AVLdiy.cn - WBOX 控制协议说明（Release1.2）

固件版本大于 V3.5.1.102 才支持此协议　_by hahan

第一章：RAW 数据协议说明

```
//-------------------------------------------------------------------------
网络传输协议为 UDP：
通过发 UDP 包控制灯光，盒子相当于一台服务器，UDP 端口必须是：(0x1939)6457
//-------------------------------------------------------------------------

//----------------控制数据协议格式-----------------------------------------
//--------数据长度： BYTE：为字节[1Byte]  WORD：双字节[2Byte] ---------
//------------ 字节顺序为小端模式 ---------------------------------------
struct my_dmx {
    BYTE id[10];            //0～9     盒子协议头 AVLdiy.cn0 (固定值，不变)
    WORD Command;           //10～11   控制命令，不同命令见下 DMX 协议命令说明
    WORD Length;            //12～13   发送 DMX 数据的长度
    WORD Universe;          //14～15   哪个输出口 PORT0～PORT7，如果多台盒子组合，设置方法请参考教学视频 universe 的讲解
    WORD Channel;           //16～17   CC/CD 命令需要指定哪个通道(0～511)，从 0 开始计，注意不是(1～512)
    BYTE DmxData_val_Start; //18       功能 1:[CB/CC/CD 命令需要指定 xxx 值（0～255）]， 功能 2:[发整包数据时 dmx 填写的起始位值]
};
//-------------------------------------------------------------

//-------------------------------------------------------------------------------------------------
DMX 控制协议命令：
Command = 0x0100:       //整包 512 通道 dmx 数据发到指定输出口（可以指定到输出口 PORT0～PORT7）
Command = 0x0200:       //整包 512 通道 dmx 数据发到所有输出口（PORT0-PORT7 端口的数据是一样的）
Command = 0xCA00:       //把指定输出口的全部 512 通道变为 xxx 值（xxx 为 0～255 指定的值）
Command = 0xCB00:       //把每个输出口（PORT0～PORT7）的全部 512 通道变为 xxx 值（xxx 为 0～255 指定的值）
```

```
Command = 0xCC00:       //把指定输出口的某个通道变为 xxx 值（xxx 为 0～255 指定的值）
Command = 0xCD00:       //把所有输出口 PORT0～PORT7 的某个通道变为 xxx 值（xxx 为 0～255 指定的值）
Command = 0xCE00:       //把指定输出口的某个地址，连续赋 16 通道的 dmx 值
Command = 0xC100:       //LED 专用命令，把指定输出口的全部红灯打开，亮度为 xxx 值（xxx 为 0～255 指定的值）
Command = 0xC200:       //LED 专用命令，把指定输出口的全部绿灯打开，亮度为 xxx 值（xxx 为 0～255 指定的值）
Command = 0xC300:       //LED 专用命令，把指定输出口的全部蓝灯打开，亮度为 xxx 值（xxx 为 0～255 指定的值）
Command = 0xB100:       //LED 专用命令，把指定输出口的全部红灯打开，和 0xC100 命令不同是会清除原先的值
Command = 0xB200:       //LED 专用命令，把指定输出口的全部绿灯打开，和 0xC200 命令不同是会清除原先的值
Command = 0xB300:       //LED 专用命令，把指定输出口的全部蓝灯打开，和 0xC300 命令不同是会清除原先的值
Command = 0xB400:       //LED 专用命令，把指定输出口的全部通道变成 RGB 值
Command = 0xFEFE:       //检测盒子是否开机在线
Command = 0xXXXX:       //盒子恢复到出厂设置（危险命令，需要此命令的可以联系我们）
//-------------------------------------------------------------------------------------------------------------
```

```
//#############################################################################################################
```

例子 1：

```
//这个例子演示指定输出端口 512 通道值，每一个通道都可以改变，需要定义至少 512+20 个字节的 buf

    unsigned char b[600];  //定义一个 buff，如果没有这么多空间，只需要定义 512+20bytes
    b[0]='A';   //我们的 id[0]
    b[1]='V';
    b[2]='L';
    b[3]='d';
    b[4]='i';
    b[5]='y';
    b[6]='.';
    b[7]='c';
    b[8]='n';
    b[9]=0;     //id[10]  end
```

```
    b[10]= 0x00;     //command 低位字节 （comm=0x0100 整包 512 通道 dmx 数据发到指定输出口）
    b[11]= 0x01;     //command 高位字节 （注意网络字节顺序的高低位）
    b[12]= 0x00;     //Length 低位 盒子默认 512 通道(0x0200=512)，如果不填，0x0000 缺省是 512 通道
    b[13]= 0x02;     //Length 高位， 0x0200 = 512
    b[14]= 0～7;     //输出口 port 低位，指定端口的 universe 值
    b[15]= 0;        //输出口 port 高位，如果输出端口大于 255，就需要进位，转换成 2 字节，填这里，缺省 0
    b[16]= 0;        //Channel 低位， 这条命令下无效
    b[17]= 0;        //Channel 高位， 这条命令下无效
    b[18]= xxx;      //0～255，第一个推子的高度   DMXData[0]
    b[19]= xxx;      //0～255，第 2 个推子的高度   DMXData[1]
    b[20]= xxx;      //0～255，第 3 个推子的高度   DMXData[2]
      .
      .
    b[18+512]= xxx;      //0～255，第 512 个推子的高度 DMXData[511]
    UDPSend(b,532);      //这个就是发 udp 包函数，把上面的 532 字节发出去，盒子就会变化，注意，发 udp 必须对应端口(0x1939)6457

//如果是编程语言，会用 memcpy()函数，不会自己一个一个字节填 512 通道，直接用下面 copy 语句
//memcpy( &(header->DmxData_val_Start),DMXData,512);
//技巧：真正要改变的是 DMXData[512]数组里面每个通道的值，其它值可以固定，可以用一个定时器每 30ms 发一次这个包

//-------------------------------------------------------------------------------
中控 16 进制发码(你要发的是这串 16 进制，通过 UDP 把这串 16 进制发到端口(0x1939)6457):
                              41 56 4C 64 69 79 2E 63 6E 00 00 01 00 00 xx xx 00 00 xx xx xx ... xx
真正要改的是这个 xxxx 值:      41 56 4C 64 69 79 2E 63 6E 00 00 01 00 00 xx xx 00 00 xx xx xx ... xx
解释：                           固定的命令头                  命令  长度   端口      推子 1 推子 2 ...推子 512
解释：                           A  V  L  d  i  y  .  c  n  00 00 01 00 00 xx xx 00 00 xx   xx   ... xx
自己写程序：                   b[0] b[1] .....                                          b[18] b[19] ...b[18+512]

注意： AVLdiy.cn 转换成 16 进制就是：41 56 4C 64 69 79 2E 63 6E
//-------------------------------------------------------------------------------
```

```
//#########################################################################################################################
```

