



Generation-4 SBUS

SMART-BUS Protocol

Version: V1.4

Updated Date: March 15, 2012

Website: www.SmartHomeBUS.com

I Definition of Protocol Base Structure

	Start Code	LEN of Data Package	Original Subnet ID	Original Device ID	Original Device Type	Operation Code	Target Subnet ID	Target Device ID	Additional Content	CRC H (higher 8 bit)	CRC L (lower 8 bit)
Data Type	16bit	8Bit	8Bit	8Bit	16Bit	16Bit	8Bit	8Bit	0-N	8Bit	8Bit
Data Range	0xAAAA	13-78	0-254	0-254	0-0FFFF	0-0xFFFF	0-254	0-254	0-N bytes		
SN	1	2	3	4	5	6	7	8	9	10	

Start code

Start code is starting symbol of data package and fixed format is 0xAAAA, it will start to receive the whole package when the receiver get the fix format from the data and take a data as length of data package.

LEN of Data Package

this one is showing how many bytes for the data package.

Data Range: 11-78.

How to calculate?

From SN 2 to 10, it's not included SN 1.

Original subnet ID & Original device ID

Address of the device which sends the data package, value scope is 0-254

Address includes 2 parts, subnet ID & Device ID.

Original Device Type

Type of original device, different module has different device type; please see the definition table below. Value scope is 0-65535

Operation code

Operation codes specify all functions & commands of system. From 0-65535

Target subnet ID & Target Device ID

Address of the device which will receive the data package.

Data scope is 0-255

if subnet ID and Device ID are both 255, it means broadcast, the data package will be received by all the modules.

Additional Content

The data of Additional content is variable, it depends on Operation Code, different operation code might has different content.

CRC H & CRC L

CRC verification code, to check data lost or not.

How to get CRC Data?

We will get the data from SN 2 to 9, and then use CRC arithmetic to generate the verification code.

We will explain the detail below.

2. How to use this protocol?

2.1. Device address must be unique on the same network.

2.2 Operation code

for example:

Lighting Scene Control is 0x0002,

Lighting Single Channel control is 0x0031

Lighting Sequence Control is 0x001A

2.3 Additional content

This is related to operation code.

A. Lighting Scene Control

Additional Content for Lighting Scene Control		Operation Code: 0x0002
Index of Additional Content	Remark	
0	Area No	
1	Scene No	

Reply from device for scene control

Reply from device		Operation Code: 0x0003
Index of Additional Content	Remark	
0	Area No	
1	Scene No	

B. Lighting Single Channel Control

Additional Content for Lighting Channel Control		Operation Code: 0x0031
Index of Additional Content	Remark	Value
0	Light Channel No	1byte
1	Brightness Level	1byte 0-100
2	High 8 bit of Running time	2 bytes MAX. running time is
3	Low 8 bit of Running Time	3600s

Reply from device for Lighting Single control

Reply from device		Operation Code: 0x0032
Index of Additional Content	Remark	Value
0	Channel No	1byte
1	Sign for success or failure	1byte, Success:0xF8 failure:0xF5
2	Brightness Level	1byte, 0-100
3	QTY of Channels	1byte
4	Status of channels	N bit, N is the QTY of Channels

C. Lighting Sequence Control

Additional Content for Lighting Sequence Control		Operation Code: 0x001A
Index of Additional Content	Remark	Value
0	Area No	1byte
1	Sequence No	1byte

Reply from device for Lighting Sequence control

Reply from device		Operation Code: 0x001B
Index of Additional Content	Remark	Value
0	Area No	1byte
1	Sequence No	1byte

For example:

