Description of AVLdiy.cn-WBOX Control Protocol (Release1.2)

The firmware version is greater than V3.5.1.102 to support this protocol _by hahan.

Chapter 1: Description of RAW data protocol

```
//-------------------------------------------------------------------------
```

The network protocol is UDP:

The light is controlled by sending UDP packets, UDP port must be: **(0x1939)6457**. WBOX like a server.

```
//-------------------------------------------------------------------------

//----------------------------control data protocol format ---------------------------
//---------------- Data length: BYTE: [1byte]    WORD: [2byte] ----------------
///---------------------- Byte order little-endian ----------------------------------
struct my_dmx {
      BYTE id[10];                //0 ~ 9 box protocol header AVLdiy.cn0 (fixed value, unchanged)
      WORD Command;               //10 ~ 11 control commands. See the DMX protocol command description for different commands.
      WORD Length;                //12 ~ 13 Length of DMX data sent
      WORD Universe;              //14 ~ 15 Which output port is Port 0 ~ Port 7?    please refer to the mean of the teaching video universe for the setting method.
      WORD Channel;               //16 ~ 17 CC/CD command needs to specify which channel (0 ~ 511), index from 0, note that it is not (1 ~ 512)
      BYTE DmxData_val_Start;   //18    1: [xxx value (0 ~ 255) needs for CB/CC/CD command],    2:[ Start bit value filled by dmx when sending whole packet data]
};
//-------------------------------------------------------------------

//-----------------------------------------------------------------------------------------------
```

DMX control protocol command:

```
Command = 0x0100:     //The whole package of 512-channel dmx data is sent to the output port (it can be assign to the output port port0 ~ port7)
Command = 0x0200:     //The whole package of 512-channel dmx data is sent to all output ports (the data of PORT0-PORT7 is the same)
Command = 0xCA00:     //Change all 512 channels of the specified output port to xxx value (xxx is the value of 0 ~ 255)
Command = 0xCB00:     //Change all 512 channels of each output PORT0 ~ PORT7 to xxx value (xxx is the value of 0 ~ 255)
Command = 0xCC00:     //Change a channel of the output port to xxx value (xxx is the value of 0 ~ 255)
```

Command = 0xCD00: //Change a channel of all output PORT0 ~ PORT7 to xxx value (xxx is the value of 0 ~ 255)
Command = 0xCE00: //Assign the dmx value of 16 channels continuously to an address of the output port.
Command = 0xC100: //LED command, turn on all red lights at the output port, and the brightness is xxx (xxx is the value from 0 to 255)
Command = 0xC200: //LED command, turn on all green lights of the output port, and the brightness is xxx (xxx is the value from 0 to 255)
Command = 0xC300: //LED command, turn on all blue lights at the output port, and the brightness is xxx (xxx is the value from 0 to 255)
Command = 0xB100: //LED command, turns on all red lights of the output port, unlike the 0xC100 command, it will clear the other dmx value.
Command = 0xB200: //LED command, turns on all green lights of the output port. unlike 0xC200 command, it will clear the other dmx value.
Command = 0xB300: //LED command, turn on all blue lights at the output port, unlike 0xC300 command, it will clear the other dmx value.
Command = 0xB400: //LED command to change all channels of the output port into RGB values.
Command = 0xFFFF: //Check whether the box is turned on and online.
Command = 0xXXXX: //Restore the box to factory settings (Dangerous command, you can contact us if you need this command).
//----------

//##########

Example 1:

//This example demonstrates one output out 512 channels data. Each channel data can be changed, 512+20 bytes buffer needs to be defined.

unsigned char b[600]; //define a buff. If there is not so much space, you only need to define 512+20bytes.
b[0]='A'; //our 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 low-order byte (comm=0x0100, the whole packet of 512-channel dmx data is sent to the output port)
b[11]= 0x01; //Command high-order byte (note the high and low bits of the network byte order)
b[12]= 0x00; //Length low byte, box defaults to 512 channels (0x0200=512), 0x0000 defaults to 512 channels.
b[13]= 0x02; //Length high byte, 0x0200 = 512
b[14]= 0 ~ 7; //Universe low byte of the output port
b[15]= 0; //Universe high byte. If the output port is greater than 255, it needs to be converted into 2 bytes. Fill here, and the default value is 0.
b[16]= 0; //Channel low byte, which is invalid under this command.
b[17]= 0; //Channel high byte, which is invalid under this command.
b[18]= xxx; //0 ~ 255, the height of the first fader DMXData[0]
b[19]= xxx; //0 ~ 255, the height of the second fader DMXData[1]
b[20]= xxx; //0 ~ 255, the height of the third fader DMXData[2]
.
.
b[18+512]= xxx; //0 ~ 255, the height of the 512th fader DMXData[511]
UDPSend(b,532); // function of sending udp packets. Send the 532 bytes to wbox. Note that sending udp must correspond to port (0x1939)6457.

//If it is any programming language, it will use the memcpy () function, don't filling the 512 channel one by one, just use the following copy function.
//memcpy(&(header->DmxData_val_Start),DMXData,512);
//Tip: only to be changed is the value of each channel in DMXData[512] array. Other ID values can be fixed. You can use a timer to send this packet every 30ms.