例子 2：

```
//这个例子演示全部 PORT0～PORT7 输出口输出相同的 DMXData[512]值，每一个通道都可以改变，需要定义至少 512+20 个字节的 buf
    unsigned char b[600];  //定义一个 buff，如果没有这么多空间，只需要定义 512+20byte
    b[0]='A';        //我们的 id[0]
    b[1]='V';
    b[2]='L';
    b[3]='d';
    b[4]='i';
    b[5]='y';
    b[6]='.';
    b[7]='c';
    b[8]='n';
    b[9]=0;          //id[9]  end
    b[10]= 0x00;     //command 低字节（comm=0x0200 整包 512 通道 dmx 数据发到所有输出口，全部输出口内容相同）
    b[11]= 0x02;     //command 高字节（注意网络字节顺序的高低位）
    b[12]= 0x00;     //Length 低位，盒子默认 512 通道(0x0200=512)，如果不填，0x0000 缺省是 512 通道
    b[13]= 0x02;     //Length 高位，  0x0200 = 512
    b[14]= 0;        //输出口 port 低位，这条命令下无效，缺省为全部 8 个口同时输出相同值，可以为 0
    b[15]= 0;        //输出口 port 高位，这条命令下无效，缺省为全部 8 个口同时输出相同值，可以为 0
    b[16]= 0;        //Channel 低位，  这条命令下无效，可以为 0
    b[17]= 0;        //Channel 高位，  这条命令下无效 ，可以为 0
    b[18]= xxx;      //0～255，第一个推子的高度    DMXData[0]
    b[19]= xxx;      //0～255，第 2 个推子的高度    DMXData[1]
    b[20]= xxx;      //0～255，第 3 个推子的高度    DMXData[2]
      .
      .
    b[18+512]= xxx;      //0～255，第 512 个推子的高度 DMXData[511]
    UDPSend(b,532); //这个就是发 udp 包函数，把上面的 532 字节发出去，盒子就会变化，注意，发 udp 必须对应端口(0x1939)6457
```

//如果是编程语言，会用 memcpy()函数，不会自己一个一个字节填 512 通道，直接用下面 copy 语句
//memcpy(&(header->DmxData_val_Start),DMXData,512);

//---
中控 16 进制发码(你要发的是这串 16 进制，通过 UDP 把这串 16 进制发到端口(0x1939)6457):

41 56 4C 64 69 79 2E 63 6E 00 00 02 00 00 00 00 00 00 xx xx xx ... xx

真正要改的是这个 xxxx 值: 41 56 4C 64 69 79 2E 63 6E 00 00 02 00 00 00 00 00 00 xx xx xx ... xx

解释: 固定的命令头 命令 长度 PORT CH 推子 1 推子 2 ...推子 512

解释: A V L d i y . c n 00 00 02 00 00 00 00 00 00 xx xx ... xx

自己写程序: b[0] b[1] ..... b[18] b[19] ...b[18+512]

//---

//###

例子 3：

//这个例子演示指定某输出口的 512 通道全部输出 xxx 的值，相当于让这个输出口的 512 个推子全部推到相同位值（0～255）

```
    unsigned char b[600];  //定义一个 buff，如果没有这么多空间，只需要定义 20byte
    b[0]='A';   //盒子固定头 id[0]
    b[1]='V';
    b[2]='L';
    b[3]='d';
    b[4]='i';
    b[5]='y';
    b[6]='.';
    b[7]='c';
    b[8]='n';
    b[9]=0;     //id[9]  end

    b[10]= 0x00;    //command 低字节 （comm=0xCA00 把指定输出口的全部 512 通道变为 xxx 值）
    b[11]= 0xCA;    //command 高字节 （注意网络字节顺序的高低位）
```

```
    b[12]= 0;        //Length 低位，这条命令下无效，缺省盒子默认 512，可以缺省 0
    b[13]= 0;        //Length 高位，这条命令下无效，缺省盒子默认 512，可以缺省 0
    b[14]= 0～7;     //输出口 port 低位，指定端口的 universe 值，盒子缺省是 0～7
    b[15]= 0;        //输出口 port 高位，如果输出端口大于 255，就需要进位，转换成 2 字节，填这里，缺省 0
    b[16]= 0;        //Channel 低位，这条命令下无效，缺省 0
    b[17]= 0;        //Channel 高位，这条命令下无效，缺省 0
    b[18]= 0～255;  //xxx 值 0～255，就是推子的高度（技巧：可以把这个值改为 0 和 255 间隔发出去，就可以检测灯是不是受控）

    UDPSend(b,20); //这个就是发 udp 包函数，把上面的 20 字节发出去，盒子就会变化，注意，发 udp 必须对应端口(0x1939)6457

//-----------------------------------------------------------------
如果是中控，需要 16 进制发码，把上面数据翻译成 16 进制，你要发的是这串 16 进制，通过 UDP 把这串 16 进制发到端口(0x1939)6457 :
                                41 56 4C 64 69 79 2E 63 6E 00 00 CA 00 00 xx 00 00 00 xx
真正要改的是这个 xxx 值:          41 56 4C 64 69 79 2E 63 6E 00 00 CA 00 00 xx 00 00 00 xx
解释：                              固定的命令头                  命令       port       推子高度 0～255
解释：                             A  V  L  d  i  y  .  c  n  00 00 CA 00 00 xx xx 00 00 xx
自己写程序：                      b[0] b[1] ......                                    b[18]

注意： AVLdiy.cn 转换成 16 进制就是：41 56 4C 64 69 79 2E 63 6E
//-----------------------------------------------------------------


//#################################################################################################################
例子 4：
//这个例子演示全部输出口的 512 通道全部输出 xxx 的值，相当于全部输出口的全部 512 推子推到相同位值 xxx（0～255）

    unsigned char b[600];  //定义一个 buff，如果没有这么多空间，只需要定义 20byte

    b[0]='A';   //我们的 id[0]
    b[1]='V';
```

```
    b[2]='L';
    b[3]='d';
    b[4]='i';
    b[5]='y';
    b[6]='.';
    b[7]='c';
    b[8]='n';
    b[9]=0;      //id[9]  end

    b[10]= 0x00;    //command 高字节 （comm=0xCB00 所有输出口的全部 512 通道变为 xxx 值）
    b[11]= 0xCB;    //command 低字节 （注意网络字节顺序的高低位）
    b[12]= 0;       //Length 高位 这条命令下无效 缺省 盒子默认 512，可以缺省 0
    b[13]= 0;       //Length 低位 这条命令下无效 缺省 盒子默认 512，可以缺省 0
    b[14]= 0;       //输出口 port 高位，这条命令下无效 缺省为全部 8 个口同时输出相同值，可以为 0
    b[15]= 0;       //输出口 port 低位，这条命令下无效 缺省为全部 8 个口同时输出相同值，可以为 0
    b[16]= 0;       //Channel 推子 100 （从 0 开始算，99 就是第 100 个推子）
    b[17]= 0;       //Channel 如果大于 255 以上的值，就需要进位
    b[18]= 0～255;  //0～255，就是推子的高度（技巧：可以把这个值改为 0 和 255 间隔发出去，就可以检测灯是不是受控）
                    //关灯走人技巧：可以把这个值改为 0 发出去，就可以把全部输出口上的设备关闭
    UDPSend(b,20); //这个就是发 udp 包函数，把上面的 20 字节发出去，盒子就会变化，注意，发 udp 必须对应端口(0x1939)6457

//-----------------------------------------------------------------------
中控 16 进制发码(你要发的是这串 16 进制，通过 UDP 把这串 16 进制发到端口(0x1939)6457:
                              41 56 4C 64 69 79 2E 63 6E 00 00 CB 00 00 00 00 00 00 xx
真正要改的是这个 xxxx 值:      41 56 4C 64 69 79 2E 63 6E 00 00 CB 00 00 00 00 00 00 xx
解释：                          固定的命令头                  命令                    推子高度 0～255
解释：                          A  V  L  d  i  y  .  c  n  00 00 CB 00 00 00 00 00 00 xx
自己写程序：                  b[0] b[1] ......                                      b[18]
//-----------------------------------------------------------------------
```

