4）：第几组游戏灯效（0-4） （bit3-bit0）:负载数据区的有效数据 ZZ：负载数据 CRC: 0xFF&（byte0+..+byte18） 例：读取板载 1 第一组游戏灯效数据 Set_report: 13 420100 40 00(14byte 的 0) crc IN： 13 421b00 4e xx(14byte 的有效数据) crc IN： 13 421b01 4e xx(14byte 的有效数据) crc 。。。 IN： 13 421b1a4e xx(14byte 的有效数据) crc Set_report 命令里面的总包数为 1，上传的总包数由设备计算，驱动对数据连续性做校验，如果不连续，等数据上传之后，在重新读取，重复十次失败，驱动报错。
13	03	VAL	XX	YY	ZZ	CRC	功能：写宏数据，数据格式参考可变长度宏格式。 VAL：（bit7 用于上报）=0：下发数据成功 （bit7 用于上报）=1：下发数据失败 （bit6-bit0）：该命令会下发包总数 XX：（bit6-bit0）：当前包序号，从 0 开始至包总数减 1 YY:（bit7- bit4）：第多少组 512 字节（从 0 开始） （bit3-bit0）:负载数据区的有效数据 ZZ：负载数据 CRC: 0xFF&（byte0+..+byte18） 例：写宏数据共 515 字节 Set_report: 13 032500 0e xx(14byte 的有效数据) crc IN：（成功） 13 032500 0e xx(14byte 的有效数据) crcSet_report: 13 0325010e xx(14byte 的有效数据) crc IN：（失败，驱动重发当前笔） 13 03a5010e xx(14byte 的有效数据) crc Set_report: 13 0325010e xx(14byte 的有效数据) crc IN：（成功） 13 0325010e xx(14byte 的有效数据) crc 。。。

							Set_report: 13 03252408 xx(8byte 的有效数据) 00（6byte 的 0）crc IN：（成功） 13 03252408 xx(8byte 的有效数据) 00（6byte 的 0）crc Set_report: 13 030100 13 xx(3byte 的有效数据) 00（11byte 的 0）crc IN：（成功） 13 030100 13 xx(3byte 的有效数据) 00（11byte 的 0）crc 若同一笔失败 10 次，则认为设备出问题驱动报错，不在下传。若发送数据没有等到回复，则过 30ms 再次重发，重发 10 次，一直没回复，驱动报错，不在下发。
13	43	VAL	XX	YY	ZZ	CRC	功能：读宏数据，数据格式参考可变长度宏格式。 VAL：（bit6-bit0）：该命令会下发包总数 XX：（bit6-bit0）：当前包序号，从 0 开始至包总数减 1 YY：（bit7- bit4）：第多少组 512 字节（从 0 开始） （bit3-bit0）:负载数据区的有效数据 ZZ：下发命令时最前面 2byte 为需读取的数据长度，上传时为负载数据 CRC: 0xFF&（byte0+..+byte18） 例：读宏数据共 515 字节 Set_report: 13 430100 10 03 02 00(12byte 的 0) crc IN： 13 4325000e xx(14byte 的有效数据) crc IN： 13 4325010e xx(14byte 的有效数据) crc 。。。 IN： 13 43252408 xx(8byte 的有效数据) 00（6byte 的 0）crc IN： 13 430100 13 xx(3byte 的有效数据) 00（11byte 的 0）crc Set_report 命令里面的总包数为 1，上传的总包数由设备计算，驱动对数据连续性做校验，如果不连续，等数据上传之后，在重新读取，重复十次失败，驱动报错。