//--
Crestron, AMX / Central control hexadecimal code (send this string of hexadecimal to port (0x1939)6457 through UDP):

	41 56 4C 64 69 79 2E 63 6E 00 00 01 00 00 xx xx 00 00 xx xx ...	xx
Ony to be changed this xxxx value:	41 56 4C 64 69 79 2E 63 6E 00 00 01 00 00 xx xx 00 00 xx xx ...	xx
Mean:	Fixed command header cmd len Port fader1 fader2 ...	fader512
Mean:	A V L d i y . c n 00 00 01 00 00 xx xx 00 00 xx xx ...	xx
Program buffer:	b[0] b[1] ... b[18] b[19] ...	b[18+512]

Note: AVLdiy.cn is converted into hexadecimal: 41 56 4C 64 69 79 2E 63 6E.
//--

//###

Example 2:

//This example demonstrates that all PORT0 ~ PORT7 out the same DMXData[512] value, each channel can be changed. 512+20 bytes buffer needs to be defined.

unsigned char b[600]; //define a buff. If there is not so much space, you only need to define 512+20bytes.

b[0]='A'; //our 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 low byte (comm=0x0200 the whole packet of 512-channel dmx data is sent to all output ports, and all output ports have the same value)
b[11]= 0x02; //Command high byte (note the high and low bits of the network byte order)
b[12]= 0x00; //Length low byte, box defaults to 512 channels (0x0200=512), 0x0000 defaults to 512 channels.
b[13]= 0x02; //Length high byte, 0x0200 = 512
b[14]= 0 ~ 7; //Universe low byte of the output port
b[15]= 0; //Universe high byte. If the output port is greater than 255, it needs to be converted into 2 bytes. Fill here, and the default value is 0.
b[16]= 0; //Channel low byte, which is invalid under this command.
b[17]= 0; //Channel high byte, which is invalid under this command.
b[18]= xxx; //0 ~ 255, the height of the first fader DMXData[0]
b[19]= xxx; //0 ~ 255, the height of the second fader DMXData[1]
b[20]= xxx; //0 ~ 255, the height of the third fader DMXData[2]
.
.
b[18+512]= xxx; //0 ~ 255, the height of the 512th fader DMXData[511]

UDPSend(b,532); // function of sending udp packets. Send the 532 bytes to wbox. Note that sending udp must correspond to port (0x1939)6457.

//If it is any programming language, it will use the memcpy () function, don't filling the 512 channel one by one, just use the following copy function.
//memcpy(&(header->DmxData_val_Start),DMXData,512);
//Tip: only to be changed is the value of each channel in DMXData[512] array. Other ID values can be fixed. You can use a timer to send this packet every 30ms.

//--
If it is central control device like Crestron and AMX, send this code of hexadecimal through UDP, UDP port is (0x1939)6457:

	41 56 4C 64 69 79 2E 63 6E 00 00 02 00 00 xx xx 00 00 xx xx ... xx
Ony to be changed this xxxx value:	41 56 4C 64 69 79 2E 63 6E 00 00 02 00 00 xx xx 00 00 xx xx ... xx
Mean:	Fixed command header cmd len Port fader1 fader2 ... fader512
Mean:	A V L d i y . c n 00 00 02 00 00 xx xx 00 00 xx xx ... xx
Program buffer:	b[0] b[1] ... b[18] b[19] ... b[18+512]

Note: AVLdiy.cn is converted into hexadecimal: 41 56 4C 64 69 79 2E 63 6E.
//--

//##

Example 3:

//This example demonstrates that all 512 channels of an output port to xxx values, which is equivalent to pushing all 512 faders to the same value (0 ~ 255)

unsigned char b[600]; //define a buff. If there is not so much space, you only need to define 20byte.

b[0]='A'; //fixed head 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 low byte (comm=0xCA00 changes all 512 channels of the output port into xxx values)
b[11]= 0xCA; //command high byte
b[12]= 0; //Length low byte, which is invalid under this command. The channels number is 512 by default, which can be 0 by default.
b[13]= 0; //Length high byte, which is invalid under this command. The channels number is 512 by default, which can be 0 by default.
b[14]= 0 ~ 7; //low byte of the port universe value, the default value of the box is 0 ~ 7.
b[15]= 0; //Universe high byte. If the output port is greater than 255, it needs to be carried and converted into 2 bytes. fill here, the default value is 0.
b[16]= 0; //Channel low byte, which is invalid under this command. The default value is 0.
b[17]= 0; //Channel high byte, which is invalid under this command. The default value is 0.
b[18]= 0 ~ 255; //dmx value of the fader. (Tip: You can change this value to 0 and send it out at intervals of 255, and you can check whether the lighting is controlled or not.)

UDPSend(b,20); //This is the function of sending udp packets. Send the above 20 bytes to wbox. Note that sending udp must correspond to port (0x1939)6457.

//--
If it is central control device like Crestron and AMX, send this code of hexadecimal through UDP, UDP port is (0x1939)6457:

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

Ony to be changed this xxxx value: 41 56 4C 64 69 79 2E 63 6E 00 00 CA 00 00 xx 00 00 00 xx

Mean: Fixed command header cmd len Port dmx(0~255)

Mean: A V L d i y . c n 00 CA 00 00 00 xx 00 00 00 xx

Program buffer: b[0] b[1] ... b[18]

Note: AVLdiy.cn is converted into hexadecimal: 41 56 4C 64 69 79 2E 63 6E.

//--

//###

Example 4:

//This example demonstrates that all 512 channels of all output ports to xxx values (0 ~ 255)

unsigned char b[600]; //define a buff. If there is not so much space, you only need to define 20byte.

```
b[0]='A';        //fixed head 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 low byte (comm=0xCB00 All 512 channels of all output ports become xxx values)
b[11]= 0xCB; //command high byte
b[12]= 0;        //Length low byte, which is invalid under this command. The channels number is 512 by default, which can be 0 by default.
b[13]= 0;        //Length high byte, which is invalid under this command. The channels number is 512 by default, which can be 0 by default.
b[14]= 0;        //Universe low byte, which is invalid under this command. by default, all 8 ports output the same value at the same time, which can be 0.
b[15]= 0;        //Universe high byte. which is invalid under this command. by default, all 8 ports output the same value at the same time, which can be 0.
b[16]= 0;        //Channel low byte, which is invalid under this command. The default value is 0.
b[17]= 0;        //Channel high byte, which is invalid under this command. The default value is 0.
b[18]= 0 ~ 255; //dmx value of the fader. (Tip: You can change this value to 0 and send it out at intervals of 255, and you can check whether the lighting is controlled or not.)
                 //Tip: set to 0, all device in output port will be close.
```

UDPSend(b,20); //This is the function of sending udp packets. Send the above 20 bytes to wbox. Note that sending udp must correspond to port (0x1939)6457.