//###

例子 5：

//这个例子演示指定某输出口的某个通道为 xxx 的值，相当于让这个输出口的某个推子推到 0～255

//以下演示输出口 1 的第 100 推子，推到 128 位置

```
unsigned char b[600];  //定义一个 buff，如果没有这么多空间，只需要定义 20byte

b[0]='A';   //盒子固定头 id[0]
b[1]='V';
b[2]='L';
b[3]='d';
b[4]='i';
b[5]='y';
b[6]='.';
b[7]='c';
b[8]='n';
b[9]=0;     //id[9]  end

b[10]= 0x00;    //command 低位字节 （comm=0xCC00 把指定输出口的某个通道变为 xxx 值（0～255）
b[11]= 0xCC;    //command 高位字节
b[12]= 0;       //Length 低位，这条命令下无效，可以缺省 0
b[13]= 0;       //Length 高位，这条命令下无效，可以缺省 0
b[14]= 1;       //输出口 port 低位，指定端口的 universe 值（盒子默认是 0～7 输出口）
b[15]= 0;       //输出口 port 高位，如果输出端口大于 255，转换成 2 字节，高位填这里，缺省 0
b[16]= 99(0x63);//Channel 低位，推子 100 （从 0 开始算，99 就是第 100 个推子）
b[17]= 0;       //Channel 高位，如果大于 255 以上的值，就需要进位，转换成 2 字节，高位填这里
b[18]= 128(0x80);//xxx 值 0～255，就是推子的高度，128 的 16 进制是 0x80
UDPSend(b,20);  //这个就是发 udp 包函数，把上面的 20 字节发出去，盒子就会变化，注意，发 udp 必须对应端口(0x1939)6457
```

//---

如果是中控，需要 16 进制发码，把上面数据翻译成 16 进制，你要发的是这串 16 进制，通过 UDP 把这串 16 进制发到端口(0x1939)6457 ：

							41	56	4C	64	69	79	2E	63	6E	00	00	CC	00	00	01	00	63	00			80
真正要改的是这个 xxx 值:							41	56	4C	64	69	79	2E	63	6E	00	00	CC	00	00	xx	00	xx	xx		xx
解释：							固定的命令头												命令			port		通道 0~511		推子高度 0～255
解释：							A	V	L	d	i	y	.	c	n	00	00	CC	00	00	01	00	63	00			80
自己写程序：							b[0]	b[1]	.....																		b[18]

注意： AVLdiy.cn 转换成 16 进制就是：41 56 4C 64 69 79 2E 63 6E

//---

//###

例子 6：

//这个例子演示所有输出口的某个通道为 xxx 的值，相当于让全部输出口 PORT0～PORT7 的某个推子推到（0～255）

//以下演示全部输出口的第 200 推子，推到 255 位置

```
    unsigned char b[600];  //定义一个 buff，如果没有这么多空间，只需要定义 20byte

    b[0]='A';   //盒子固定头 id[0]
    b[1]='V';
    b[2]='L';
    b[3]='d';
    b[4]='i';
    b[5]='y';
    b[6]='.';
    b[7]='c';
    b[8]='n';
    b[9]=0;     //id[9]  end
```

```
    b[10]= 0x00;          //command 高位字节 （comm=0xCD00 把所有输出口 PORT0～PORT7 的某个通道变为 xxx 值）
    b[11]= 0xCD;          //command 低位字节
    b[12]= 0;             //Length 低位，这条命令下无效，可以缺省 0
    b[13]= 0;             //Length 高位，这条命令下无效，可以缺省 0
    b[14]= 0;             //输出口 port 低位，这条命令下无效，缺省为全部 8 个口同时输出相同值，可以为 0
    b[15]= 0;             //输出口 port 高位，这条命令下无效，缺省为全部 8 个口同时输出相同值，可以为 0
    b[16]= 199(0xC7);     //Channel 低位，推子 200 （从 0 开始算，199 就是第 200 个推子）
    b[17]= 0;             //Channel 高位，如果大于 255 以上的值，就需要进位填这里
    b[18]= 255(0xFF);     //xxx 值 0～255，就是推子的高度
    UDPSend(b,20); //这个就是发 udp 包函数，把上面的 20 字节发出去，盒子就会变化，注意，发 udp 必须对应端口(0x1939)6457

//-------------------------------------------------------------------
如果是中控，需要 16 进制发码，把上面数据翻译成 16 进制，你要发的是这串 16 进制，通过 UDP 把这串 16 进制发到端口(0x1939)6457 ：
                              41 56 4C 64 69 79 2E 63 6E 00 00 CD 00 00 00 00 C7 00 FF
真正要改的是这个 xxx 值:          41 56 4C 64 69 79 2E 63 6E 00 00 CD 00 00 00 00 xx xx xx
解释：                              固定的命令头                  命令              通道   推子高度 0～255
解释：                              A  V  L  d  i  y  .  c  n  00 00 CD 00 00 00 00 C7 00 FF
自己写程序：                       b[0] b[1] ......                                      b[18]

注意： AVLdiy.cn 转换成 16 进制就是：41 56 4C 64 69 79 2E 63 6E
//-------------------------------------------------------------------


//###################################################################################################################
例子 7：
//这个例子演示某个输出口的从某个地址开始，连续 16 通道进行赋值为 xxx 值（0～255）
//以下演示输出口 2 从第 65 个推子开始，连续 16 个推子推到 255 位置

    unsigned char b[600];  //定义一个 buff，如果没有这么多空间，只需要定义 36byte
    b[0]='A';   //盒子固定头 id[0]
```

```
b[1]='V';
b[2]='L';
b[3]='d';
b[4]='i';
b[5]='y';
b[6]='.';
b[7]='c';
b[8]='n';
b[9]=0;     //id[9]  end
b[10]= 0x00;        //command 低位字节 （comm=0xCE00 把指定输出口的某个地址，连续赋 16 通道的 dmx 值）
b[11]= 0xCE;        //command 高位字节
b[12]= 0;           //Length 低位，这条命令下无效，可以缺省 0
b[13]= 0;           //Length 高位，这条命令下无效，可以缺省 0
b[14]= 2;           //输出口 port 低位，例子填 2
b[15]= 0;           //输出口 port 高位
b[16]= 64(0x40);    //Channel 低位，推子 65 （从 0 开始算，65 就是第 64 个推子）
b[17]= 0;           //Channel 高位，如果大于 255 以上的值，就需要进位填这里
b[18]= 255(0xFF);   //xxx 值 0～255，就是推子的高度
b[19]= 255(0xFF);   //xxx 值 0～255，就是推子的高度
b[20]= 255(0xFF);   //xxx 值 0～255，就是推子的高度
b[21]= 255(0xFF);   //xxx 值 0～255，就是推子的高度
b[22]= 255(0xFF);   //xxx 值 0～255，就是推子的高度
b[23]= 255(0xFF);   //xxx 值 0～255，就是推子的高度
.
.
.
b[33]= 255(0xFF);   //xxx 值 0～255，就是推子的高度
b[34]= 255(0xFF);   //xxx 值 0～255，就是推子的高度

UDPSend(b,36); //这个就是发 udp 包函数，把上面的 36 字节发出去，盒子就会变化，注意，发 udp 必须对应端口(0x1939)6457
```

//---

如果是中控，需要 16 进制发码，把上面数据翻译成 16 进制，你要发的是这串 16 进制，通过 UDP 把这串 16 进制发到端口(0x1939)6457 ：

41 56 4C 64 69 79 2E 63 6E 00 00 CE 00 00 02 00 40 00 FF FF FF FF FF FF … FF(16 个 FF)

真正要改的是这个 xxx 值: 41 56 4C 64 69 79 2E 63 6E 00 00 CE 00 00 xx 00 XX XX xx xx xx xx xx xx … xx(16 个 xx)

解释: 固定的命令头 命令 PORT 地址 推子高度 0～255(连续 16 个)

解释: A V L d i y . c n 00 00 CE 00 00 02 00 40 00 FF FF FF FF FF ……

自己写程序: b[0] b[1] ..... b[18]