13	04	VAL	XX	YY	ZZ	CRC	功能：写配置数据（profile 表） VAL：（bit7 用于上报）=0：下发数据成功 （bit7 用于上报）=1：下发数据失败 （bit6-bit0）：该命令会下发包总数 XX：（bit6-bit0）：当前包序号，从 0 开始至包总数减 1 YY:（bit7- bit4）：板载序号（0-2） （bit3-bit0）:负载数据区的有效数据 ZZ：负载数据 CRC: 0xFF&（byte0+..+byte18） 例：写板载 1 配置（profile 表） Set_report: 13 040a00 1e xx(14byte 的有效数据) crc IN：（成功） 13 040a00 1e xx(14byte 的有效数据) crc Set_report: 13 040a011e xx(14byte 的有效数据) crc IN：（失败，驱动重发当前笔） 13 048a011e xx(14byte 的有效数据) crc Set_report: 13 040a011e xx(14byte 的有效数据) crc IN：（成功） 13 040a011e xx(14byte 的有效数据) crc 。。。 Set_report: 13 040a0912 xx(2byte 的有效数据) 00（2byte 的 0）crc IN：（成功） 13 040a0912 xx(2byte 的有效数据) 00（12byte 的 0）crc 若同一笔失败 10 次，则认为设备出问题驱动报错，不在下传。若发送数据没有等到回复，则过 30ms 再次重发，重发 10 次，一直没回复，驱动报错，不在下发。
13	44	VAL	XX	YY	ZZ	CRC	功能：读配置数据（profile 表） VAL：（bit7 用于上报）=0：下发数据成功 （bit7 用于上报）=1：下发数据失败 （bit6-bit0）：该命令会下发包总数 XX：（bit6-bit0）：当前包序号，从 0 开始至包总数减 1 YY:（bit7- bit4）：板载序号（0-2） （bit3-bit0）:负载数据区的有效数据 ZZ：负载数据 CRC: 0xFF&（byte0+..+byte18） 例：读板载 1 配置（profile 表）

							Set_report: 13 440100 1000(14byte 的 0) crc IN: 13 440a00 1e xx(14byte 的有效数据) crc IN: 13 440a011e xx(14byte 的有效数据) crc 。。。 IN: 13 440a0912 xx(2byte 的有效数据) 00（12byte 的 0）crc Set_report 命令里面的总包数为 1，上传的总包数由设备计算，驱动对数据连续性做校验，如果不连续，等数据上传之后，在重新读取，重复十次失败，驱动报错。
13	05	VAL	00	YY	ZZ	CRC	功能：读取键盘密码或者宏存储空间大小 VAL：(bit6-bit0)：该命令会下发包总数 YY:（bit7）=0：读取密码和设备版本号（驱动决定是否需要读取版本号，不读取版本号则只有 6byte 密码，读取版本号则 6byte 密码+2byte 版本号<高字节在前低字节在后>） （bit7）=1：读取宏空间大小 （bit3-bit0）:负载数据区的有效数据 ZZ：负载数据 CRC: 0xFF&（byte0+..+byte18） 例：读取密码 Set_report: 13 050100 00 00(14byte 的 0) crc IN: 13 050100 06 xx(6byte 的有效数据)00（8byte 的 0）crc 读取密码和版本号 Set_report: 13 050100 00 00(14byte 的 0) crc IN: 13 050100 06 xx(8byte 的有效数据<6byte 密码+2byte 版本号>)00（6byte 的 0）crc 读宏空间，宏预留空间 1K Set_report: 13 050100 8000(14byte 的 0) crc IN: 13 050100 8400 04 00 00 00(10byte 的 0) crc 读回密码错误，驱动不允许打开。宏空间大小取决于 IC 预留的大小，驱动在设置宏的时