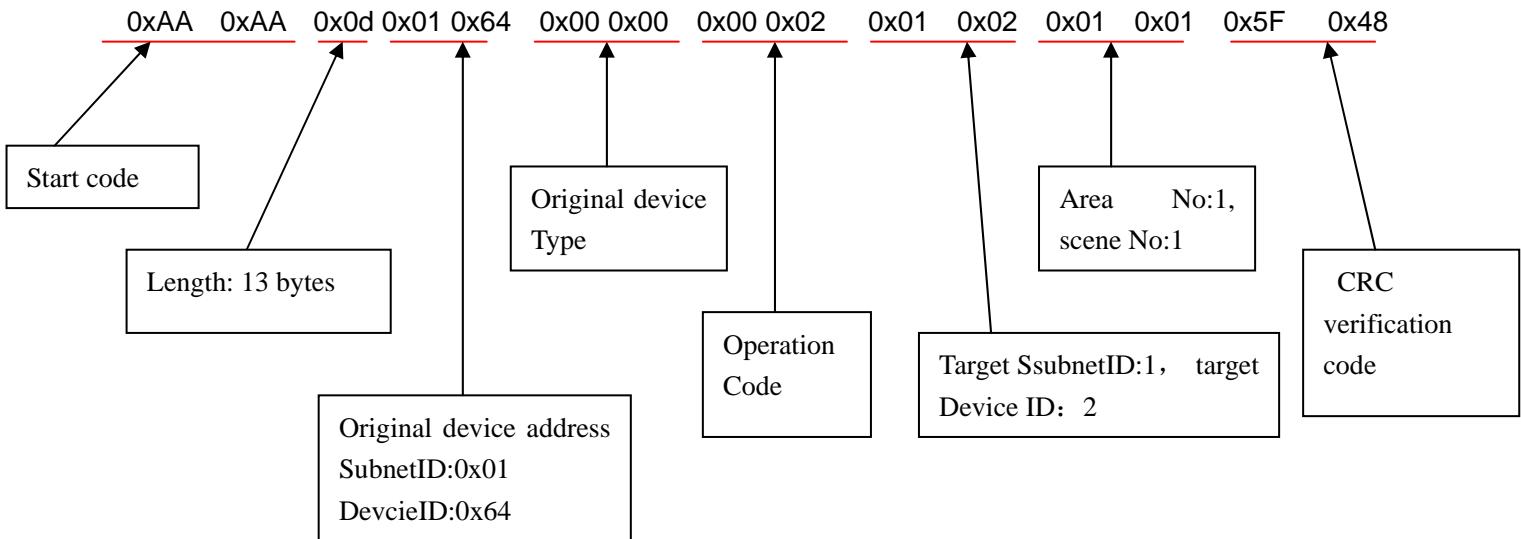
We have a dimmer, the subnet ID is 1, the device ID is 2;
Switch Panel, Subnet ID is 1, device ID is 64, device Type is 0x0000.

We want to control the dimmer to open the lighting scene by Switch Panel.

So Switch Panel is Original device, dimmer is target device.

Operation code of Scene Control is 0x0002

so the command is as following:



