DreamScreen V2 WiFi UDP Protocol Rev *5*

The DreamScreen-WiFi platform uses UDP unicasting and broadcasting over the WLAN to provide communication. Port 8888 is used for both sending and receiving. The DreamScreen protocol is in a message-based, binary format, which is fast to send and parse. Please note, other features not within the scope of this document should be left delegated to the official DreamScreenTV iOS and Android apps.

Message Structure

	Start of Packet	Packet Length	Group Address	Flags	Cmd Upper	Cmd Lower	Payload	CRC
Size (Bytes)	1	1	1	1	1	1	variable	1

Start of Packet - Used to provide synchronization when parsing packets. Always 0xFC

Packet Length - Packet Length from Group Address (inclusive) to CRC (inclusive)

Group Address - The group number to which the device belongs. 0x00 indicates ‘No specified Group’, 0x01 indicates group 1, 0x02 indicates group 2, etc. If the Group Address is incorrect, DreamScreen will discard the message.

Flags - Provides context for handling the message. 0x11= write group, 0x21=write individual, 0xFF = Device discovery, 0x1E=request data?, 0x3C=?.

Command Upper - specifies command namespace.
Command Lower - specifies individual command within namespace.
Payload - variable length, depending upon the context of the command.
CRC - 8 bit CRC for error detection. If incorrect, DreamScreen will discard the message.

Example, Set Mode 0:
FC:06:01:21:03:01:00:C6. Command is 03:01, with payload 0x00, packet length 0x06

Commands
(updated 12/4/2019)

Description	**Cmd Upper**	**Cmd Lower**	**Payload Description**	**Payload Length (Bytes)**
Get Serial	0x01	0x03	Device serial number	<todo>
Reset ESP	0x01	0x05	Reset ESP (Bad idea!)	<todo>
Name	0x01	0x07	Device friendly name	16 (UTF8)
Group Name	0x01	0x08	Group friendly name	16 (UTF8)
Group Number	0x01	0x09	Group number 0 unassigned, 1-254 valid group numbers	1
Subscribe to Sector Data	0x01	0x0C	1 - Request to subscribe(Read only)	1
Stop ESP Drivers	0x01	0x11		
Unknown	0x01	0x13	Not sure what this is, but it happens a lot...	
Read bootloader mode	0x01	0x15	Need to investigate flags, payload	
Read Connect Version?	0x02	0x01		
Read PCI (PIC?) Version	0x02	0x02		
Read Diagnostic	0x02	0x03		
Mode	0x03	0x01	0 - Sleep 1 - Video 2 - Music 3 - Ambient	1
Brightness	0x03	0x02	0-100, indicating percentage	1
Zones	0x03	0x03	12-byte array indicating sector on/off. If on, byte for that sector is set to it's hex value. To enable zone 3, byte 3 is set to 0x03, etc.	12

Zones Brightness	0x03	0x04	12-byte array with brightness value from 0-100 per sector	12
Ambient Color	0x03	0x05	Color as 3-byte RGB; Red:Green:Blue	3
Saturation	0x03	0x06	Color as 3-byte RGB; Red:Green:Blue	3
Ambient Mode Type	0x03	0x08	0x00 - RGB Color 0x01 - Scene	1
Music Mode Type	0x03	0x09	0x00 - Audio Jack 0x01 - HDMI Input	1
Music Mode Colors	0x03	0x0A		
Music Mode Weights	0x03	0x0B		
Minimum Luminosit y	0x03	0x0C	Luminosity value, 0-100?	1
Ambient Scene	0x03	0x0D	0x00 - Random Color 0x01 - Fireside 0x02 - Twinkle 0x03 - Ocean 0x04 - Rainbow 0x05 - July 4th 0x06 - Holiday 0x07 - Pop 0x08 - Enchanted Forest	1
Fade Rate	0x03	0x0E	Not sure the max, but probably 0-100	1
Indicator Light Auto Off	0x03	0x13	0x00 - off 0x01 - on	1
USB Power enable	0x03	0x14	0x00 - off 0x01 on	1
Color Data	0x03	0x16	Readonly 12x3 array of color data, 3-byte RGB for 12 sectors	36
Sector Assignment	0x03	0x17		
Sector Broadcast	0x03	0x18		