							候不允许大于预留空间大小。
13	06	01	00	VAL	00	CRC	功能：键盘复位 VAL：=0：整机复位（profile，按键，灯光） =1：按键复位 =2：灯光复位 CRC: 0xFF&（byte0+..+byte18） 例：整机复位 Set_report: 13 060100 0000(14byte 的 0) crc IN： 13 060100 0000(14byte 的 0) crc 回复完数据之后，设备自动复位（恢复出厂设置）
13	07	01	00	Val	XX	CRC	功能:驱动主动读取设备和 dongle 的连接状态 VAL：set 数据为 0 IN 数据为 1 XX：=0：未连接 =1：连接 CRC: 0xFF&（byte0+..+byte18） 例： 查询到连接 Set_report: 13 07 0100 0000(14byte 的 0) crc IN： 13 0701000101 00(13byte 的 0) crc 查询到未连接 Set_report: 13 07 0100 0000(14byte 的 0) crc IN： 13 070100 0100 00(13byte 的 0) crc
13	88	Val	XX	YY	ZZ	CRC	功能：发送音乐灯效数据（实时更新） Val：（bit6-bit0）：该命令会下发包总数 XX：（bit6-bit0）：当前包序号，从 0 开始至包总数减 1 YY：（bit7-bit4）颜色编码（0-2） （bit3-bit0）负载数据区的有效数据 ZZ：颜色编码不一样，代表的意义不一样 颜色编码 0：每 4byte 对应一个按键颜色，格式为 R,G,B,按键位置，未指定按键为不发光。 颜色编码 1：先是 RGB 的值，接着是按键数量 N，最后是 N 个按键的具体位置，未指定不发光。格式：R,G,B,数量，位置 颜色编码 2：整个键盘同一颜色，数据只有一个 RGB。

							CRC: 0xFF&（byte0+..+byte18） 例：ESC 键亮红色，TAB 键亮绿色 Set_report: 13 88 0100 08ff 00 00 00 00 ff 00 02 00(6byte 的 0) crc 或者 Set_report: 13 88 0100 1aff 00 00 01 00 00 ff 00 01 02 00(4byte 的 0) crc 整个键盘为绿色 Set_report: 13 88 0100 2300 ff 00 00(11byte 的 0) crc
13	09	VAL	XX	YY	ZZ	CRC	功能：写灯效对应颜色数据（每个灯效下有 7 组 RGB 支持修改，即切换当前灯效颜色循环时候显示的颜色），数据格式灯效 0 对应的颜色 RGB,RGB,RGB...(共 7 组)，灯效 1 对应的颜色 RGB,RGB,RGB...(共 7 组) 共灯效 0-22，总数据 23*7*3=483byte。 VAL：（bit7 用于上报）=0：下发数据成功 （bit7 用于上报）=1：下发数据失败 （bit6-bit0）：该命令会下发包总数 XX：（bit6-bit0）：当前包序号，从 0 开始至包总数减 1 YY:（bit7- bit4）：板载序号（0-2） （bit3-bit0）:负载数据区的有效数据 ZZ：负载数据 CRC: 0xFF&（byte0+..+byte18） 例：写板载 1 灯光颜色 Set_report: 13 092300 1e xx(14byte 的有效数据) crc IN：（成功） 13 092300 1e xx(14byte 的有效数据) crc Set_report: 13 0923011e xx(14byte 的有效数据) crc IN：（失败，驱动重发当前笔） 13 09a3011e xx(14byte 的有效数据) crc Set_report: 13 0923011e xx(14byte 的有效数据) crc IN：（成功） 13 0923011e xx(14byte 的有效数据) crc 。。。 Set_report: 13 09232217 xx(7byte 的有效数据) 00（7byte 的 0）crc IN：（成功）

							13 09232217 xx(7byte 的有效数据) 00（7byte 的 0）crc 若同一笔失败 10 次，则认为设备出问题驱动报错，不在下传。若发送数据没有等到回复，则过 30ms 再次重发，重发 10 次，一直没回复，驱动报错，不在下发。
13	49	VAL	XX	YY	ZZ	CRC	功能：灯效对应颜色数据（每个灯效下有 7 组 RGB 支持修改，即切换当前灯效颜色循环时候显示的颜色），数据格式灯效 0 对应的颜色 RGB,RGB,RGB...(共 7 组)，灯效 1 对应的颜色 RGB,RGB,RGB...(共 7 组) 共灯效 0-22，总数据 23*7*3=483byte。 VAL：（bit7 用于上报）=0：下发数据成功 （bit7 用于上报）=1：下发数据失败 （bit6-bit0）：该命令会下发包总数 XX：（bit6-bit0）：当前包序号，从 0 开始至包总数减 1 YY:（bit7- bit4）：板载序号（0-2） （bit3-bit0）:负载数据区的有效数据 ZZ：负载数据 CRC: 0xFF&（byte0+..+byte18） 例：读板载 1 颜色表 Set_report: 13 490100 1000(14byte 的 0) crc IN： 13 492300 1e xx(14byte 的有效数据) crc IN： 13 4923011e xx(14byte 的有效数据) crc 。。。 IN： 13 49232217 xx(7byte 的有效数据) 00（7byte 的 0）crc Set_report 命令里面的总包数为 1，上传的总包数由设备计算，驱动对数据连续性做校验，如果不连续，等数据上传之后，在重新读取，重复十次失败，驱动报错。
13	4a	01	00	XX	YY, ZZ,00	CRC	功能：读电量信息 XX:（bit3-bit0）:负载数据区的有效数据 YY：电池剩余电量 ZZ：（bit7-bit4）=0：未充电 =1：正在充电 （bit3-bit0）=0：未充满 （bit3-bit0）=1：已充满 CRC: 0xFF&（byte0+..+byte18） 例：充电中电池电量 75