3. RS232 & RS485 commutation

RS232:9600bps, 11bit: initial , 8 data bit, efficacy bit, stop bit

CRC programming are written by C language

```
uchar Check_crc(uchar *ptr, uchar len)
{
    uint crc;
    uchar dat;
    crc=0;
    while(len--!=0)
    {
        dat=crc>>8;          /* */
        crc<<=8;              /* */
        crc^=CRC_TAB[dat^*ptr];/* */
        ptr++;
    }
    dat=crc;
    if((*ptr==(crc>>8))&&(*ptr+1)==dat))
        return(TRUE);
    else
        return(FALSE);
}
```

```
void Pack_crc(uchar *ptr, uchar len)
{
    uint  crc;
    uchar dat;
    crc=0;
```

```

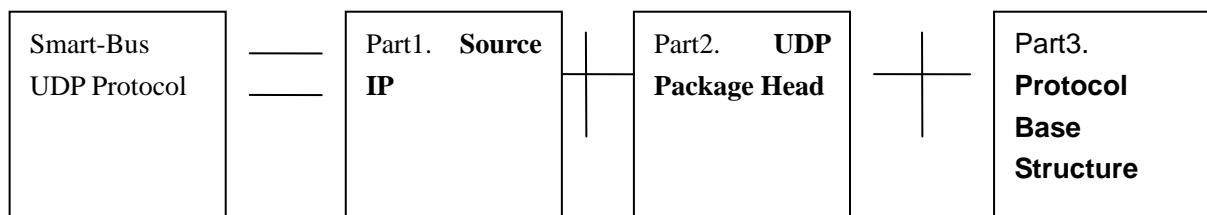
while(len--!=0)
{
    dat=crc>>8;           /* */
    crc<<=8;               /* */
    crc^=CRC_TAB[dat^*ptr]; /* */
    ptr++;
}
*ptr=crc>>8;
ptr++;
*ptr=crc;
}

unsigned int  CRC_TAB[]={          /* CRC tab */
    0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50a5, 0x60c6, 0x70e7,
    0x8108, 0x9129, 0xa14a, 0xb16b, 0xc18c, 0xd1ad, 0xe1ce, 0xf1ef,
    0x1231, 0x0210, 0x3273, 0x2252, 0x52b5, 0x4294, 0x72f7, 0x62d6,
    0x9339, 0x8318, 0xb37b, 0xa35a, 0xd3bd, 0xc39c, 0xf3ff, 0xe3de,
    0x2462, 0x3443, 0x0420, 0x1401, 0x64e6, 0x74c7, 0x44a4, 0x5485,
    0xa56a, 0xb54b, 0x8528, 0x9509, 0xe5ee, 0xf5cf, 0xc5ac, 0xd58d,
    0x3653, 0x2672, 0x1611, 0x0630, 0x76d7, 0x66f6, 0x5695, 0x46b4,
    0xb75b, 0xa77a, 0x9719, 0x8738, 0xf7df, 0xe7fe, 0xd79d, 0xc7bc,
    0x48c4, 0x58e5, 0x6886, 0x78a7, 0x0840, 0x1861, 0x2802, 0x3823,
    0xc9cc, 0xd9ed, 0xe98e, 0xf9af, 0x8948, 0x9969, 0xa90a, 0xb92b,
    0x5af5, 0x4ad4, 0x7ab7, 0x6a96, 0x1a71, 0x0a50, 0x3a33, 0x2a12,
    0xdbfd, 0xcbdc, 0xfbff, 0xeb9e, 0x9b79, 0x8b58, 0xbb3b, 0xab1a,
    0x6ca6, 0x7c87, 0x4ce4, 0x5cc5, 0x2c22, 0x3c03, 0x0c60, 0x1c41,
    0xedae, 0xfd8f, 0xcdec, 0xddcd, 0xad2a, 0xbd0b, 0x8d68, 0x9d49,
    0x7e97, 0x6eb6, 0x5ed5, 0x4ef4, 0x3e13, 0x2e32, 0x1e51, 0x0e70,
    0xff9f, 0xefbe, 0xdfdd, 0xcffc, 0xbf1b, 0xaf3a, 0x9f59, 0x8f78,
    0x9188, 0x81a9, 0xb1ca, 0xa1eb, 0xd10c, 0xc12d, 0xf14e, 0xe16f,
    0x1080, 0x00a1, 0x30c2, 0x20e3, 0x5004, 0x4025, 0x7046, 0x6067,
    0x83b9, 0x9398, 0xa3fb, 0xb3da, 0xc33d, 0xd31c, 0xe37f, 0xf35e,
    0x02b1, 0x1290, 0x22f3, 0x32d2, 0x4235, 0x5214, 0x6277, 0x7256,
    0xb5ea, 0xa5cb, 0x95a8, 0x8589, 0xf56e, 0xe54f, 0xd52c, 0xc50d,
    0x34e2, 0x24c3, 0x14a0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
    0xa7db, 0xb7fa, 0x8799, 0x97b8, 0xe75f, 0xf77e, 0xc71d, 0xd73c,
    0x26d3, 0x36f2, 0x0691, 0x16b0, 0x6657, 0x7676, 0x4615, 0x5634,
    0xd94c, 0xc96d, 0xf90e, 0xe92f, 0x99c8, 0x89e9, 0xb98a, 0xa9ab,
    0x5844, 0x4865, 0x7806, 0x6827, 0x18c0, 0x08e1, 0x3882, 0x28a3,
    0xcb7d, 0xdb5c, 0xeb3f, 0xfb1e, 0x8bf9, 0x9bd8, 0xabbb, 0xbb9a,
    0x4a75, 0x5a54, 0x6a37, 0x7a16, 0x0af1, 0x1ad0, 0x2ab3, 0x3a92,
    0xfd2e, 0xed0f, 0xdd6c, 0xcd4d, 0xbdaa, 0xad8b, 0x9de8, 0x8dc9,
    0x7c26, 0x6c07, 0x5c64, 0x4c45, 0x3ca2, 0x2c83, 0x1ce0, 0x0cc1,
    0xef1f, 0xff3e, 0xcf5d, 0xdf7c, 0xaf9b, 0xbfbba, 0x8fd9, 0x9ff8,
    0x6e17, 0x7e36, 0x4e55, 0x5e74, 0x2e93, 0x3eb2, 0x0ed1, 0x1ef0
};

```

4. Smart-BUS UDP Protocol

Structure of Smart-Bus UDP protocol



Source IP: the IP of device which send the UDP data package.