注意： AVLdiy.cn 转换成 16 进制就是：41 56 4C 64 69 79 2E 63 6E

//---

//###

例子 8：

//这个例子演示 LED 灯 3 通道专用命令，指定某输出口的 512 通道里面全部 RGB 灯的变化，对应命令 0xC100，0xC200，0xC300，0xB100，0xB200，0xB300

0xC100，0xC200，0xC300 不会改变其他通道的值，会叠加原先的颜色值； 0xB100，0xB200，0xB300 会清除原先的颜色值，然后变成单色

```
unsigned char b[600];  //定义一个 buff，如果没有这么多空间，只需要定义 20byte
b[0]='A';   //盒子固定头 id[0]
b[1]='V';
b[2]='L';
b[3]='d';
b[4]='i';
b[5]='y';
b[6]='.';
b[7]='c';
b[8]='n';
b[9]=0;     //id[9]  end
b[10]= 0x00;    //command 低字节 （RGB 颜色变化命令，把指定输出口的全部 512 通道 RGB 变为 xxx 值）
b[11]= 0xC1;    //command 高字节 （C1 红色，C2 绿色，C3 蓝色 / B1 红色，B2 绿色，B3 蓝色）
b[12]= 0;       //Length 低位，这条命令下无效，缺省盒子默认 512，可以缺省 0
```

```
    b[13]= 0;        //Length 高位，这条命令下无效，缺省盒子默认 512，可以缺省 0
    b[14]= 0～7;     //输出口 port 低位，指定端口的 universe 值，盒子缺省是 0～7
    b[15]= 0;        //输出口 port 高位，如果输出端口大于 255，就需要进位，转换成 2 字节，填这里，缺省 0
    b[16]= 0;        //Channel 低位，这条命令下无效，缺省 0
    b[17]= 0;        //Channel 高位，这条命令下无效，缺省 0
    b[18]= 0～255;   //xxx 值 0～255，就是推子的高度（颜色的亮度值）

    UDPSend(b,20); //这个就是发 udp 包函数，把上面的 20 字节发出去，盒子就会变化，注意，发 udp 必须对应端口(0x1939)6457

//-----------------------------------------------------------------
如果是中控，需要 16 进制发码，把上面数据翻译成 16 进制，你要发的是这串 16 进制，通过 UDP 把这串 16 进制发到端口(0x1939)6457 ：
                              41 56 4C 64 69 79 2E 63 6E 00 00 C1 00 00 xx 00 00 00 xx
真正要改的是这个 xxx 值:          41 56 4C 64 69 79 2E 63 6E 00 00 xx 00 00 xx 00 00 00 xx
解释：                              固定的命令头                  命令        port        亮度 0～255
解释：                             A  V  L  d  i  y  .  c  n  00 00 C1 00 00 xx 00 00 00 xx
自己写程序：                       b[0] b[1] ......                                    b[18]

注意： AVLdiy.cn 转换成 16 进制就是：41 56 4C 64 69 79 2E 63 6E
//-----------------------------------------------------------------


//#########################################################################################################################
例子 9：
//这个例子演示 LED 灯 3 通道专用命令，指定某输出口的全部 512 通道里面 RGB 的值

    unsigned char b[600];  //定义一个 buff，如果没有这么多空间，只需要定义 22byte
    b[0]='A';   //盒子固定头 id[0]
    b[1]='V';
    b[2]='L';
    b[3]='d';
```

```
    b[4]='i';
    b[5]='y';
    b[6]='.';
    b[7]='c';
    b[8]='n';
    b[9]=0;      //id[9]  end
    b[10]= 0x00;    //command 低字节 （comm=0xB400，RGB 颜色变化命令，把指定输出口的全部通道变成 RGB 值）
    b[11]= 0xB4;    //command 高字节
    b[12]= 0;       //Length 低位，这条命令下无效，缺省盒子默认 512，可以缺省 0
    b[13]= 0;       //Length 高位，这条命令下无效，缺省盒子默认 512，可以缺省 0
    b[14]= 0～7;    //输出口 port 低位，指定端口的 universe 值，盒子缺省是 0～7
    b[15]= 0;       //输出口 port 高位，如果输出端口大于 255，就需要进位，转换成 2 字节，填这里，缺省 0
    b[16]= 0;       //Channel 低位，这条命令下无效，缺省 0
    b[17]= 0;       //Channel 高位，这条命令下无效，缺省 0
    b[18]= 0～255;  //R 值 0～255，就是推子的高度（颜色的亮度值），通过 RGB 三基色的值可以组合任何颜色
    b[19]= 0～255;  //G 值 0～255，就是推子的高度（颜色的亮度值），通过 RGB 三基色的值可以组合任何颜色
    b[20]= 0～255;  //B 值 0～255，就是推子的高度（颜色的亮度值），通过 RGB 三基色的值可以组合任何颜色

    UDPSend(b,21); //这个就是发 udp 包函数，把上面的 21 字节发出去，盒子就会变化，注意，发 udp 必须对应端口(0x1939)6457

//-------------------------------------------------------------
如果是中控，需要 16 进制发码，把上面数据翻译成 16 进制，你要发的是这串 16 进制，通过 UDP 把这串 16 进制发到端口(0x1939)6457 :
                              41 56 4C 64 69 79 2E 63 6E 00 00 B4 00 00 xx 00 00 00 xx xx xx
真正要改的是这个 xxx 值:      41 56 4C 64 69 79 2E 63 6E 00 00 B4 00 00 xx 00 00 00 xx xx xx
解释：                          固定的命令头                  命令        port        R  G  B
解释：                          A  V  L  d  i  y  .  c  n  00 00 B4 00 00 xx 00 00 00 xx xx xx
自己写程序：                   b[0] b[1] ......                                        b[18]..b[20]

注意： AVLdiy.cn 转换成 16 进制就是：41 56 4C 64 69 79 2E 63 6E
//-------------------------------------------------------------
```

//###

例子 10：

//这个例子演示检测盒子是否开机在线，把这条命令发到盒子，盒子会返回 OK，就表示盒子开机并且在线。

```
    unsigned char b[12];  //定义一个 buff，只需要定义 12byte
    b[0]='A';        //盒子固定头 id[0]
    b[1]='V';
    b[2]='L';
    b[3]='d';
    b[4]='i';
    b[5]='y';
    b[6]='.';
    b[7]='c';
    b[8]='n';
    b[9]=0;          //id[9]  end
    b[10]= 0xFE;     //command 低字节 （comm=0xFEFE，检测盒子是否开机在线）
    b[11]= 0xFE;     //command 高字节
    UDPSend(b,12);  //把上面的 12 字节发出去，盒子就会返回字符 OK
```

//---

如果是中控，需要 16 进制发码，把上面数据翻译成 16 进制，你要发的是这串 16 进制，通过 UDP 把这串 16 进制发到端口(0x1939)6457 ：

```
                                41 56 4C 64 69 79 2E 63 6E 00 FE FE
固定值:                          41 56 4C 64 69 79 2E 63 6E 00 FE FE
解释：                              固定的命令头                 命令
解释：                          A  V  L  d  i  y  .  c  n  00 FE FE
自己写程序：                     b[0] b[1] ......              b[11]
```

注意： AVLdiy.cn 转换成 16 进制就是：41 56 4C 64 69 79 2E 63 6E

//---

//###

例子 11：

//这个例子演示如何让盒子恢复出厂设置

```
    unsigned char b[12];  //定义一个 buff，只需要定义 12byte
    b[0]='A';        //盒子固定头 id[0]
    b[1]='V';
    b[2]='L';
    b[3]='d';
    b[4]='i';
    b[5]='y';
    b[6]='.';
    b[7]='c';
    b[8]='n';
    b[9]=0;          //id[9]  end
    b[10]= xx;       //command 低字节 （comm=0Xxxxx，盒子恢复出厂设置）
    b[11]= xx;       //command 高字节 （危险命令，需要此命令的可以联系我们）
    UDPSend(b,12);  //把上面的 12 字节发出去，盒子就会恢复出厂设置，恢复出厂设置一般需要 3 分钟左右
```