//----------

If it is central control device, send this code of hexadecimal through UDP, UDP port is (0x1939)6457:

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

Ony to be changed this xx value: 41 56 4C 64 69 79 2E 63 6E 00 00 CB 00 00 00 00 00 00 xx

Mean: Fixed command header cmd len Port dmx(0~255)

Mean: A V L d i y . c n 00 00 CB 00 00 00 00 00 00 xx

Program buffer: b[0] b[1] ... b[18]

Note: AVLdiy.cn is converted into hexadecimal: 41 56 4C 64 69 79 2E 63 6E.

//----------

//##

Example 5:

//This example demonstrates the value of xxx for a channel of an output port, which is equivalent to pushing a fader of this output port to 0 ~ 255.

//The following demonstrates the 100th fader of output port 1 and pushes it to position 128.

unsigned char b[600]; //define a buff. If there is not so much space, you only need to define 20byte.

```
b[0]='A';        //fixed head 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 low byte (comm=0xCC00) changes a channel of the specified output port to xxx value (0 ~ 255)
b[11]= 0xCC;        //command high byte
b[12]= 0;           //Length is low, which is invalid under this command. You can default to 0.
b[13]= 0;           //Length is high, which is invalid under this command. You can default to 0.
b[14]= 1;           //The low byte of the output port specifies the universe value of the port (the default value of the box is 0 ~ 7 output ports)
b[15]= 0;           //The high byte of the output port, if the output port is greater than 255, is converted into 2 bytes. The high byte is filled here, and the default value is 0.
b[16]= 99(0x63);    //Channel low byte, fader 100 (counting from 0, 99 is the 100th fader)
b[17]= 0;           //Channel high byte, if it is greater than 255 or more, you need to carry it and convert it into 2 bytes. Fill in the high bit here.
b[18]= 128(0x80);   //dmx value is 0 ~ 255, which is the height of the fader. The hexadecimal of 128 is 0x80.
UDPSend(b,20);      // This is the function of sending udp packets. Send the above 20 bytes to wbox. Note that sending udp must correspond to port (0x1939)6457.

//-----------------------------------------------------------------
```

If it is central control system, translate the above data into hexadecimal and send the code in hexadecimal through UDP, UDP port is (0x1939)6457:

	41 56 4C 64 69 79 2E 63 6E	00 00	CC 00	00 01	00	63 00	80
Ony changed this xx value:	41 56 4C 64 69 79 2E 63 6E	00 00	CC 00	00 xx	00	xx xx	xx.
Mean:	fixed command header		cmd	len	port	channel(0~511)	dmx(0~255)
Mean:	A V L d i y . c n	00 00	CC 00	00 01	00	xx xx	xx
Program buffer:	b[0] b[1] ...						b[18]

Note: AVLdiy.cn is converted into hexadecimal: 41 56 4C 64 69 79 2E 63 6E.

```
//-----------------------------------------------------------------


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

Example 6:

```
//This example demonstrates that a channel of all output ports is xxx, which is equivalent to pushing a fader of all output PORT0 ~ PORT7 to (0 ~ 255)
//The following demonstrates the 200th fader of all output ports, pushing it to the 255 position.

unsigned char b[600]; //define a buff. If there is not so much space, you only need to define 20byte.
```

```
b[0]='A';         //fixed head 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 high-order byte (comm=0xCD00 changes a channel of all output ports(port0 ~ port7) to xxx value)
b[11]= 0xCD;          //command low byte
b[12]= 0;             //Length low byte, which is invalid under this command. You can default to 0.
b[13]= 0;             //Length high byte, which is invalid under this command. You can default to 0.
b[14]= 0;             //Universe low byte, which is invalid under this command. By default, all 8 ports output the same value at the same time, which can be 0.
b[15]= 0;             //Universe high byte, which is invalid under this command. By default, all 8 ports output the same value at the same time, which can be 0.
b[16]= 199(0xC7); //Channel low byte, fader 200 (counting from 0, 199 is the 200th fader)
b[17]= 0;             //Channel high byte, if it is greater than 255 or more, you need to fill it here.
b[18]= 255(0xFF);  //DMX value is 0 ~ 255, which is the height of fader.
UDPSend(b,20);     //This is the function of sending udp packets. Send the above 20 bytes to box. Note that udp port must to be (0x1939)6457.

//-------------------------------------------------------------------------------
If it is central control device, send this code of hexadecimal through UDP, UDP port is (0x1939)6457:
                                        41 56 4C 64 69 79 2E 63 6E 00 00 CD 00 00 00 00   C7 00     FF
Ony to be changed this xx value:        41 56 4C 64 69 79 2E 63 6E 00 00 CD 00 00 00 00   xx xx     xx
Mean:                                     Fixed command header          cmd                  channel   dmx(0~255)
Mean:                                    A V L  d  i  y  .  c n   00 00 CD 00 00 00 00   C7 00     FF
Program buffer:                         b[0] b[1] ...                                                b[18]
//-------------------------------------------------------------------------------
```

//###

Example 7:

//This example demonstrates that from an address, 16 consecutive channels of an output port are assigned to xxx values (0 ~ 255)

//The following demonstration output port 2 starts from the 65th fader, and 16 faders push to the 255 position continuously.

