

DreamScreen V2 WiFi UDP Protocol Rev 2

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	Command Upper	Command Lower	Payload	CRC
Size (Bytes)	1	1	1	1	1	1	variable	1

Start of Packet - is 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 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

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

Description	Command Upper	Command Lower	Payload Description	Payload Length (Bytes)
<i>Mode</i>	0x03	0x01	0 - Sleep 1 - Video 2 - Music 3 - Ambient	1
<i>Brightness</i>	0x03	0x02	0-100, indicating percentage	1
<i>Ambient Color</i>	0x03	0x05	Color as 3-byte RGB; Red:Green:Blue	3
<i>Ambient Mode Type</i>	0x03	0x08	0x00 - RGB Color 0x01 - Scene	1
<i>Ambient Scene</i>	0x03	0x0D	0x00 - Random Color 0x01 - Fireside 0x02 - Twinkle 0x03 - Ocean 0x04 - Rainbow 0x05 - July 4th 0x06 - Holiday 0x07 - Pop 0x08 - Enchanted Forest	1
<i>HDMI Input</i>	0x03	0x20	0x00 - Channel 1 0x01 - Channel 2 0x02 - Channel 3	1

Notes:

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 individual 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);
```

iOS / Swift Sample Code

The iOS app uses the CocoaAsyncSocket library for sending UDP. See <https://github.com/robbiehanson/CocoaAsyncSocket>

You can install the library via CocoaPods:
pod 'CocoaAsyncSocket'

And then import into your project

```
import CocoaAsyncSocket
```

Helper for sending UDP packets. Same structure as the Android version

```
func sendUDPWrite(command1:UInt8, command2:UInt8, payload:[UInt8], broadcastingToGroup:Bool){

    var response : [UInt8] = [0xFC, 0x05 + UInt8(payload.count), UInt8(groupNumber)]

    if(broadcastingToGroup){
        response.append(0b00100001) //reserved
    } else {
        response.append(0b00010001) //reserved
    }

    response.append(command1)
    response.append(command2)

    response += payload
    let crc:UInt8 = uartComm_calculate_crc8(data: response)
    response.append(crc)

    let data : NSData = NSData(bytes: response, length: response.count)

    if broadcastingToGroup {
        socket.send(data as Data, toHost: broadcastIpString, port: dreamScreenPort, withTimeout: -1, tag: 22)
    }
    else {
        socket.send(data as Data, toHost: lightsUnicastIP, port: dreamScreenPort, withTimeout: -1, tag: 11)
    }

}
```

Initialize the IP addresses as Strings. Initialize the UDP Socket.

```
let broadcastIpString : String = "255.255.255.255"
let lightsUnicastIP : String = "192.168.1.100"

var socket : GCDAsyncUdpSocket! = GCDAsyncUdpSocket(delegate: nil, delegateQueue: DispatchQueue.main)
do {
    try socket.enableBroadcast(true)
} catch {}
```

Used to generate the 8-bit CRC

```
let uartComm_crc8_table : [UInt8] = [
    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,
    0xE0, 0xE7, 0xEE, 0xE9, 0xFC, 0xFB, 0xF2, 0xF5, 0xD8, 0xDF, 0xD6, 0xD1, 0xC4, 0xC3, 0xCA, 0xCD,
    0x90, 0x97, 0x9E, 0x99, 0x8C, 0x8B, 0x82, 0x85, 0xA8, 0xAF, 0xA6, 0xA1, 0xB4, 0xB3, 0xBA, 0xBD,
    0xC7, 0xC0, 0xC9, 0xCE, 0xDB, 0xDC, 0xD5, 0xD2, 0xFF, 0xF8, 0xF1, 0xF6, 0xE3, 0xE4, 0xED, 0xEA,
    0xB7, 0xB0, 0xB9, 0xBE, 0xAB, 0xAC, 0xA5, 0xA2, 0x8F, 0x88, 0x81, 0x86, 0x93, 0x94, 0x9D, 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,
    0x89, 0x8E, 0x87, 0x80, 0x95, 0x92, 0x9B, 0x9C, 0xB1, 0xB6, 0xBF, 0xB8, 0xAD, 0xAA, 0xA3, 0xA4,
    0xF9, 0xFE, 0xF7, 0xF0, 0xE5, 0xE2, 0xEB, 0xEC, 0xC1, 0xC6, 0xCF, 0xC8, 0xDD, 0xDA, 0xD3, 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,
    0xAE, 0xA9, 0xA0, 0xA7, 0xB2, 0xB5, 0xBC, 0xBB, 0x96, 0x91, 0x98, 0x9F, 0x8A, 0x8D, 0x84, 0x83,
    0xDE, 0xD9, 0xD0, 0xD7, 0xC2, 0xC5, 0xCC, 0xCB, 0xE6, 0xE1, 0xE8, 0xEF, 0xFA, 0xFD, 0xF4, 0xF3
]