Control				
Sector Broadcast Timing	0x03	0x19		
HDMI Input	0x03	0x20	0x00 - Channel 1 0x01 - Channel 2 0x02 - Channel 3	1
Music Mode Source	0x03	0x21	I think this is for the connect, same as the other one, but with flag 0x03 being for mic?	1
HDMI Input Name 1	0x03	0x23	HDMI 1 friendly name	16 (UTF8)
HDMI Input Name 2	0x03	0x24	HDMI 2 friendly name	16 (UTF8)
HDMI Input Name 3	0x03	0x25	HDMI 3 friendly name	16 (UTF8)
CEC Passthrough enabled	0x03	0x26	Enable (1) or disable(0)	1
CEC Switching enable	0x03	0x27	Enable (1) or disable(0)	1
HDP Enabled	0x03	0x28	Enable (1) or disable(0)	1
Video Frame Delay	0x03	0x2A	0-100?	1
Letterboxing enable	0x03	0x2B	Enable (1) or disable(0)	1
HDMI Active Channels	0x03	0x2C	(read only) Bits 2 to 0 Indicate HDMI Inputs 3 to 1 (0 - HDMI Not Valid. 1 - HDMI Valid)	1
Color Boost	0x03	0x2D	Enable (1) or disable(0)	1
CEC Power Enable	0x03	0x2E	Enable (1) or disable(0)	1

Pillarboxing Enable	0x03	0x2F	Enable (1) or disable(0)	1
SKU Setup	0x03	0x40	Enable (1) or disable(0)?	1
Flex Setup	0x03	0x41	Enable (1) or disable(0)?	1
HDR Tone Remapping	0x03	0x60	Enable (1) or disable (0)	1
Botloader setup	0x04	0x01	Don’t mess with this	
Reset PIC	0x04	0x02		
Factory reset DS	0x04	0x03		
ESP Connected to wifi	0x04	0x0D		
Other connected to wifi	0x04	0x14		
Display Animation	0x05	0x01		
Ambient Light auto adjust	0x05	0x02		
Microphone audio broadcast enable	0x05	0x03		
IR Enable	0x05	0x10		
Set IR Learning mode	0x05	0x11		

Set IR Manifest Entry	0x05	0x13		
Set email address	0x05	0x20		
Set thing name	0x05	0x21		

Notes (updated 12/4/2019):

Beware, some of the above commands are for internal use only. You may risk bricking your device, especially if you incorrectly use the ones related to firmwares and resets. You've been warned.

While in Ambient Mode, the Ambient Mode Type determines whether the Ambient Color or the Ambient Scene gets displayed.

Setting the HDMI Input will not initiate the switch if an HDMI source is not available on that particular port.

Android Sample Code

Helper for sending UDP packets

```
 /**
*  Builds a DreamScreen message and sends it over UDP
*  @param commandUpper specifies command namespace
*  @param commandLower specifies individal command within namespace
*  @param payload payload of the message, length depending upon the context of the
   Command
*  @param broadcastingToGroup true if message should be UDP broadcasted, false if
   message to be UDP unicasted
 */
void sendUDPWrite(byte commandUpper, byte commandLower, byte[] payload, boolean
 broadcastingToGroup) {
    ByteArrayOutputStream response = new ByteArrayOutputStream();
    response.write(0xFC);
response.write((byte) (0x05 + payload.length));
    response.write((byte) groupNumber); //typically 0, 1, or 2

    if (broadcastingToGroup) response.write(0b00100001); //reserved
    else response.write(0b00010001); //reserved

response.write(commandUpper);
    response.write(commandLower);

for (byte b : payload) response.write(b);

    byte crc = uartComm_calculate_crc8(response.toByteArray());
    response.write(crc);

if (broadcastingToGroup) sendUDPBroadcast(response.toByteArray());
    else sendUDPUnicast(response.toByteArray());
}
```