```
unsigned char b[600]; //define a buff. If there is not so much space, you only need to define 36byte.
b[0]='A';        //fixed head 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 low byte (comm=0xCE00 assigns dmx value of 16 channels continuously to an address of the specified output port)
b[11]= 0xCE;         //command high byte
b[12]= 0;            //Length low byte, which is invalid under this command. You can default to 0.
b[13]= 0;            //Length high byte, which is invalid under this command. You can default to 0.
b[14]= 2;            //Universe low byte, and the example is 2.
b[15]= 0;            // Universe high byte.
b[16]= 64(0x40);     //Channel low byte, fader 65 (counting from 0, 65 is the 64th fader)
b[17]= 0;            //Channel high byte, if it is greater than 255 or more, you need to carry it here.
b[18]= 255(0xFF);  //dmx value is 0 ~ 255, which is the height of fader.
b[19]= 255(0xFF);  //dmx value is 0 ~ 255, which is the height of fader.
b[20]= 255(0xFF);  //dmx value is 0 ~ 255, which is the height of fader.
b[21]= 255(0xFF);  //dmx value is 0 ~ 255, which is the height of fader.
b[22]= 255(0xFF);  //dmx value is 0 ~ 255, which is the height of fader.
```

b[23]= 255(0xFF); //dmx value is 0 ~ 255, which is the height of fader.

.

.

.

b[33]= 255(0xFF); //dmx value is 0 ~ 255, which is the height of fader.

b[34]= 255(0xFF); //dmx value is 0 ~ 255, which is the height of fader.

UDPSend(b,36); //This is the function of sending udp packets. Send the above 36 bytes to box. Note that sending udp must correspond to port (0x1939)6457.

//---

If it is central control device, send this code of hexadecimal through UDP, UDP port is (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…(16 FF)
Ony to be changed this xx value:	41 56 4C 64 69 79 2E 63 6E 00 00 CE 00 00 xx 00	xx xx	xx xx xx xx xx …(16 xx)
Mean:	Fixed command header cmd port	address	dmx(0~255)
Mean:	A V L d i y . c n 00 00 CE 00 00 02 00	40 00	FF FF FF FF FF…(16 FF)
Program buffer:	b[0] b[1] ...		b[18]

Note: AVLdiy.cn is converted into hexadecimal: 41 56 4C 64 69 79 2E 63 6E.

//---

//##

Example 8:

//This example demonstrates the special command for 3 channels of LED lights, which specifies the changes of all RGB lights in 512 channels of an output port, corresponding to the commands 0xC100, 0xC200, 0xC300, 0xB100, 0xB200 and 0xB300. 0xC100, 0xC200 and 0xC300 will not change the values of other channels, but will superimpose the original color values; 0xB100, 0xB200, 0xB300 will clear the original color value, and then become monochrome.

unsigned char b[600]; //define a buff. If there is not so much space, you only need to define 20byte.

b[0]='A';   //Box fixed head 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 low byte (RGB color change command, which changes all 512 channels of RGB of the specified output port into xxx value)
b[11]= 0xC1; //Command high byte (C1 red, C2 green, C3 blue / B1 red, B2 green, B3 blue)
b[12]= 0; //Length low byte, which is invalid under this command. The default box is 512 by default, which can be 0 by default.
b[13]= 0; //Length is high, which is invalid under this command. The default box is 512 by default, which can be 0 by default.
b[14]= 0 ~ 7; //The low byte of the output port specifies the universe value of the port, and the default value of the box is 0 ~ 7.
b[15]= 0; //Universe high byte. If the output port is greater than 255, it needs to be carried and converted into 2 bytes. Fill here, and the default value is 0.
b[16]= 0; //Channel low byte, which is invalid under this command. The default value is 0.
b[17]= 0; //Channel high byte, which is invalid under this command. The default value is 0.
b[18]= 0 ~ 255; //dmx value is 0 ~ 255, which is the height of the fader (the brightness value of the color)

UDPSend(b,20); //This is the function of sending udp packets. Send the above 20 bytes and the box will change. Note that sending udp port must to (0x1939)6457.

//--
If it is central control device, send this code of hexadecimal through UDP, UDP port is (0x1939)6457:
41 56 4C 64 69 79 2E 63 6E 00 00 C1 00 00 xx xx 00 00 xx
Ony to be changed this xx value: 41 56 4C 64 69 79 2E 63 6E 00 00 C1 00 00 xx xx 00 00 xx
Mean: Fixed command header cmd universe dmx(0~255)
Mean: A V L d i y . c n 00 00 C1 00 00 xx xx 00 00 xx
Program buffer: b[0] b[1] ... b[18]

Note: AVLdiy.cn is converted into hexadecimal: 41 56 4C 64 69 79 2E 63 6E.
//--

//###

Example 9:

//This example demonstrates the 3-channel dedicated command of LED lamp, which specifies the RGB values in all 512 channels of an output port.

unsigned char b[600]; //Define a buff. If you don't have so much space, you only need to define 22byte.

b[0]='A'; //Box fixed head 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 low byte (comm=0xB400, RGB color change command, which changes all channels of the specified output port into RGB values)
b[11]= 0xB4; //command high byte
b[12]= 0; //Length low byte, which is invalid under this command. The default box is 512 by default, which can be 0 by default.
b[13]= 0; //Length high byte, which is invalid under this command. The default box is 512 by default, which can be 0 by default.
b[14]= 0 ~ 7; //Universe low byte of the output port, and the default value of the box is 0 ~ 7.
b[15]= 0; //Universe high byte. If the output port is greater than 255, it needs to be carried and converted into 2 bytes. Fill here, and the default value is 0.
b[16]= 0; //Channel low byte, which is invalid under this command. The default value is 0.
b[17]= 0; //Channel high byte, which is invalid under this command. The default value is 0.
b[18]= 0 ~ 255; //R value is 0 ~ 255, which is the height of fader (brightness value of color). Any color can be combined by the values of RGB three primary colors.
b[19]= 0 ~ 255; //G value is 0 ~ 255, which is the height of fader (brightness value of color). Any color can be combined by the values of RGB three primary colors.
b[20]= 0 ~ 255; //B value is 0 ~ 255, which is the height of fader (brightness value of color). Any color can be combined by the values of RGB three primary colors.

UDPSend(b,22); //This is the function of sending udp packets. Send the above 21 bytes to box. Note that sending udp must correspond to port (0x1939)6457.

//----------

If it is central control device, send this code of hexadecimal through UDP, UDP port is (0x1939)6457:

	41 56 4C 64 69 79 2E 63 6E 00 00 B4 00 00 xx xx 00 00 xx xx xx
Ony to be changed this xx value:	41 56 4C 64 69 79 2E 63 6E 00 00 B4 00 00 xx xx 00 00 xx xx xx
Mean:	Fixed command header cmd universe R G B
Mean:	A V L d i y . c n 00 00 B4 00 00 xx xx 00 00 xx xx xx
Program buffer:	b[0] b[1] ... b[18]…b[20]

Note: AVLdiy.cn is converted into hexadecimal: 41 56 4C 64 69 79 2E 63 6E.

//----------

//##

Example 10:

//This example demonstrates how to check whether the box is online. Send this command to the box, and the box will return **OK**, indicating that the box is online.

unsigned char b[12]; //define a buff, only need to define 12byte.