							Set_report: 13 4a0100 0000(14byte 的 0) crc IN: 13 4a0100 024b 10 00(12byte 的 0) crc
13	0b	01	00	XX	YY	CRC	功能：发送屏幕显示信息 XX:（bit3-bit0）:负载数据区的有效数据 YY：第一个 byte 是系统音量 第二个 byte 是 cpu 利用率 第三个 byte 是内存使用率 第四个 byte 是年低字节 第五个 byte 是年高字节 第六个 byte 是月 第七个 byte 是日 第八个 byte 是时 第九个 byte 是分 第十个 byte 是秒 第十一个 byte 是周几 CRC: 0xFF&（byte0+..+byte18） 例：驱动发送音量 20，cpu 使用率 50，内存 50，2024 年 3 月 20 日 19 点 20 分 30 秒星期三 Set_report: 13 0b0100 0b14 32 32 18 14 03 14 13 14 1e 03 00(3byte 的 0) crc IN: 13 0b0100 0b14 32 32 18 14 03 14 13 14 1e 03 00(3byte 的 0) crc
13	8e	Val	XX	YY	ZZ	CRC	功能：发送周边灯音乐灯效数据（实时更新） Val：（bit6-bit0）：该命令会下发包总数 XX：（bit6-bit0）：当前包序号，从 0 开始至包总数减 1 YY：（bit7-bit4）颜色编码（0-2） （bit3-bit0）负载数据区的有效数据 ZZ：颜色编码不一样，代表的意义不一样 颜色编码 0：每 4byte 对应一个按键颜色，格式为 R,G,B,按键位置，未指定按键为不发光。 颜色编码 1：先是 RGB 的值，接着是按键数量 N，最后是 N 个按键的具体位置，未指定不发光。格式：R,G,B,数量，位置 颜色编码 2：整个键盘同一颜色，数据只有一个 RGB。 CRC: 0xFF&（byte0+..+byte18） 例：第一个灯键亮红色，第三个灯键亮绿色 Set_report:

							13 8e 0100 08ff 00 00 00 00 ff 00 02 00(6byte 的 0) crc 或者 Set_report: 13 8e 0100 1aff 00 00 01 00 00 ff 00 01 02 00(4byte 的 0) crc 整个灯条为绿色 Set_report: 13 8e 0100 2300 ff 00 00(11byte 的 0) crc
13	10	01	00	XX	YY	CRC	功能：设置当前板载 XX:（bit3-bit0）:负载数据区的有效数据 YY：当前板载（0-2） CRC: 0xFF&（byte0+..+byte18） 例：设置当前板载 1 Set_report: 13 1001000101 00(13byte 的 0) crc IN： 13 1001000101 00(13byte 的 0) crc
13	11	01	00	XX	YY	CRC	功能：读当前板载 XX:（bit3-bit0）:负载数据区的有效数据 YY：当前板载（0-2） CRC: 0xFF&（byte0+..+byte18） 例：读取当前板载，当前板载 1 Set_report: 13 110100 0000(14byte 的 0) crc IN： 13 1101000101 00(13byte 的 0) crc
13	71	01	00	XX	YY	CRC	功能：下发用户自定义数据，用于设备的区别（产品序列号，产品名称等，这部分数据升程序后也不允许修改）。 XX:（bit3-bit0）:负载数据区的有效数据 YY：有效数据 CRC: 0xFF&（byte0+..+byte18） 例：设置产品序列号 03 05 08 Set_report: 13 7101000303 05 08 00(11byte 的 0) crc IN：（成功） 13 7101000303 05 08 00(11byte 的 0)
13	72	01	00	XX	YY	CRC	功能：读取用户自定义数据，用于设备的区别（产品序列号，产品名称等，这部分数据升程序后也不允许修改）。 XX:（bit3-bit0）:负载数据区的有效数据 YY：有效数据 CRC: 0xFF&（byte0+..+byte18） 例：读产品序列号，有效长度为 3