UDP Package Head :

ASCII of SMARTCLOUD

SMART – BUS UDP communication protocol is based on Winsock to communicate, following take Delphi as an example to explain socket.

- A. Initialize Socket: port: 6000
- B. Sending data package: Using SendTo function
- C receiving data package: using recvfrom function

maraySendUDPBuf : data type is Array of byte

Part1.Source IP

```

maraySendUDPBuf[0]:=mbytlPPart1;{eg. Source IP: 192.168.18.5, so mbytlPPart1=192}
maraySendUDPBuf[1]:=mbytlPPart2;{ here is 168}
maraySendUDPBuf[2]:=mbytlPPart3; {here is 18 }
maraySendUDPBuf[3]:=mbytlPPart4; {here is 5 }

```

UDP Package Head:

```

maraySendUDPBuf[4]:=$53;{S}
maraySendUDPBuf[5]:=$4D;{M}
maraySendUDPBuf[6]:=$41;{A}
maraySendUDPBuf[7]:=$52;{R}
maraySendUDPBuf[8]:=$54;{T}
maraySendUDPBuf[9]:=$43;{C}
maraySendUDPBuf[10]:=$4C;{L}
maraySendUDPBuf[11]:=$4F;{O}
maraySendUDPBuf[12]:=$55;{U}

```

```
maraySendUDPBuf[13]:=$44;{D}
```

Part3.Protocol Base Structure

```
maraySendUDPBuf[14]=$AA; { leading code }
maraySendUDPBuf[15]=$AA; { leading code}
maraySendUDPBuf[16]=$0F; { data package length }
maraySendUDPBuf[17]=$01; {original subnet ID}
maraySendUDPBuf[18]=$FA; { original device ID }
maraySendUDPBuf[19]=$FF; {original device type: higher then 8 }
maraySendUDPBuf[20]=$FE; { original device type: lower then 8 }
maraySendUDPBuf[21]=$00; { Operation code: higher then 8 }
maraySendUDPBuf[22]=$31; { Operation code: lower then 8}
maraySendUDPBuf[23]=$01; {subnet ID of targeted device }
maraySendUDPBuf[24]=$44 { device ID of targeted device }
maraySendUDPBuf[25]=$01; { additional, channel No }
maraySendUDPBuf[26]=$46; { additional, intensity }
maraySendUDPBuf[27]=$00 { additional, channel running time, higher then 8 }
maraySendUDPBuf[28]=$00; { additional, channel running time, lower then 8 }

maraySendUDPBuf[29]= $1D; {CRC, higher then 8 }
maraySendUDPBuf[30]= $A3; {CRC, lower then 8}
```

And then we can user the following function to send the UDP data package.

[SendTo\(moSocket,maraySendUDPBuf,intUDPBufLen,0,moSockAddrIn,sizeof\(moSockAddrIn\)\);](#)

5. code for CRC verification (for Delphi)

5.1. CRC table

```
{-----CRC table-----}
constCRCTab:array[0..255] of word=(
$0000, $1021, $2042, $3063, $4084, $50a5, $60c6, $70e7,
$8108, $9129, $a14a, $b16b, $c18c, $d1ad, $e1ce, $f1ef,
$1231, $0210, $3273, $2252, $52b5, $4294, $72f7, $62d6,
$9339, $8318, $b37b, $a35a, $d3bd, $c39c, $f3ff, $e3de,
$2462, $3443, $0420, $1401, $64e6, $74c7, $44a4, $5485,
$a56a, $b54b, $8528, $9509, $e5ee, $f5cf, $c5ac, $d58d,
$3653, $2672, $1611, $0630, $76d7, $66f6, $5695, $46b4,
$b75b, $a77a, $9719, $8738, $f7df, $e7fe, $d79d, $c7bc,
$48c4, $58e5, $6886, $78a7, $0840, $1861, $2802, $3823,
$c9cc, $d9ed, $e98e, $f9af, $8948, $9969, $a90a, $b92b,
$5af5, $4ad4, $7ab7, $6a96, $1a71, $0a50, $3a33, $2a12,
$dbfd, $cbdc, $fbff, $eb9e, $9b79, $8b58, $bb3b, $ab1a,
$6ca6, $7c87, $4ce4, $5cc5, $2c22, $3c03, $0c60, $1c41,
$edae, $fd8f, $cdec, $ddcd, $ad2a, $bd0b, $8d68, $9d49,
$7e97, $6eb6, $5ed5, $4ef4, $3e13, $2e32, $1e51, $0e70,
```