//--

如果是中控，需要 16 进制发码，把上面数据翻译成 16 进制，你要发的是这串 16 进制，通过 UDP 把这串 16 进制发到端口(0x1939)6457 ：

```
                              41 56 4C 64 69 79 2E 63 6E 00 xx xx
固定值:                       41 56 4C 64 69 79 2E 63 6E 00 xx xx
解释：                           固定的命令头                 命令
解释：                        A  V  L  d  i  y  .  c  n  00 xx xx
自己写程序：                  b[0] b[1] ......              b[11]
```

注意： AVLdiy.cn 转换成 16 进制就是：41 56 4C 64 69 79 2E 63 6E

//--

第 2 章：C 代码实现方法：

提示：因为 WBOX 有 2 种控制模式，既可以控制 DMX512，又可以控制 SPI LED 灯带，3 通道的 RGB 灯用 DMX 控制只能控制 170 颗，170x3=510 通道，所以 SPI 控制模式下为了兼容 dmx 模式，buf 定义为 buf[510]，在 dmx 控制模式下 buf 定义为 buf[512]

```
/*
    //用 comm=0x0100 整包 512 通道 dmx 数据发到指定输出口 填充例子
    unsigned char b[540]={0};
    b[0] = 'A';
    b[1] = 'V';
    b[2] = 'L';
    b[3] = 'd';
    b[4] = 'i';
    b[5] = 'y';
    b[6] = '.';
    b[7] = 'c';
    b[8] = 'n';
    b[9] = 0;
    b[10] = 0x00;           //command 低位
    b[11] = 0x01;           //command 高位
    b[12] = 0xFE;           //Length(低位) SPI 模式填 0xFE  0x01FE=510（byte） / DMX 模式填 00 , 0x0200=512(byte) /或者全部填 0，用缺省值
    b[13] = 0x01;           //Length(高位) SPI 模式填 01                    / DMX 模式填 02,                  /或者全部填 0，用缺省值
    b[14] = Universe;       //Universe 低位，就是输出口，对应于 PORT0~PORT7（如果是 LED 模式，内部会拼接 0~3 为一根线输出，4~7 为一根线输出）
    b[15] = 0;              //Universe 高位，一般灯光项目没有这么多域，填 0
    b[16] = 0;              //NON
    b[17] = 0;              //NON
    b[18] = 0~255           //R0 第 1 点 RGB 值
    b[19] = 0~255;          //G0
    b[20] = 0~255;          //B0
    b[21] = 0~255           //R1 第 2 点 RGB 值
```

```
    b[22] = 0~255;          //G1
    b[23] = 0~255;          //B1
    ...
    b[525] = 0~255          //R169 第 170 点 RGB 值
    b[526] = 0~255;         //G169
    b[527] = 0~255;         //B169
*/
```

C 例子具体实现代码：

```
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <…>        //对应自己的开发平台 包含头文件
…
#define DMX_MODE 1  //1 为 DMX 模式， 0 为 SPI 模式
#if DMX_MODE
    #define BUF_SIZE 512
#else
    #define BUF_SIZE 510
#endif
```

/* 自己填写示例

```
void send_to_wbox(unsigned char universe, unsigned char *dmxdata_buf)
{
    unsigned char b[540]={0};
    b[0] = 'A';
    b[1] = 'V';
    b[2] = 'L';
```

```
    b[3] = 'd';
    b[4] = 'i';
    b[5] = 'y';
    b[6] = '.';
    b[7] = 'c';
    b[8] = 'n';
    b[9] = 0;
    b[10] = 0x00;           //command 低位
    b[11] = 0x01;           //command 高位
    b[12] = 0x00;           //Length(低位) RGB 数组的长度 SPI 模式填 01          /DMX 模式填 02  /或者全部填 0，用缺省值 510 或者 512
    b[13] = 0x00;           //Length(高位) SPI 模式填 0xFE  0x01FE=510（byte） /DMX 模式填 00, 0x0200=512(byte) /或者全部填 0，用缺省值
    b[14] = universe;       //Universe（域，就是输出口，SPI 模式下，盒子内部会拼接 0~3 域为一根线输出，4~7 域为一根线输出）
                            //DMX 模式下，对应 PORT0~PORT7 输出口
    b[15] = 0;              // Universe 高位，一般灯光项目没有这么多域，填 0
    memcpy(&b[18], dmxdata_buf, BUF_SIZE);  //b[18] 是 dmx 数据填充的起始位置

    //通过 udp 包发到我们盒子，这个函数需要自己写
    //发 udp 包的端口必须是 0x1939=6457，IP 可以单播，广播都可以（IP=255.255.255.255）
    udpsend(b, sizeof(b));
}
*/

//---- 用结构体填写通用函数 ----
void send_to_wbox(unsigned char universe, unsigned char *dmxdata_buf)
{
    unsigned char b[540]={0};                       //多出几个字节没有关系的
    struct my_dmx *dmx = (struct my_dmx *)&b[0];    //结构体
    dmx->id[0]= 'A';
    dmx->id[1]= 'V';
    dmx->id[2]= 'L';
```

```
    dmx->id[3]= 'd';
    dmx->id[4]= 'i';
    dmx->id[5]= 'y';
    dmx->id[6]= '.';
    dmx->id[7]= 'c';
    dmx->id[8]= 'n';
    dmx->id[9]= 0;
    dmx->Command = 0x0100;       //command
    dmx->Length = BUF_SIZE;      //数据长度， SPI 模式=510, DMX 模式=512 / 也可以填 0，盒子内部会根据模式，直接用缺省值 510 或者 512
    dmx->Universe = universe;    //域，SPI 模式下盒子内部会拼接 0~3 域为一根线输出，4~7 域为一根线输出，每根线控制 680 个 3 通道 RGB 像素
                                 //DMX 模式下，对应 PORT0~PORT7 输出口
    memcpy( &dmx->DmxData_Val_Start, dmxdata_buf, BUF_SIZE);

    //通过 udp 包发到我们盒子，这个函数需要自己写
    //发 udp 包的端口必须是 0x1939=6457，IP 可以单播，广播都可以（IP=255.255.255.255）
    udpsend(b, sizeof(b));  //按自己平台，自己写 udp 发包函数
}


int main()
{
    init_udp(); //按自己平台，自己写，初始化 udp 发包函数

    //定义一个保存域 0~7 全部通道 4096 的数组，每个域 510 字节（LED SPI 模式）或者 512 字节（DMX 模式），每个域控制 170 个 3 通道 RGB 像素灯
    // dmxdata[0] ~  dmxdata[511]   保存的是 域 0/PORT0 输出值
    // dmxdata[512] ~dmxdata[1023]  保存的是 域 1/PORT1 输出值
    // dmxdata[1024]~dmxdata[1535]  保存的是 域 2/PORT2 输出值
    // ……
    unsigned char dmxdata[ 8 * BUF_SIZE ] = {0};    // dmxdata[4096] 保存全部 8 个输出口 4096 通道的 dmx 值
```

```
memset( dmxdata, 255, 8 * BUF_SIZE );           //数组里面的值全部为255，全亮
for(int i=0; i<8; i++){
    send_to_wbox(i, &dmxdata[BUF_SIZE * i] );   //把0~7共8个域的值发送到盒子，盒子就会按照dmxdata的值变化
}

Sleep(5); //延时5s看一下效果

memset( dmxdata, 0, 8* BUF_SIZE );              //全灭，数组里面的值全部为0
for(int i=0; i<8; i++){
    send_to_wbox(i, &dmxdata[BUF_SIZE * i] );   //把8个域的值发送到盒子，盒子就会按照dmxdata的值变化
}

Sleep(5); //延时5s看一下效果

memset( &dmxdata[BUF_SIZE *2], 255, BUF_SIZE); //第2个域全亮
send_to_wbox( 2, &dmxdata[BUF_SIZE *2 ] );      //把第2个域的值发送到盒子，盒子就会按照dmxdata的值变化

Sleep(5); //延时5s看一下效果

memset( &dmxdata[BUF_SIZE *2], 0, BUF_SIZE );   //第2个域全灭
send_to_wbox( 2, &dmxdata[BUF_SIZE *2 ] );      //把第2个域的值发送到盒子，盒子就会按照dmxdata的值变化

Sleep(5); //延时5s看一下效果

//第5个域只亮红灯
for(int i=0;i<510;i=i*3){
    dmxdata[BUF_SIZE *5 + i ] = 255;    //0,3,6,9…… = 255, Red=255, other is 0
}
send_to_wbox( 5, &dmxdata[BUF_SIZE *5 ] );  //把第5个域的值发送到盒子，盒子就会按照dmxdata的值变化
```

```
    //第 4 个域只亮绿灯
    for(int i=0;i<510;i=i*3){
        dmxdata[BUF_SIZE *4 + i +1 ] = 255; //1,4,7,10…… = 255, Green=255, other is 0
    }
    send_to_wbox( 4, &dmxdata[BUF_SIZE *4 ] );  //把第 4 个域的值发送到盒子，盒子就会按照 dmxdata 的值变化

    //第 7 个域只亮蓝灯
    for(int i=0;i<510;i=i*3){
        dmxdata[BUF_SIZE *7 + i +2 ] = 255; //2,5,8,11…… = 255, Blue=255, other is 0
    }
    send_to_wbox( 7, &dmxdata[BUF_SIZE *7 ]);   //把第 7 个域的值发送到盒子，盒子就会按照 dmxdata 的值变化

//技巧：真正要做的就是改变 dmxdata[4096]里面的值，然后发到盒子，可以用一个定时器，每 30ms 发一次，然后你就不需要关注网络传输方面的东西，只需要按照灯光效果去改变 dmxdata[4096]里面的值，这样就能做出不同灯光效果。要善于用 sin, cos 函数去改变 dmx 值。

}//end main
```

```
//----全部清零，关灯函数，简单方法----
void sendzero()
{
    unsigned char b[24] = { 0 };

    b[0] = 'A';
    b[1] = 'V';
    b[2] = 'L';
    b[3] = 'd';
    b[4] = 'i';
    b[5] = 'y';
    b[6] = '.';
    b[7] = 'c';
    b[8] = 'n';
    b[9] = 0;
    b[10] = 0x00;
    b[11] = 0xCB;
    b[12] = 0;
    b[13] = 0;
    b[14] = 0;
    b[15] = 0;
    b[16] = 0;
    b[17] = 0;
    b[18] = 0;
    b[19] = 0;
    b[20] = 0;
    udpsend(b, 24);
}
//-------------------------------------------------------------
```