Initialize IP address of DreamScreen. Recommended to reserve this IP address on the router. Put this in onCreate or a constructor, referenced as global variables

```
//initialize ip
 addresses try {
    InetAddress lightsUnicastIP = InetAddress.getByName("192.168.1.100");
 //reserved IP address of DreamScreen/SideKick
    InetAddress broadcastIP = InetAddress.getByName("255.255.255.255");
 //default, works for many routers
} catch (UnknownHostException e) {}
```

Used to generate the 8-bit CRC

```
private static final byte[] uartComm_crc8_table = new byte[]{
        0x00, 0x07, 0x0E, 0x09, 0x1C, 0x1B, 0x12, 0x15, 0x38, 0x3F, 0x36, 0x31, 0x24,
0x23, 0x2A, 0x2D, 0x70, 0x77, 0x7E, 0x79, 0x6C, 0x6B, 0x62, 0x65, 0x48, 0x4F, 0x46,
0x41, 0x54, 0x53, 0x5A, 0x5D, (byte) 0xE0, (byte) 0xE7, (byte) 0xEE, (byte) 0xE9,
(byte) 0xFC, (byte) 0xFB, (byte) 0xF2, (byte) 0xF5, (byte) 0xD8, (byte) 0xDF, (byte)
0xD6, (byte) 0xD1, (byte) 0xC4, (byte) 0xC3, (byte) 0xCA, (byte) 0xCD, (byte) 0x90,
(byte) 0x97, (byte) 0x9E, (byte) 0x99, (byte) 0x8C, (byte) 0x8B, (byte) 0x82, (byte)
0x85, (byte) 0xA8, (byte) 0xAF, (byte) 0xA6, (byte) 0xA1, (byte) 0xB4, (byte) 0xB3,
(byte) 0xBA, (byte) 0xBD, (byte) 0xC7, (byte) 0xC0, (byte) 0xC9, (byte) 0xCE, (byte)
0xDB, (byte) 0xDC, (byte) 0xD5, (byte) 0xD2, (byte) 0xFF, (byte) 0xF8, (byte) 0xF1,
(byte) 0xF6, (byte) 0xE3, (byte) 0xE4, (byte) 0xED, (byte) 0xEA, (byte) 0xB7, (byte)
0xB0, (byte) 0xB9, (byte) 0xBE, (byte) 0xAB, (byte) 0xAC, (byte) 0xA5, (byte) 0xA2,
(byte) 0x8F, (byte) 0x88, (byte) 0x81, (byte) 0x86, (byte) 0x93, (byte) 0x94, (byte)
0x9D, (byte) 0x9A, 0x27, 0x20, 0x29, 0x2E, 0x3B, 0x3C, 0x35, 0x32, 0x1F, 0x18, 0x11,
0x16, 0x03, 0x04, 0x0D, 0x0A, 0x57, 0x50, 0x59, 0x5E, 0x4B, 0x4C, 0x45, 0x42, 0x6F,
0x68, 0x61, 0x66, 0x73, 0x74, 0x7D, 0x7A, (byte) 0x89, (byte) 0x8E, (byte) 0x87,
(byte) 0x80, (byte) 0x95, (byte) 0x92, (byte) 0x9B, (byte) 0x9C, (byte) 0xB1, (byte)
0xB6, (byte) 0xBF, (byte) 0xB8, (byte) 0xAD, (byte) 0xAA, (byte) 0xA3, (byte) 0xA4,
(byte) 0xF9, (byte) 0xFE, (byte) 0xF7, (byte) 0xF0, (byte) 0xE5, (byte) 0xE2, (byte)
0xEB, (byte) 0xEC, (byte) 0xC1, (byte) 0xC6, (byte) 0xCF, (byte) 0xC8, (byte) 0xDD,
(byte) 0xDA, (byte) 0xD3, (byte) 0xD4, 0x69, 0x6E, 0x67, 0x60, 0x75, 0x72, 0x7B,
0x7C, 0x51, 0x56, 0x5F, 0x58, 0x4D, 0x4A, 0x43, 0x44, 0x19, 0x1E, 0x17, 0x10, 0x05,
0x02, 0x0B, 0x0C, 0x21, 0x26, 0x2F, 0x28, 0x3D, 0x3A, 0x33, 0x34, 0x4E, 0x49, 0x40,
0x47, 0x52, 0x55, 0x5C, 0x5B, 0x76, 0x71, 0x78, 0x7F, 0x6A, 0x6D, 0x64, 0x63, 0x3E,
0x39, 0x30, 0x37, 0x22, 0x25, 0x2C, 0x2B, 0x06, 0x01, 0x08, 0x0F, 0x1A, 0x1D, 0x14,
0x13, (byte) 0xAE, (byte) 0xA9, (byte) 0xA0, (byte) 0xA7, (byte) 0xB2, (byte) 0xB5,
(byte) 0xBC, (byte) 0xBB, (byte) 0x96, (byte) 0x91, (byte) 0x98, (byte) 0x9F, (byte)
0x8A, (byte) 0x8D, (byte) 0x84, (byte) 0x83, (byte) 0xDE, (byte) 0xD9, (byte) 0xD0,
(byte) 0xD7, (byte) 0xC2, (byte) 0xC5, (byte) 0xCC, (byte) 0xCB, (byte) 0xE6, (byte)
0xE1, (byte) 0xE8, (byte) 0xEF, (byte) 0xFA, (byte) 0xFD, (byte) 0xF4, (byte) 0xF3
};

// FC:05:00:10:03:01:(A3)
private byte uartComm_calculate_crc8(byte[] data) {
   byte size = (byte) (data[1] + 0x01);
   byte cntr = 0x00;
   byte crc = 0x00;

   while (cntr < size) {
       crc = uartComm_crc8_table[(byte) (crc ^ (data[cntr])) & 0xFF];
       cntr++;
   }
   return crc;
}
```