```
b[0]='A';          //Box fixed head 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 low byte (comm=0xFEFE, check whether the box is powered on online)
b[11]= 0xFE;       //command high byte
```

UDPSend(b,12); //Send the above 12 bytes, and the box will return the character **OK**.

//--

If it is central control device, send this code of hexadecimal through UDP, UDP port is (0x1939)6457:

	41 56 4C 64 69 79 2E 63 6E 00 FE FE
Fixed value:	41 56 4C 64 69 79 2E 63 6E 00 FE FE
Mean:	Fixed command header cmd
Mean:	A V L d i y . c n 00 FE FE
Program buffer:	b[0] b[1] ... b[11]

Note: AVLdiy.cn is converted into hexadecimal: 41 56 4C 64 69 79 2E 63 6E.

//--

//##

Example 11:

//This example demonstrates how to restore the box to factory settings.

unsigned char b[12]; //define a buff, only need to define 12byte.

b[0]='A'; //Box fixed head 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]= 0xXX; //command low byte (comm=0xXXXX, Restore the box to factory settings)
b[11]= 0xXX; //command high byte (Dangerous command, you can contact us if you need this command)

UDPSend(b,12); //Send the above 12 bytes, then box will restore.
//--
If it is central control device, send this code of hexadecimal through UDP, UDP port is (0x1939)6457:

41 56 4C 64 69 79 2E 63 6E 00 XX XX

Fixed value:	41 56 4C 64 69 79 2E 63 6E 00 XX XX
Mean:	Fixed command header cmd
Mean:	A V L d i y . c n 00 XX XX
Program buffer:	b[0] b[1] ... b[11]

Note: AVLdiy.cn is converted into hexadecimal: 41 56 4C 64 69 79 2E 63 6E.
//--

Chapter 2: C code implementation method:

Tip: WBOX has two control modes, which can control dmx512 and SPI LED strip. Only 170 RGB lamps with 3 channels can be controlled by DMX, and 170x3=510 channels. Therefore, buf is defined as buf[510] in SPI control mode and buf [512] in DMX control mode.

```
/*
//Use comm=0x0100 to send 512-channel dmx data to the specified output port to fill the example.
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 low byte
b[11] = 0x01;       //command high byte
b[12] = 0xFE;       //Length (low byte) SPI mode filled 0xFE, 0x01FE=510(byte)   / DMX mode filled 00, 0x0200=512(byte)   / or all 0 are filled with default values.
b[13] = 0x01;       //Length (high byte) SPI mode filled 01                      / DMX mode filled 02                      / or all 0, with the default value.
b[14] = Universe;   //Universe low byte, corresponding to PORT0~PORT7 (if it is LED mode, 0~3 will be spliced internally for one line output, and 4~7 for one line output)
b[15] = 0;          //Universe high byte. There are not so many universes in general lighting projects. Fill in 0.
b[16] = 0;          //NON
b[17] = 0;          //NON
b[18] = 0~255       //R0 RGB value of point 1
b[19] = 0~255;      //G0
b[20] = 0~255;      //B0
b[21] = 0~255       //R1 RGB value of point 2
```

```
b[22] = 0~255;          //G1
b[23] = 0~255;          //B1
...
b[525] = 0~255          //R169 RGB value at the 170th point
b[526] = 0~255;         //G169
b[527] = 0~255;         //B169
*/
```

C example specific implementation code:

```
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include <…>        //Include header files for your own development platform.
…
#define DMX_MODE 1 //1 is DMX mode, and 0 is SPI LED mode.
#if DMX_MODE
    #define BUF_SIZE 512
#else
    #define BUF_SIZE 510
#endif
```

/* Fill in the examples yourself

```
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 low bit
b[11] = 0x01;        //command high bit
b[12] = 0x00;        //Length (low byte) The length of RGB array, SPI mode filled 01, DMX mode filled 02 / or all filled 0 the default value is 510 or 512.
b[13] = 0x00;        //Length (high byte) SPI mode filled 0xFE 0x01FE=510(byte)    / DMX mode filled 00, 0x0200=512(byte) /or all zeros are filled with default values.
b[14] = universe;    //Universe (In SPI mode, port 0~3 will be spliced inside the box for one line output, and port 4~7 will be one line output)
                     //DMX mode, it corresponds to PORT0~PORT7 output port.
b[15] = 0;           // Universe is high. There are not so many universe in general lighting projects. Fill in 0.

memcpy(&b[18], dmxdata_buf, BUF_SIZE); //b[18] is the starting position of dmx data filling.

//Send it to our box through udp packet. This function needs to be written by itself
//The port where udp packets are sent must be 0x1939=6457, and IP can be unicast or broadcast (IP=255.255.255.255)
udpsend(b, sizeof(b));
}
*/

//---Fill in the general function with the structure-
void send_to_wbox(unsigned char universe, unsigned char *dmxdata_buf)
{
unsigned char b[540]={0};                    //It doesn't matter if there are a few extra bytes.
struct my_dmx *dmx = (struct my_dmx *)&b[0];        //Structure
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;         //data length, SPI mode =510, DMX mode =512 / You can also fill in 0, the default value of 510 or 512 in box of box mode.
dmx->Universe = universe;       //SPI LED mode, universe 0~3 will be spliced into one line output and universe 4~7 will be one line output.
                                //Each line controls 680 3-channel RGB pixels.    DMX mode, it corresponds to PORT0~PORT7 output port.

memcpy( &dmx->DmxData_Val_Start, dmxdata_buf, BUF_SIZE);

//Send it to our box through udp packet. This function needs to be written by itself
//The port where udp packets are sent must be 0x1939=6457, and IP can be unicast or broadcast (IP=255.255.255.255)
udpsend(b, sizeof(b)); //according to your own platform, write your own udp packet function.
}


int main()
{
    init_udp();      //according to your own platform, write and initialize udp packet function.