							Set_report: 13 720100 0000(14byte 的 0) crc IN: 13 720100 03xx(3byte 有效数据) 00(11byte 的 0) crc

附：自定义描述符

```
    0x06,   0x02,   0xff,                  // Usage Page (Vendor-Defined 3)
    0x09,   0x02,                          // Usage (Vendor-Defined 2)
    0xa1,   0x01,                          // Collection (Application)
    0x85,   0x08, // report ID:8
    0x15,   0x00,                          // Logical Minimum (0)
    0x26,   0xff, 0x00,                    // Logical Maximum (255)
0x75,   0x08,                              // Report Size (8)
    0x95,   0x13,// Report Count (19)
    0x09,   0x02,                          // Usage (Vendor-Defined 2)
    0x81,   0x00,                          // Input (Data,Ary,Abs)
    0x09,   0x02,                          // Usage (Vendor-Defined 2)
    0x91,   0x00,                          // Output (Data,Ary,Abs,NWrp,Lin,Pref,NNul,NVol,Bit)
    0xc0,                                  // End Collection
```

Profile 格式

```
const U8 code PROFILE_DATA[PROFILE_SIZE]=
{
 //128byte
  0,//CurrentProfile   0-1  当前 profile，
  3,//CurrentReportRate:0~3
  3,//CurrentRepeat:0~3
  0,//CurrentDebounce:0~3     去抖次数
  0,//CurrentMacroMode
  0,// led 亮度速度颜色分开还是统一控制，0 为分开，1 为统一
  19,//统一控制的亮度
  0x04,// 统一控制的速度
  0x0b,// 统一控制的颜色
  0,//选择当前 LED 模式是否是游戏模式，非 0 为游戏模式，0 为其他模式
  1,//LED 模式
  0x20,//led 游戏模式
  0,//profile0 无线通道
  0,// profile1 无线通道
  0,//键盘处于有线还是无线模式，0 为有线，1 为 2.4，2 为蓝牙
  0,//WIN 锁
```