Helper AsyncTasks that perform the network connections, to offload from the main thread of the application. One is used for unicasting, the other used for broadcasting

```
 private class UDPUnicast extends AsyncTask<byte[], Void,
    Void> { @Override
    protected Void doInBackground(byte[]...
        bytes) { try {
            DatagramSocket s = new
            DatagramSocket(); byte[] command =
            bytes[0];
            DatagramPacket p = new DatagramPacket(command,
 command.length, lightsUnicastIP, dreamScreenPort);
s.send(p);
s.close();
} catch (Exception e) {
}
return null;
}
}

 private class UDPBroadcast extends AsyncTask<byte[], Void,
    Void> { @Override
    protected Void doInBackground(byte[]...
        bytes) { try {
            DatagramSocket s = new
            DatagramSocket();
            s.setBroadcast(true);
byte[] command = bytes[0];
            DatagramPacket p = new DatagramPacket(command,
 command.length, broadcastIP, dreamScreenPort);
s.send(p);
s.close();
} catch (Exception e) {
}
return null;
}
}
```

Use it:

```
byte[] payload = new byte[]{(byte) 0x01}; //set to Video Mode
 sendUDPWrite((byte) 0x03, (byte) 0x01, payload, broadcastingToGroup);
```

```
byte[] payload = {(byte) 0xDD, (byte) 0x00, (byte) 0xFF};   //set ambient color
 purple
sendUDPWrite((byte) 0x03, (byte) 0x05, payload, broadcastingToGroup);
```

Appended 2/3/2017

Device Discovery and State

The recommended way to perform discovery is by sending a special ‘read current state’ UDP broadcast, which will cause every DreamScreen and SideKick to respond. You can then map each device to the IP where the response message originated.

Description	**Read/ Write**	**Command Upper**	**Command Lower**	**Payload Description**	**Payload Length (Bytes)**
Current State	*R*	0x01	0x0A	Dump of all attributes	Variable

‘Read current state’ message: **FC:05:FF:30:01:0A:2A**

‘Read current state’ response: The payload received varies by what the device is and the version of firmware. However, the last index of the payload always contains the productId of the device, which you should use to determine the proper context of the payload. When parsing, always ensure indexes are valid for the size of payload received for full compatibility with all past and future firmware versions.