    //define an array to store all channels 4096 data of port 0~7, each universe has 510 bytes (LED SPI mode) or 512 bytes (DMX mode),
    //and each universe controls 170 pcs 3-channel RGB pixel lamps.
    // dmxdata[0] ~dmxdata[511] saves the output value of universe 0/PORT0.
    // dmxdata[512] ~ dmxdata[1023] saves the output value of universe 1/PORT1.
    // dmxdata[1024]~dmxdata[1535] saves the output value of universe 2/PORT2.
```

```
// ……
unsigned char dmxdata[ 8 * BUF_SIZE ] = {0};          // dmxdata[4096] saves dmx values of 4096 channels of all 8 output ports.

memset( dmxdata, 255, 8 * BUF_SIZE );                 //All the values in the array are 255, all lighting bright.
for(int i=0;    i<8; i++){
      send_to_wbox(i, &dmxdata[BUF_SIZE * i] );       //Send 8 universe to the box, and the box will change according to the value of dmxdata.
}

Sleep(5); //delay 5s to see the effect.

memset( dmxdata, 0, 8* BUF_SIZE );                    //clear all, all values in the array are 0.
for(int i=0;    i<8; i++){
      send_to_wbox(i, &dmxdata[BUF_SIZE * i] );       // Send 8 universe to the box, and the box will change according to the value of dmxdata.
}

Sleep(5); //delay 5s to see the effect.

memset( &dmxdata[BUF_SIZE *2], 255, BUF_SIZE); //The 2nd universe is fully lit.
send_to_wbox( 2, &dmxdata[BUF_SIZE *2 ] );   //Send the value of the second field to the box, and the box will change according to the value of dmxdata.

Sleep(5); //delay 5s to see the effect.

memset( &dmxdata[BUF_SIZE *2], 0, BUF_SIZE ); //The second universe all clear
send_to_wbox( 2, &dmxdata[BUF_SIZE *2 ] );          //Send the value of the second field to the box, and the box will change according to the value of dmxdata.

Sleep(5); //delay 5s to see the effect.

//The 5th universe only lights up with a red light
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 ] ); //Send the value of the 5th field to the box, and the box will change according to the value of dmxdata.

    //The 4th universe is only green.
    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 ] ); //Send the value of the 4th field to the box, and the box will change according to the value of dmxdata.

    //The 7th universe only lights blue.
    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 ]); //Send the value of the 7th field to the box, and the box will change according to the value of dmxdata.

//Tip: What you really need to do is change the value in dmxdata[4096] and send it to the box. You can use a timer and send it every 30ms. Then you don't need to pay attention to the network transmission, just change the value in dmxdata[4096] according to the lighting effect, so that you can make different lighting effects. Be good at using sin, cos functions to change dmx value.

}//end main
```

```
//-all reset, turn off the lights function, simple method-
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);
}
//-------------------------------------------------------------
```

Chapter 3: Web page secondary development mode:

Please visit our website http://box.AVLdiy.cn download the source code to watch, or browse the box. http://192.168.7.1/sdkdemo.html watch the source code.
This kind of development method is very powerful. As long as you can make web pages, you can develop them. Many of our customers have developed many powerful applications in this way. WBOX itself is a server, which can run JavaScript. There are many open source JS on the Internet that can be used for reference. Around lighting control, many integrated functions such as graphics, images, 3D, audio, fountain, project management, time management, task planning, etc. can be developed. Only one of our boxes is needed to complete them all.

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
//-------------------------------------------------------------
```

Chapter 4: Control mode of text command in central control:

We have compiled a program ccplay for WBOX, which is specially used for centralized control of command text. If you want to use Crestron, AMX and other domestic brands' centralized control systems to control lighting by sending udp text commands, you can upgrade this firmware WBOX_Crestron_AMX_v2.3.1.102_20210603.bin, This firmware will automatically run the ccplay centralized control command receiving program, and the lights can be controlled by sending udp packets through the centralized control system. The contract issuing command is ASCII text. Note that it is ASCII text, not numbers, and not HEX. The port of udp packets is **6799**. If the firmware is of other versions, when wbox is turned on, ccplay won't run automatically. Therefore, if other versions of firmware want to start ccplay automatically, just edit the startup file "/etc/rc.local" and add the program ccplay we need to start before "exit 0". When the box starts, everything in this startup file will run automatically.

The command format is as follows:

All output ports output the same dmx values, the command format is:

PUSH address first channel value second channel value third channel value ...

All output ports, all faders output the same value, like to our web version of AllTo255, AllTo0 command, and the format is:

SETV 255/SETV 0/SETV 128 ...

Assign the output port of the box and output the dmx value. The command format is:

PORT[0 ~ 7] address first channel value second channel value third channel value ...

Note that 0 corresponds to the DMX0 output port and 7 corresponds to the DMX7 output port.

Assign the output port of the box to output the values of all 512 faders (all channels). The command format is:

SETP[0 ~ 7] 0 / SETP2 255 / SETP7 128

Note that 0 corresponds to the first DMX0 output port and 7 corresponds to the DMX7 output port.

For example, there is a lamp whose channel properties are as follows:

The first channel is: X

The 2nd channel is: Y

The 3rd channel is: XY precision
The fourth channel is: dim
The 5th channel is: R
The 6th channel is: G
The 7th channel is: B

18 sets fixture were connected, and the address codes of the fixture were 1, 17, 33, 49, 65, 81, 97, 113, 129, 145, 161, 177, 193, 209, 225, 241, 257, 273. ...