第 3 章：Web 网页二次开发方式：

请浏览我们网站 http://box.AVLdiy.cn 下载源代码观看，或者浏览盒子 http://192.168.7.1/sdkdemo.html 观看源代码
这种开发方式十分强大，只要会做网页就会开发，我们很多客户通过这种方式开发出很多强大功能的应用。WBOX 本身是一台服务器，可以运行 JavaScript，网上有很多开源 JS 可以借鉴，围绕灯光控制，可以开发出很多图形、图像、3D、音频、喷泉、项目管理，时间管理，计划任务等一体功能，只需要 1 台我们盒子就全部完成。

```
wboxsdk.js is a wbox web sdk file , in your web html page only include this file
//-------------------------------------------------------------
//function send_dmxdata(dmxdata_buf);
// dmxdata_buf[4096] is a dmxdata buffer, save all 4096 channels dmx value
// dmxdata_buf[0]~dmxdata_buf[511] is port0,
// dmxdata_buf[512]~dmxdata_buf[1023] is port1
// dmxdata_buf[1024]~dmxdata_buf[1535] is port2,
// .....
//-------------------------------------------------------------

//--this function not in wboxsdk.js, it's in sdkdemo.js--------
//function send_channels(universe, channels, dmx_value);
// universe only 0~7 port output
// channels only 0~511 channels
// dmx_value only 0~255
// For example:
// send_channels(0, 100, 255);  //set port0, 100 channel to 255
// send_channels(7, 1, 64);     //set port7, 1 channel to 64
//-------------------------------------------------------------
```

```
//-------------------------------------------------------------
//function start_catch(dmxrate, dev, filename);
//to starting catch dmx file, dmx file will save to usb disk or sd card "/light" DIR
//dmxrate: 25ms~65535ms, default is 30ms , 33Hz
//dev: "usb" or "sd"
//filename: save as filename, filename don't above 80 char, don't use Chinese filename
// For example:
// start_catch(30, "usb", "mydemo.dmx") ;   //each 33Hz(1000/30)) will catch a dmx file to usb/light/mydemo.dmx
// start_catch(50, "sd", "demo2.dmx") ;     //each 20Hz(1000/50)) will catch a dmx file to sd/light/demo2.dmx
//-------------------------------------------------------------


//-------------------------------------------------------------
//function stop_catch();
//call this function will stop catch dmx file function
//-------------------------------------------------------------
```

第 4 章：中控发文本命令控制方式：

我们为 WBOX 编有专门用于集中控制发命令文本的程序 **ccplay**，如果你想用 Crestron、AMX、国内其它品牌的集中控制系统通过发 udp 文本命令控制灯光的话，可以升级这个固件 wbox_Crestron_AMX_v2.3.1.102_20210603.bin，这个固件会自动运行 ccplay 集中控制命令接收程序，通过集中控制系统发 udp 包就可以控制灯光，发包命令为 txt 文本，注意是 ASCII 文本，不是数字，也不是 HEX，**udp 包的端口为 6799**。如果固件是其他版本，wbox 开机时，ccplay 是不会自动运行的，因此其他版本的固件如果要开机自动运行 ccplay 的话，只需要编辑“/etc/rc.local”这个启动文件，在“exit 0”前加入我们需要启动的程序 ccplay，盒子启动时就会自动运行这个启动文件里面的所有东西。

命令格式如下：

全部输出口，输出相同推子的值，命令格式为： **PUSH 灯的地址码 这个地址码的灯的第 1 个通道值， 第 2 个通道值， 第 3 个通道值...** ；

全部输出口，全部推子输出相同值，类似我们网页版的 AllTo255，AllTo0 命令，格式为： **SETV 255 / SETV 0 / SETV 128 ...** ；

指定盒子输出口，输出推子的值，命令格式为： **PORT[0～7] 灯的地址码 这个地址码灯的第 1 个通道值， 第 2 个通道值， 第 3 个通道值...** ， 注意 0 对应第一个 DMX0 输出口，7 对应 DMX7 输出口 ；

指定盒子输出口，输出全部 512 个推子的值（全部通道），命令格式为：**SETP[0～7] 0 / SETP2 255 / SETP7 128**， 注意 0 对应第一个 DMX0 输出口，7 对应 DMX7 输出口 ；

比如有这么一个灯，其通道属性如下：

第 1 通道为： X

第 2 通道为： Y

第 3 通道为： XY 精度

第 4 通道为： 总亮度

第 5 通道为： R

第 6 通道为： G

第 7 通道为： B

分别接了 18 台灯，灯的地址码分别是 1，17，33，49，65，81，97，113，129，145，161，177，193，209，225，241，257，273...