Product ID	*Device*
0x01	DreamScreen HD
0x02	DreamScreen 4K
0x03	SideKick
0x04	Connect
0x07	DreamScreen Solo

DreamScreen HD and 4K Current State Payload

Index	**Size (Bytes)**	**Attribute**
0-15	16 (UTF8)	*Name*
16-31	16 (UTF8)	*Group Name*
32	1	*Group Number*
33	1	*Mode*
34	1	*Brightness*
35	1	*Zones*
36-39	3	*Zones Brightness*
40-42	3	*Ambient Color*
43-45	3	*Saturation*
46-51	6	*Flex Setup*
52	1	*Music Mode Type*
53-55	3	*Music Mode Colors*
56-58	3	*Music Mode Weights*
59-61	3	*Minimum Luminosity*
62	1	*Ambient Scene*
63	1	*Fade Rate*
69	1	*Indicator Light Auto Off*
70	1	*USB Power Enabled*
71	1	*Sector Broadcast Control*
72	1	*Sector Broadcast Timing*
73	1	*HDMI Input*
74	1	*Music Mode Source*
75-90	16 (UTF8)	*HDMI Input Name 1*
91-106	16 (UTF8)	*HDMI Input Name 2*
107-122	16 (UTF8)	*HDMI Input Name 3*
123	1	*CEC Passthrough*
124	1	*CEC Switching Enabled*

125	1	*HDP Enabled*
127	1	*Video Frame Delay*
128	1	*Letterboxing Enabled*
129	1	*HDMI Active Channels*
130	1	*ESP Firmware Version*
131	1	*ESP Firmware Version*
132	1	*PIC Version Number*
133	1	*PIC Version Number*
134	1	*ColorBoost*
135	1	*CEC POWER ENABLE(optional?)*
136	1	*SKU SETUP(optional)*
137	1	*BOOTSTATE(optional)*
138	1	*Pillarboxing enable(optional)*
139	1	*HDMI Tone Remapping(optional)*

SideKick Current State Payload

Index	**Size (Bytes)**	**Attribute**
0-15	16 (UTF8)	*Name*
16-31	16 (UTF8)	*Group Name*
32	1	*Group Number*
33	1	*Mode*
34	1	*Brightness*
35-37	3	*Ambient Color*
38-40	3	*Saturation*
42-56	15	*Sector Data*
57-58	2	*FW Version (major, minor)*
59	1	*Ambient Mode Type (opt)*
60	1	*Ambient Scene (opt)*

Connect Current State Payload

Index	**Size (Bytes)**	**Attribute**
0-15	16 (UTF8)	*Name*
16-31	16 (UTF8)	*Group Name*
32	1	*Group Number*
33	1	*Mode*
34	1	*Brightness*
35-37	3	*Ambient Color*
38-40	3	*Saturation...not shown, but it's here*
41	1	*Fade Rate*
42-56	15	*Sector Data*
57-58	2	*FW Version (major, minor)*
59	1	*Ambient Mode Type (opt)*
60	1	*Ambient Scene (opt)*
61	1	*HDMI Input*
62	1	*display anim enabled*
63	1	*Ambient Light Auto Adjust*
64	1	*Microphone Audio broadcast enabled?*
65	1	*IR Enabled*
66	1	*IR Learning Mode*
67-106	40	*IR Register. 8x Possible saved actions, with 5x bytes each. Byte 1 is the action ID, the other 4 are the IR code.*
115-177	63	*Thing name (AWS Lambda URL?)*

Appended 9/10/2018

Subscribing to Sector Data

The RGB data that DreamScreen displays to the LEDs behind the TV is averaged into 12 different sectors, and then sent out to all subscribed clients (SideKicks) within the group. To make your own client, just keep an active subscription alive and DreamScreen will be handing over the 36 bytes of RGB data. You can then perform your own logic on what to do with it.

How the subscription works is that DreamScreen sends out a 0x010C 'subscription request' read broadcast over the network to all members of the group, at a 5 second interval. Unicast back to the command with a payload of 0x01. DreamScreen will then start streaming the sector data. The subscription will timeout after 3 missed 'subscription requests', so make sure to keep it alive. Sector data will come as 0x0316 with a 36 byte payload, formatted as sectors 1 to 12 each being 24-bit rgb. If the streaming never begins, make sure your client is in the same group as DreamScreen.

7	6	5	4	3
8				2
9	10	11	12	1