Control the first fixture X to turn full, Y to turn half a turn, and the red light is on. Send the text to port 6799 through UDP. Just send this string of text:
PUSH 1 255 128 0 255 255

Control the 2nd desk fixture to rotate by half X, turn by full Y, the blue light is on, Send the text to port 6799 through UDP. Just send this string of txt text:
PUSH 17 128 255 0 255 0 0 255

Control the 4th fixture to turn on the green light only, send the text to port 6799 through UDP, and send this string of txt text:

	PUSH	**49**	**0**	**0**	**0**	**255**	**0**	**255**	**0**
Mean:	command	fixture4 address	x	y	xy	dim	R	G	B

Suppose a fixture is connected to the output port DMX7, the address is 257, and it needs XY rotation. Send udp packet:
PORT7 257 255 255

Suppose a fixture is connected to the output port DMX3, the address is 177, and it needs bright RGB. Send udp packet:
PORT3 177 0 0 0 255 255 255 255

Turn off all lights of 8 outputs, and all 4096 channels dmx value become 0: **SETV 0**
Only turn off all lights of dmx4 output port, and all 512 channels of dmx4 are cleared to 0: **SETP4 0**
All 512 channels of dmx2 output port are the brightest: **SETP2 255**

Chapter 5: Linux Shell development methods:

This kind of development method is so powerful that it has no friends. It is recommended to use it, especially when combined with websdk. JS can make many tall applications by making interactive pages. Many of our customers have developed many successful cases under our guidance.

We have developed many programs for the box, all of which are packed in the box, and can be combined by shell or script, such as light, port, port_new, effects, light play, catch_devfile, play_devfile … and so on. By combining these programs like building blocks through scripts, many complex tasks can be completed.

Specifically, you can log in to the box for development through the terminal, or log in to the box for development through our webui. The box has a vi editor, which can be edited directly, or you can edit it locally according to your own meaning through editing software such as Notepad and VS, upload it to the box through terminal software such as winscp and ssh, and at the same time, make a running permission.

This kind of development method is too powerful to write specific development examples. If you have specific projects, you can contact me if you are unclear.

If you need to add any function, you can also contact us. As long as you can suggest it, basically we can do it!

Note: For the early firmware, these commands and scripts can also be called or run through udp. The method is to run the ~~**netcomm**~~ program(udp port is **5688**), and then you can directly send commands through udp to run the programs in the shell or box. For specific methods, please watch our video teaching update chapter: (http://down.avldiy.cn/down_server/W-BOX/video_teaching/)

Please ignore this point in the firmware after the new version V3.5.1.102, and look at the next chapter. There are more advanced methods and procedures that can be used.

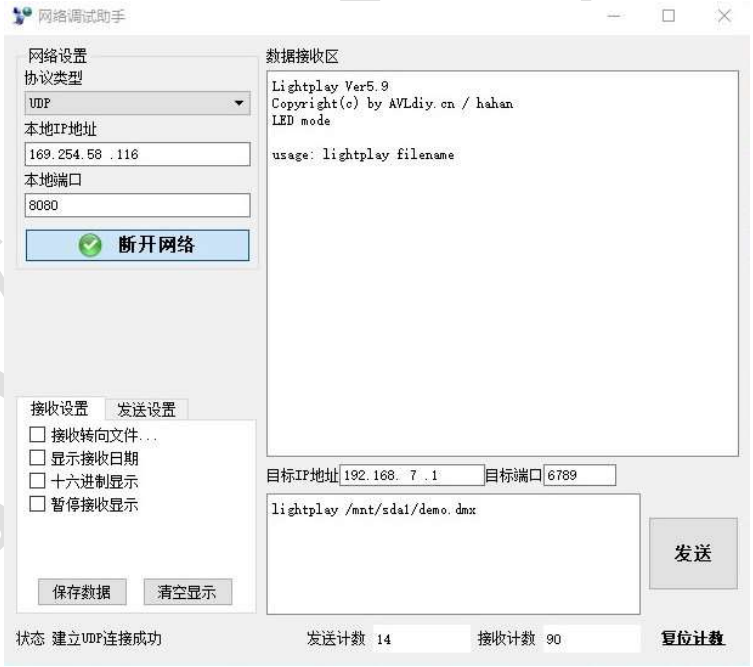
Chapter 6: UDP Shell development mode:

Many friends who do system integration and central control projects prefer to call the scenes in the play box by sending simple commands. The box can capture different lighting effects into dmx files by capturing mode, audio and dmx files can be played synchronously by lightplay, dmx files can be played separately by play_devfile, many RGB effects can be produced by effects, and dmx values of different lights can be set by light program, so a general method that can be called externally is needed. For this reason, we have developed such a program separately, which can directly run the linux shell through udp, and directly move the linux shell to you through udp. Now, you can run commands, programs, app, scripts, check and monitor devices, etc. by sending udp. This function is integrated inside the box of the new firmware, and you can directly use it out of the box.

Before using this function, we suggest that you have a certain command line operation foundation, such as having the experience of cmd input command in windows and Linux shell input command. If you have no similar experience, you can imagine the input command, which is actually similar to the program that you double-click your desktop in windows, and then windows will open this program and run it.

Note before use: Note that the port of udp packet is **6789**, it's not same RAW udp port. Below, we simulate the central control system through the network debugging assistant. Of course, the upper device can also be any other device that can send network udp.

Example: Suppose there is an offline file of demo.dmx on the USB disk. How can it be called through udp?



As shown above, set IP: 192.168.7.1, port: 6789, and send command: **lightplay /mnt/sda1/demo.dmx** to the box.

Mean: The above meaning is to tell the box to run this statement. Among them, **/mnt/sda1** represents the USB disk device. Note that everything in linux is a file, and the hardware device is also a file, which is quite different from Windows. **lightplay** is a special player written by us. For specific usage, please see our early video introduction (http://down.avldiy.cn/down_server/W-BOX/video_teaching/) 。

After running this statement, the box will send back the running information through udp. The information sent back is equivalent to your monitor. In the network debugging assistant, enter the command below, and the command running result will be displayed above. When debugging, you can pay attention to these displayed information, which is very helpful, and you can ignore this information when using it. For example, if we want PORT2, 3 and 7 to output the 14th effect with a built-in **effects** of 20 seconds, what should we do? There are too many effects parameters in the built-in effect program, so I forgot it. You can help it by adding -h parameter first. The specific usage of effects can be found in our early video introduction (http://down.avldiy.cn/down_server/W-BOX/video_teaching/) 。

Enter the command: **effects -h**, which will return help information on how to use the method. Note: The new version of **effects1.2** adds the **-u** parameter, which can assign an output port to output effects without affecting other output ports. This parameter must be enclosed in single or double quotes, and each output port that needs to be output is separated by a space. The **-t** parameter is the setting effect running time in seconds, and **-f** is the number of different effects.