控制第 1 台灯 X 转满，Y 转半圈，红灯亮，通过 UDP 发包到 6789 端口，发这串 txt 文字就可以： **PUSH 1 255 128 0 255 255**

控制第 2 台灯 X 转一半，Y 转整圈，蓝灯亮，通过 UDP 发包到 6789 端口，发这串 txt 文字就可以： **PUSH 17 128 255 0 255 0 0 255**

控制第 4 台灯只亮绿灯，通过 UDP 发包到 6789 端口，发这串 txt 文字就可以：

	PUSH	**49**	**0**	**0**	**0**	**255**	**0**	**255**	**0**
解释：	推推子命令	第 4 台灯地址码	x	y	xy	总亮度	R	G	B

假设其中一台灯接在输出端口 DMX7 上，地址码是 257， 需要 XY 旋转，发 udp 包： **PORT7 257 255 255**
假设其中一台灯接在输出端口 DMX3 上，地址码是 177， 需要亮 RGB，发 udp 包： **PORT3 177 0 0 0 255 255 255 255**

关闭 8 个输出口的全部灯，所有 4096 通道变为 0： **SETV 0**
只关闭 dmx4 输出口的全部灯，dmx4 所有 512 通道清 0： **SETP4 0**
dmx2 输出口的 512 通道全部最亮： **SETP2 255**

第 5 章：Linux Shell 开发方式：

这种开发方式强大到没有朋友，推荐使用，特别是和 websdk 结合，JS 里面做交互式页面，可以做出很多高大上的应用。我们很多客户在我们的指导下，开发出很多成功案例。

我们为盒子开发了很多程序，全部固化在盒子里面，可以通过 shell 或者脚本组合起来，比如 light, port, port_new, effects, lightplay, catch_devfile, play_devfile ……等等，把这些程序像搭积木一样通过脚本组合起来，就可以完成很多复杂任务。

具体方法，可以通过终端登入到盒子进行开发，也可以通过我们 webui 登入到盒子进行开发，盒子带 vi 编辑器，可以直接编辑 ，或者本地通过记事本、VS 等编辑软件按自己意思编辑好，通过 winscp、ssh 等终端软件上传到盒子上，同时赋予运行权限就可以。

此种开发方式太过强大，没办法具体写开发例子，如果你有具体项目，有什么不清楚的地方可以联系我交流。

如果你需要加什么功能，也可以联系我们，只要你能提出，基本上我们都可以做到！

注意：早期版本的固件，这些命令和脚本也可以通过 udp 发包进行调用或者运行，方法是运行 ~~netcomm~~ 程序（udp 包的端口为 5688），然后就可以直接通过 udp 发命令运行 shell 或者盒子里面的程序，具体方法可以观看我们视频教学 update 那讲（http://down.avldiy.cn/down_server/W-BOX/video_teaching/ ）。新版本 V3.5.1.102 后的固件请忽略此点，看下章，有更先进的方法和程序可以使用。

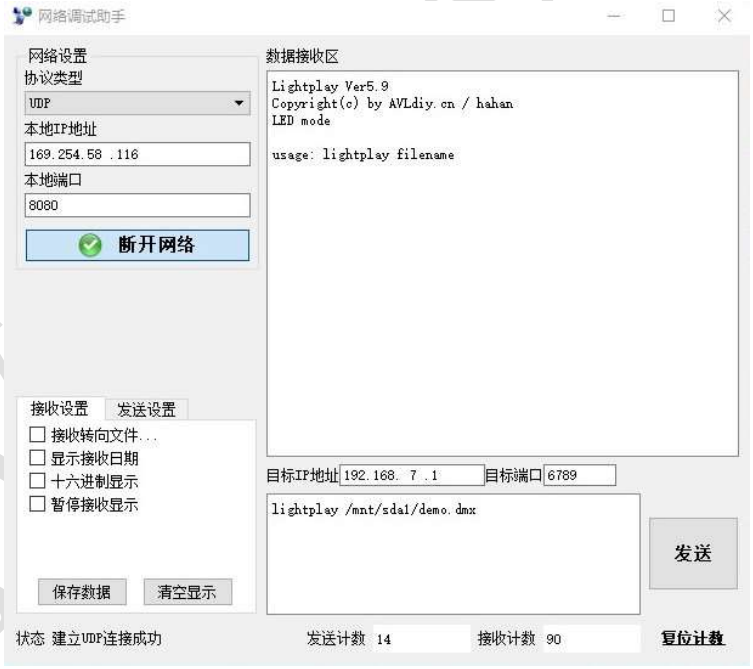
第 6 章：UDP Shell 开发方式（外部调用重点推荐方式）：

很多做系统集成和中控项目的朋友偏向于通过发简单的命令，达到调用播放盒子里面的场景。盒子可以通过捕捉方式把不同灯光效果捕捉成 dmx 文件，通过 lightplay 可以声光同步播放音频和 dmx 文件，通过 play_devfile 可以单独播放 dmx 文件，通过 effects 可以产生很多 RGB 效果，通过 light 程序可以设置不同灯光 dmx 值，因此就需要一个可以让外部调用的通用方法，为此我们单独开发了这样的程序，可以通过 udp 直接运行 linux shell，直接把 linux 的 shell 通过 udp 搬到你面前，现在，你可以通过发 udp 运行命令，运行程序，运行脚本，编写脚本，查看、监控设备等等，新版本固件的盒子内部集成了这个功能，开箱就可以直接使用这个功能。

在使用这个功能前，我们建议您具备一定的命令行操作基础，比如有过 windows 的 cmd 输入命令的经历，有过 Linux shell 输入命令的经历，如果没有类似经历，你可以想象成输入的命令，实际上就类似你在 windows 里面用鼠标双击桌面的程序一样，然后 windows 会打开这个程序，运行起这个程序。

使用前注意事项：注意，**udp 发包的端口是 6789 和前面 RAW 协议的端口是不一样的，请注意一下**。以下操作，我们通过网络调试助手模拟中控操作，当然上位设备也可以是其它任何可以发网络 udp 的设备。

例 1：假设 U 盘上有一个 demo.dmx 的脱机文件，通过 udp 如何调用呢？



如上图，设置发送到盒子的 **IP：192.168.7.1， 端口：6789**，发送命令：**lightplay /mnt/sda1/demo.dmx**

说明：上面的意思就是告诉盒子要运行这条语句。其中 /mnt/sda1 表示 U 盘这个设备，注意，**linux 里面一切皆文件**，硬件设备也是文件，这点和 Windows 大

不相同，**lightplay** 是我们编写的专用播放器，具体用法可以看我们早期视频介绍（http://down.avldiy.cn/down_server/W-BOX/video_teaching/ ）。

盒子运行这条语句后，会通过 udp 发回运行的信息，发回的信息相当于你的显示器，在网络调试助手里面，下面输入命令，上面显示命令运行结果。调试时，可以关注这些显示的信息，有非常大的帮助，使用时可以忽略这些信息。比如我们要让 PORT2，3，7 输出 20 秒内置效果的第 14 号效果，要怎么做呢？内置效果程序 effects 参数太多，忘记了，可以通过加-h 参数先帮助一下。effects 具体用法可以看我们早期视频介绍（http://down.avldiy.cn/down_server/W-BOX/video_teaching/ ）。