\$ff9f, \$efbe, \$dfdd, \$cffc, \$bf1b, \$af3a, \$9f59, \$8f78,
 \$9188, \$81a9, \$b1ca, \$a1eb, \$d10c, \$c12d, \$f14e, \$e16f,
 \$1080, \$00a1, \$30c2, \$20e3, \$5004, \$4025, \$7046, \$6067,
 \$83b9, \$9398, \$a3fb, \$b3da, \$c33d, \$d31c, \$e37f, \$f35e,
 \$02b1, \$1290, \$22f3, \$32d2, \$4235, \$5214, \$6277, \$7256,
 \$b5ea, \$a5cb, \$95a8, \$8589, \$f56e, \$e54f, \$d52c, \$c50d,
 \$34e2, \$24c3, \$14a0, \$0481, \$7466, \$6447, \$5424, \$4405,
 \$a7db, \$b7fa, \$8799, \$97b8, \$e75f, \$f77e, \$c71d, \$d73c,
 \$26d3, \$36f2, \$0691, \$16b0, \$6657, \$7676, \$4615, \$5634,
 \$d94c, \$c96d, \$f90e, \$e92f, \$99c8, \$89e9, \$b98a, \$a9ab,
 \$5844, \$4865, \$7806, \$6827, \$18c0, \$08e1, \$3882, \$28a3,
 \$cb7d, \$db5c, \$eb3f, \$fb1e, \$8bf9, \$9bd8, \$abbb, \$bb9a,
 \$4a75, \$5a54, \$6a37, \$7a16, \$0af1, \$1ad0, \$2ab3, \$3a92,
 \$fd2e, \$ed0f, \$dd6c, \$cd4d, \$bdaa, \$ad8b, \$9de8, \$8dc9,
 \$7c26, \$6c07, \$5c64, \$4c45, \$3ca2, \$2c83, \$1ce0, \$0cc1,
 \$ef1f, \$ff3e, \$cf5d, \$df7c, \$af9b, \$bfba, \$8fd9, \$9ff8,
 \$6e17, \$7e36, \$4e55, \$5e74, \$2e93, \$3eb2, \$0ed1, \$1ef0);

5.2. Get two value of CRC:

```

//-----
procedure PackCRC(arayPtrBuf:array of byte;intBufLen:integer);
var
  wdCRC:word;
  wdPtrCount:word;
  bytDat:byte;
begin
  try
    wdCRC:=0;
    wdPtrCount:=0;
    while intBufLen<>0 do
    begin
      bytdat:=wdCRC shr 8;
      wdCRC:=wdCRC shl 8;

      wdCRC:=wdCRC xor constCRCTab[bytdat xor arayPtrBuf[wdPtrCount]];
      wdPtrCount:=wdPtrCount+1;
      intBufLen:=intBufLen-1;
    end;

    arayPtrBuff[wdPtrCount]:=wdCRC shr 8;
    mbytCRCHighData:=arayPtrBuf[wdPtrCount];
    wdPtrCount:=wdPtrCount+1;
    arayPtrBuff[wdPtrCount]:=wdCRC and $FF;
    mbytCRCLowData:=arayPtrBuf[wdPtrCount];
  end;

```

```

except
  on ex:Exception do
    begin
      MessageDlg(ex.Message+'(PackCRC)',mtError,[mbOK],0);
    end;
  end;
end;

//-----

```

higher 8 bit and lower 8 bit, and distribute to mbytCRCHighData and mbytCRCLowData

Parameter 1: ArayPtrBuf is from data length of data package (not including 0xAA,0xAA)

e.g. Data package is (170, 170, 13, 1, 250, 255, 254, 0, 2, 1, 2, 1, 1, 0, 0), 170 means Hex 0xAA, so

parameter 1 ArayPtrBuf is (13, 1, 250, 255, 254, 0, 2, 1, 2, 1, 1, 0, 0) (not including 0xAA,0xAA)

parameter 2: intBufLen deduct 2 byte from this data package(because CRC take 2 byte)

eg. Length of above data package is 13 byte, so intBufLen =13-2=11

Finally, it can get CRC verification code from above two parameters by function PackCRC.