Example 2: Port 2, 3, and 7 output the 14th effect of the built-in effects for 20 seconds.

Udp send this command: **effects -t20 -f14 -u"2 3 7"**

Note: All commands can be superimposed, that is, they can be run again. For example, while the above 2, 3 and 7 output effects, we can continue to superimpose them, and let the output ports 1 and 4 run other effects. For example, let the output ports 1 and 4 output effects 9 for 30 seconds, with random colors. Just keep sending:

Effects -t30 -f9 -u"1 4" -C

Example 3: It's still the example of Example 2. I don't want to send multiple commands one by one. How can I do it?

Just use semicolons to separate commands from each other. Note that they are English semicolons, not Chinese semicolons. All commands must be entered in English without Chinese characters. If the text sent by udp can include newlines, one command per line, and each line can be divided, then semicolons can be used. If not, extra marks should be given, so the above command will look like the following:

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

Example 4: In this example of Example 3, after the effect playing time is up, the output state of the last step will be retained, and the final brightness will continue to be retained. What should we do if we want to turn it off?

You can use the **light** command to set the dmx value of the channel to 0 to close it. See our early videos for the specific usage of **light** (http://down.avldiy.cn/down_server/W-BOX/video_teaching/) 。

The udp command is as follows:

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

Is this string of commands familiar? Haha, isn't this the script! Yes, now you can write scripts through udp, and you do not include the header in the script. Let's rearrange this string of codes, which is what your usual script looks like.

effects -t20 -f14 –u”2 3 7” &

effects -t30 –f9 -u”1 4” -C ;

light 8 s 1 0

The above is replaced with an ampersand ; (semicolon), why?

This involves a concept of linux, the problem of blocking and non-blocking. The so-called blocking means that a program will not quit immediately after running, but it needs to quit after running completely. For example, if the above **effects -t20** runs for 20 seconds, it will take 20 seconds to quit. Before quitting, it will block the running of the following programs, and only after it quits, will it run the following statements one by one. To solve this problem, we can add an **&** After the ampersand is added, you

can't add extra points.

But the above statement is still problematic, because after 20 seconds, the effects of PORT2, 3, and 7 stopped. We want to turn off the brightness of output ports 2, 3, and 7 immediately, instead of keeping it as the brightness value of the last effect. But the above statement can't do it, and it will remain for 10 seconds until the **effects- t30 -f8 -u "1 4" -C** exits, for 30 seconds.

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

Mean: We let effects run in the background mode, without blocking other statements, and then start timing. After 20 seconds, turn off outputs 2, 3 and 7, and then time again. After 10 seconds, turn off all outputs, with a total duration of 30 seconds. Here, the delay is increased by 1~2 seconds, because the delay of shell is not accurate, and the system scheduling will take a little time. In order to ensure the error of the exit time of effects, and to turn off the output correctly, add 1~2 seconds to ensure the shutdown. If the time-sensitive synchronization project can be timed by other methods, it can reach microsecond level. For example, the delay program **mysleep** written by us can reach microsecond level.

Example 5: How to close the running program? Assuming that **effects** is running all the time, we want to close it in advance. What should we do? Just send the following command from udp.

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

Mean: After any program or script runs, there is a PID. Find this PID and kill it to close the program. Note that there are two oblique dots enclosed after the kill. This dot is the dot in front of the keyboard number 1. Please pay attention. What if you want to turn off **lightplay**? What should you do? Just change the name of the program, as follows:

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

Example 6: How to check whether the box is turned on or healthy?

udp only needs to send **hello**, and the box will return the character **OK**. When OK is detected, it means that the box is turned on and healthy.

Udp send this text: **hello** , wbox reply: **OK**

Example 7: How to run the script saved in the box or the script written by yourself?

Just enter the script path and name, send it through udp, and it will run. The operation is exactly the same as that of the shell. For example, the box /mydemo directory stores a test program **test.sh**. Just send this statement through udp: **/mydemo/test.sh**

Example 8: How to run the commands in linux? ls, mkdir, PS, etc.

It's ok to send commands directly from udp, but it should be noted that blocking commands and programs can only return information when they exit. For example, the top command is blocked all the time, and the returned information can only be seen when you exit the top. However, after the top runs, it won't exit unless it is shut down artificially, so it is recommended to use **ps** command to query the running process. The above **example 5** is **ps** command to find out all processes, then format and display PID, and finally kill them.

……

There are many, many examples ..., as long as you know the basic operation of linux, you can play many, many tricks. There is no protocol problem with this calling method. Everything is up to you, which can give full play to your freedom and flexibility. It is very suitable for third parties to use, and it is a secondary development method that we highly recommend.

As far as we know, this kind of calling method should not be available in the market at present, and it is our original sharp weapon. You are welcome to use our sharp weapon to develop more useful things on wbox lighting platform and shine brilliantly for your project.

Chapter 7: Capturing dmx files (supplementary documents)

External calls often require the capture dmx function. There are 2 sets of capture systems in wbox, among which the capture function in webui is to capture and save 8 outlets at the same time. It captures the data sent by external console or software through artnet and sCAN or the professional software we support. The captured data will be encrypted and must be a whole, so it is impossible to capture only one outlet, but only 8 outlets at a time. These data can't be split, and will be garbled after splitting.

Another set of capture system is captured by catch_devfile software, and the corresponding playback program is play_devfile. This set of software can capture one or several output ports separately. This set of capture program mainly captures the data developed by itself, such as the data sent by the command of RAW protocol com=0x0100, or the data generated by our effect generator effects, or the effects generated by our own script program. These data can be split, each can be independent, and the data is simply deformed.

Contact information:
http://box.AVLdiy.cn
Email: 1195722899@qq.com
Hahan(XieWenCai):13808858586

2022-12-13