发命令：**effects -h** ，会返回使用方法的帮助信息。注：新版本的 effects1.2 增加了**-u** 参数，可以指派某个输出口输出效果，dmx 值不会影响其他输出口，这个参数一定要用单引号或者双引号括起来，每个需要输出的输出口用空格隔开，**-t** 参数是设置效果运行时间，单位是秒，**-f** 是不同效果的编号。**需要特别注意的是：参数前面的“-”以及“”引号必须是英文的，不能是中文的**，请特别注意，即使拷贝下面的命令去测试，也需要注意是不是中文字符。

例 2：PORT2，3，7 输出 20 秒内置效果的第 14 号效果。

udp 发这条命令： **effects -t20 -f14 -u"2 3 7"**

注意：所有的命令都可以叠加，就是可以再次运行，比如上面的 2，3，7 在输出效果的同时，我们还可以继续叠加，让 1，4 输出口运行其他效果，比如让 1，4 输出效果 9，时间 30 秒，颜色随机，只需要继续发：**effects -t30 -f9 -u"1 4"-C** 。

例 3：还是例 2 的例子，不想一条一条发，想一次性发多条命令，要怎么才能做到呢？
只需要命令和命令之间用**分号隔开**就可以，注意是英文的分号，不能用中文的分号，所有命令的输入必须是英文状态，不能有中文字符，如果 udp 发的文本可以包括换行符，每一行一条命令，一行一行可以分割，那么分号可以不用，不能的话要加分号，于是上面的命令就变成如下样子了：

effects -t20 -f14 -u"2 3 7" ; effects -t30 -f9 -u"1 4"-C

例 4：例 3 的这个例子，效果播放时间到后，会保留最后一步的输出状态，最后的亮度还会继续保留，我们想关闭要怎么办呢？
可以用 light 命令把通道的 dmx 值设置为 0 进行关闭。light 具体用法可以看我们早期视频(http://down.avldiy.cn/down_server/W-BOX/video_teaching/)。
udp 命令如下：

effects -t20 -f14 -u"2 3 7" & effects -t30 -f9 -u"1 4"-C ; light 8 s 1 0

这一串命令是不是似曾相识？哈哈，这不就是脚本吗！是的，现在你可以通过 udp 编写脚本了，而且可以省去脚本包含的头。我们把这串代码重新编排一下，就是你平时编写的脚本的样子。

```
effects -t20 -f14  -u"2 3 7" &

effects -t30  -f9 -u"1 4"-C ;

light 8 s 1 0
```

上面用**&**符号替换了**;**号（分号），为什么呢？

这就涉及到一个 linux 的概念，堵塞和非堵塞的问题，所谓堵塞就是程序运行后不会立刻退出，需要彻底的运行完才退出，比如上面 **effects -t20** 运行 20 秒，那么就需要 20 秒后才能退出，在没有退出前，会堵塞后面程序运行，只有它退出后，才接着一条一条运行后面的语句，为了解决这个问题，我们可以在堵塞的程序后面加一个**&**符号，表示后台运行，这样就不会堵塞其他程序顺序运行。加了**&**符号后，就不能再加**分号**。

但是上面的语句还是有问题，因为 20 秒后，PORT2，3，7 的效果就停了，我们想立刻把 2，3，7 输出口的亮度关闭，而不是保留成最后的那次效果的亮度值，但上面的语句做不到，还会继续保留 10 秒，直到 **effects -t30 -f9 -u"1 4"-C** 退出，30 秒后，才运行 **light 8 s 1 0**，把全部输出关闭，这个不是我们想要的结果，于是我们再改进一下（为了好看，下面按脚本方式进行排版）：

```
effects -t20 -f14 -u"2 3 7" &

effects -t30 -f9 -u"1 4"-C &

sleep 21 ;
light 2 s 1 0;
light 3 s 1 0;
light 7 s 1 0;
sleep 12;
light 8 s 1 0
```

解释：我们让 effects 都以后台方式运行，不堵塞其他语句，然后开始计时，20 秒后，关闭 2，3，7 输出，接着再计时，10 秒后，关闭全部输出，总时长是 30 秒。这里延时多加了 1~2 秒，因为 shell 的延时并不精确，同时系统调度会花一点时间，为了保证 effects 退出时间的误差，达到能够正确关闭输出，多加 1~2 秒进行确保关闭。如果时间敏感的同步项目，可以用其他方法定时，可以达到微秒级，比如我们编写的延时程序 **mysleep**，精度可以达到微秒级。

例 5：如何关闭运行程序，假设 effects 一直在运行，我们想提前关闭它，要怎么做呢？ udp 发下面命令就可以。

```
kill `ps|grep 'effects'|grep -v grep|awk '{print $1}'`
```

解释：任何程序或者脚本运行后，都有一个 PID，找出这个 PID 再 kill 掉就可以关闭程序。注意，kill 后面有 2 个反斜点括起来了，这个点是键盘数字 1 前面的那个点，请注意一下。 如果想关闭 lightplay 呢，又该怎么做？ 只需要把程序名称改一下，如下：

```
kill `ps|grep 'lightplay'|grep -v grep|awk '{print $1}'`
```

例 6：如何检测盒子是否开机或者健康？
udp 只需要发 hello，盒子会返回字符 OK，检测到 OK 就表示盒子已经开机并且健康。
udp 发文本：**hello**　盒子返回字符：**OK**

例 7：如何运行盒子里面保存的脚本或者自己编写的脚本？
只需要把脚本路径和名称输入，通过 udp 发出去，就会运行，操作和 shell 是一模一样的，比如盒子/mydemo 目录保存了一个测试程序 test.sh，只需要通过 udp 发这个语句就可以：　**/mydemo/test.sh**

例 8：如何运行 linux 里面的命令？　比如 ls，mkdir，ps……等等
udp 直接发命令就可以，不过需要注意的是：堵塞命令和程序只有退出时才能返回信息，比如 top 命令是一直堵塞的，只有退出 top 时才能看到返回信息，但 top 运行后，除非人为关闭，是不会退出的，所以查询运行的进程建议用 ps 命令，上面**例子 5** 就是 ps 命令把全部进程查找出来，再格式化显示 PID，最后再 kill 掉。

……

还有很多很多例子……，只要你会 linux 的基本操作，可以玩出很多很多花样，这种调用方式不存在协议问题，一切东西都是你说了算，可以发挥你的最大自由度和灵活度，非常适合第三方使用，是我们极力推荐的一种二次开发方式。
这种调用方式，据我们所知，目前市面上应该没有，是我们独创的一种利器，欢迎大家利用我们的利器在 wbox 灯光平台上开发出更多好用的东西，为您的工程项目大放异彩。

第 7 章：捕捉 dmx 文件（补充文档）

外部调用经常需要使用捕捉 dmx 功能。wbox 里面有 2 套捕捉系统，其中 webui 里面的捕捉功能是把 8 个输出口同时捕捉进行保存，捕捉的是外部控制台或者软件通过 artnet、sCAN 发过来的数据或者我们支持的专业软件发过来的数据，捕捉的这些数据是会加密的，而且必须是一个整体，因此不能单独只捕捉某个输出口，只能一次性捕捉 8 个输出口。这些数据不能拆分，拆分后将会乱码。

另外一套捕捉系统是通过 catch_devfile 软件进行捕捉的，对应的播放程序是 play_devfile，这套软件可以单独捕捉某个或者某几个输出口，这套捕捉程序主要捕捉的是自己二次开发的数据，比如通过 RAW 协议 com=0x0100 命令发过来的数据，或者我们效果生成器 effects 生成的数据，或者通过自己编写的脚本程序产生的效果。这些数据可以拆分，每个输出口都可以独立，数据只简单变形。

联系方式：
http://box.AVLdiy.cn
Email: 1195722899@qq.com
谢文才(hahan) 13808858586

2022-12-13