func uartComm_calculate_crc8(data : [UInt8]) -> UInt8 {
    let size:UInt8 = (data[1] + 0x01)
    var cntr:UInt8 = 0x00
    var crc:UInt8 = 0x00

    while (cntr < size) {
        crc = uartComm_crc8_table[ Int(crc ^ (data[Int(cntr)]))]
        cntr += 1
    }
    return crc;
}
```

Use it:

```
let payload : [UInt8] = [0x01] //set to video mode
self.sendUDPWrite(command1: 0x03 , command2: 0x01, payload: payload, broadcastingToGroup:
broadcastingToGroup)
```

```
let payload : [UInt8] = [0xDD, 0x00, 0xFF] //set ambient color purple
self.sendUDPWrite(command1: 0x03 , command2: 0x05, payload: payload, broadcastingToGroup:
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)
<i>Current State</i>	<i>R</i>	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. Currently DreamScreen can send up to 140 bytes; SideKick can send up to 62.

<i>Product ID</i>	<i>Device</i>
0x01	DreamScreen HD
0x02	DreamScreen 4K
0x03	SideKick

DreamScreen HD and 4K Current State Payload

Index	Attribute
[0,16)	<i>Name (UTF8)</i>
[16,32)	<i>Group Name (UTF8)</i>
32	<i>Group Number</i>
33	<i>Mode</i>
34	<i>Brightness</i>
[40,43)	<i>Ambient Color</i>
62	<i>Ambient Scene</i>
73	<i>HDMI Input</i>
[75,91)	<i>HDMI Input Name 1 (UTF8)</i>
[91,107)	<i>HDMI Input Name 2 (UTF8)</i>
[107,123)	<i>HDMI Input Name 3 (UTF8)</i>
129	<i>HDMI Active Channels</i>

SideKick Current State Payload

Index	Attribute
[0,16)	<i>Name (UTF8)</i>
[16,32)	<i>Group Name (UTF8)</i>
32	<i>Group Number</i>
33	<i>Mode</i>
34	<i>Brightness</i>
[35,38)	<i>Ambient Color</i>
60	<i>Ambient Scene</i>

Description	Command Upper	Command Lower	Payload Description	Payload Length (Bytes)
<i>Name</i>	0x01	0x07	Device friendly name	16 (UTF8)
<i>Group Name</i>	0x01	0x08	Group friendly name	16 (UTF8)
<i>Group Number</i>	0x01	0x09	Group number	1
<i>HDMI Input Name 1</i>	0x03	0x23	HDMI 1 friendly name	16 (UTF8)
<i>HDMI Input Name 2</i>	0x03	0x24	HDMI 2 friendly name	16 (UTF8)
<i>HDMI Input Name 3</i>	0x03	0x25	HDMI 3 friendly name	16 (UTF8)
<i>HDMI Active Channels</i>	0x03	0x2C (read only)	Bits 2 to 0 Indicate HDMI Inputs 3 to 1 (0 - HDMI Not Valid. 1 - HDMI Valid)	1

The other Settings can be documented if wished; email info@dreamscreentv.com.