0,//按键锁
0,//WASD 交换
0,//当前 logo 灯模式
0,// logo 灯颜色
0,// logo 灯亮度
0,// logo 灯速度
0,//tap 层 delay 时间，0 为关闭，后 7 位为有效 DELAY 时间
0,//保留
10,// 休眠时间 例 1==30 秒，默认 10 为 5 分钟，0 是不休眠
0,// 保留
0,// 保留
0,// 保留
0,// 保留
0,// 保留
0,// 保留
0,// 保留
0,// 保留
0,// 保留
0,// 保留
0,// 保留
0,// 保留
0,// 保留
0, // 保留
0, // 保留
0, // 保留
0, // 保留
0, // 保留
0, // 保留
0, // 保留
0, // 保留
0,// 保留
0, // 保留
0, // 保留
0, // 保留
0, // 保留
0, // 保留
0, // 保留
0, // 保留
0, // 保留
0, // 保留
//2 个 byte 表示颜色亮度速度

```
//前面一个 byte 为亮度（亮度共 20 级 0-19），后一个 byte 低 4bit 颜色，高 4bit 为速度
  Bin(11111111),//KEY_MODE 0      ¹ØµÆ
  Bin(11111111),//KEY_MODE 0      ¹ØµÆ
  Bin(00010011),//KEY_MODE 1   ¹âÆ×Ñ-»·
  Bin(01000111),//KEY_MODE 1
  Bin(00010011),//KEY_MODE 1   ¹âÆ×Ñ-»·
  Bin(01000111),//KEY_MODE 2
  Bin(00010011),//KEY_MODE 1   ¹âÆ×Ñ-»·
  Bin(01000111),//KEY_MODE 3
  Bin(00010011),//KEY_MODE 1   ¹âÆ×Ñ-»·
  Bin(01000111),//KEY_MODE 4
  Bin(00010011),//KEY_MODE 1   ¹âÆ×Ñ-»·
  Bin(01000111),//KEY_MODE 5
  Bin(00010011),//KEY_MODE 1   ¹âÆ×Ñ-»·
  Bin(01000111),//KEY_MODE 6
  Bin(00010011),//KEY_MODE 1   ¹âÆ×Ñ-»·
  Bin(01000111),//KEY_MODE 7
  Bin(00010011),//KEY_MODE 1   ¹âÆ×Ñ-»·
  Bin(01000111),//KEY_MODE 8
  Bin(00010011),//KEY_MODE 1   ¹âÆ×Ñ-»·
  Bin(01000111),//KEY_MODE 9
  Bin(00010011),//KEY_MODE 1   ¹âÆ×Ñ-»·
  Bin(01000111),//KEY_MODE 10
  Bin(00010011),//KEY_MODE 1   ¹âÆ×Ñ-»·
  Bin(01000111),//KEY_MODE 11
  Bin(00010011),//KEY_MODE 1   ¹âÆ×Ñ-»·
  Bin(01000111),//KEY_MODE 12
  Bin(00010011),//KEY_MODE 1   ¹âÆ×Ñ-»·
  Bin(01000111),//KEY_MODE 13
  Bin(00010011),//KEY_MODE 1   ¹âÆ×Ñ-»·
  Bin(01000111),//KEY_MODE 14
  Bin(00010011),//KEY_MODE 1   ¹âÆ×Ñ-»·
  Bin(01000111),//KEY_MODE 15
  Bin(00010011),//KEY_MODE 1   ¹âÆ×Ñ-»·
  Bin(01000111),//KEY_MODE 16     ¹ØµÆ
  Bin(00010011),//KEY_MODE 1   ¹âÆ×Ñ-»·
  Bin(01000111),//KEY_MODE 17
  Bin(00010011),//KEY_MODE 1   ¹âÆ×Ñ-»·
  Bin(01000111),//KEY_MODE 18
  0,    // 保留
  0,    // 保留
```

```
    0,    // 保留
    0,    // 保留
    0,    // 保留
    0,    // 保留
    0,    // 保留
    0,    // 保留
    0,    // 保留
    0,    // 保留
    0,    // 保留
    0,    // 保留
    0,    // 保留
    0,    // 保留
    0,    // 保留
    0,    // 保留
    0,    // 保留
    0,    // 保留
    0,    // 保留
    0,    // 保留
    0,    // 保留
    0,    // 保留

    //¸ß4bitÎªÑÕÉ«Ä£Ê½£¬µÍ4bitÎ»ÁÁ¶È
    Bin(00010011),//KEY_MODE 1   MOBA
    Bin(00010011),//KEY_MODE 2   MOBA
    Bin(00010011),//KEY_MODE 3   MOBA
    Bin(00010011),//KEY_MODE 4   MOBA
    Bin(00010011),//KEY_MODE 5   MOBA
    Bin(00010011),//KEY_MODE 6   MOBA
    Bin(00010011),//KEY_MODE 7   MOBA
    Bin(00010011),//KEY_MODE 8   MOBA
    Bin(00010011),//KEY_MODE 9   MOBA
    Bin(00010011),//KEY_MODE 10 MOBA  90
    0x5a,//±£Áô
    0xa5//±£Áô
};
```