5.3. Use CheckCRC function for CRC verification when receiving data package,

CheckCRC function

Parameter 1: arayPtrBuf also not include two oxAA data package

eg. Data package (170, 170, 11, 1, 241, 255, 254, 0, 51, 1, 59, 88, 44),

so arayPtrBuf is (11, 1, 241, 255, 254, 0, 51, 1, 59, 88, 44), exclude 170

parameter 2: intBufLen deduct 2 byte from this data package(because CRC take 2 byte)

eg. Length of above data package is 11 byte , so intBufLen =11-2=9

finally, it can be verified from above two parameters that input following PackCRC function.

```

//-----
function CheckCRC(arayPtrBuf:array of byte;intBufLen:integer):boolean;
var
  wdCRC:word;           // dual type variable
  bytDat:byte;
  bytPtrCount:byte;
  bInIsRight:boolean;   // true or false
begin
  wdCRC:=0;
  bytPtrCount:=0;

  try

```

```

while intBufLen<>0 do
begin
    bytDat:=wdCRC shr 8;
    wdCRC:=wdCRC shl 8;

    wdCRC:=wdCRC xor constCRCTab[bytDat xor arayPtrBuf[bytPtrCount]];
    bytPtrCount:=bytPtrCount+1;
    intBufLen:=intBufLen-1;
end;

if(arayPtrBuf[bytPtrCount]=(wdCRC shr 8)) and (arayPtrBuf[bytPtrCount+1]=wdCRC and $ff)
then
begin
    bInIsRight:=true;
end
else
begin
    bInIsRight:=false;
end;

except
on ex:Exception do
begin
    bInIsRight:=false;
    MessageDlg(ex.Message+'(CheckCRC)',mtError,[mbOK],0);
end;
end;
Result:=bInIsRight;
end;

//-----

```

5.4 Command on search on-line device (example)

Subnet ID of targeted device: subnet ID from 0 to 255

Device ID of targeted device: device ID255, broadcast

Operation code: 0x000E

Example 1: assigned subnet ID :1 search on-line device, device ID:FF

AAAA 0B 01 FA FF FE 00 0E 01 255 0D 3F

Totally 14 byte if send data package via Ethernet.

{take local IP for example, eg.IP : 192.168.18.5, so mbytIPPart1=192}
 maraySendUDPBuf[0]:=mbytIPPart1;

```

maraySendUDPBuf[1]:=mbytlPPart2; ( instead of the second local IP, here is 168)
maraySendUDPBuf[2]:=mbytlPPart3; ( instead of the third local IP, here is 18)
maraySendUDPBuf[3]:=mbytlPPart4; ( instead of the fourth local IP, here is 5)
maraySendUDPBuf[4]:=$48;{H}
maraySendUDPBuf[5]:=$44;{D}
maraySendUDPBuf[6]:=$4C;{L}
maraySendUDPBuf[7]:=$4D;{M}
maraySendUDPBuf[8]:=$49;{I}
maraySendUDPBuf[9]:=$52;{R}
maraySendUDPBuf[10]:=$41;{A}
maraySendUDPBuf[11]:=$43;{C}
maraySendUDPBuf[12]:=$4C;{L}
maraySendUDPBuf[13]:=$45;{E}

```

Example 2 broadcast ID :FF search online, device ID:FF

AAAA 0B 01 FA FF FE 00 0E 01 255 3D F1

6. Definition of G4 Device Type

Device Type ID	Model #	DESC
602	SB-DIM2c6A-DN	Dimmer 2ch 6A
601	SB-DIM4c3A-DN	Dimmer 4ch 3A
600	SB-DIM6c2A-DN	Dimmer 6ch 2A
434	SB-RLY4c20A-DN	Relay 4ch 20A
428	SB-RLY8c16A-DN	Relay 8ch 16A
149	SB-DDP	Dynamic Display Panel
281	SB-6BS	6 Buttons
282	SB-4BS	4 Buttons
278	SB-3BS	3 Buttons
1108	SB-Logic2-DN	Logic Module
3049	SB-SEC250K-DN	Security Module
309	SB-9in1T-CL	9 in 1 sensor
314	SB-5in1TL-CL	5 in 1 sensor
313	SB-6in1TL-CL	6 in 1 sensor
118	SB-4Z-UN	4 Zone Dry Contact
306	SB-IR-UN	IR Emitter
1201	SB-RSIP-DN	RS232IP Module
112	SB-DN-HVAC	HVAC Module
902	SB-ZAudio2-DN	Zone Audio