Technical Documentation

Kuando IoT Busylight Omega – LoRaWAN

Version 4.2.2 -– 10th September 2025
Valid for HW1.2 & 1.5

1 Contents

1. Composing the LoRa color payload for kuando Busylight.

Before the Busylight can change color, an application controlling the logic needs to be integrated/developed. Please see documentation on LoRaWAN network server to learn how to connect an application.

1.1 Color

Payload to compose a color for the Lora Busylight is a 5-byte array.
Byte 0-2 controls color and intensity

- Byte 0: Red Color intensity (0..255)
- Byte 1: Blue Color intensity (0..255)
- Byte 2: Green Color intensity (0..255)

Byte 3-4 controls steady/flashing mode

- Byte 3: On duration (0..255) (value is 1/100 seconds)
- Byte 4: Off duration (0..255) (value is 1/100 seconds). When set to 0 the mode is always steady light.

Example for a blue static light with full intensity:
Byte[0]=0
Byte[1]=255
Byte[2]=0
Byte[3]=255
Byte[4]=0

Depending on the network provider, this byte array needs to be encoded in base64 or something similar. The above byte array would result in this base64 string: AP8A/wA=. decoded string 00FF00FF00, a solid blue color with 100% brightness.

Other examples:

Solid Red – 99/255 intensity	**990000FF00**
Solid Green – 99/255 intensity	**000099FF00**
Solid Yellow – full intensity	**FF00FF6400**
Solid Purple – Full intensity	**FFFF006400**
Blue Flashing (1sec on/off) – full intensity	**00FF006464**

Warm white color is set if the same value is used for RGB.
Eg. FFFFFFFF00, will give a bright white warm color.

1.2 Automatic Uplink reply.

The device can be triggered to reply on a color downlink payload. If so, it will repond with standard keep alive uplink, including data about current state. See Chapter 2 for more info about uplink.

1.2.1 Reply triggered by downlink.

If you include an optional 6th byte [01] in the downlink payload, the device will automatically reply with a keep alive uplink.

Example for a blue static light with full intensity including an uplink reply:
Byte[0]=0 (00)
Byte[1]=255 (FF)
Byte[2]=0 (00)
Byte[3]=255 (FF)
Byte[4]=0 (00)
Byte[5]=01 (01)

Example in Hex

Solid Blue – full intensity with uplink reply	**00FF00FF0001**

1.2.2 Persistent uplink reply.

Alternatively, it is possible with a downlink command (0x06) to set the device always to send an uplink as a reply to the Downlink. Please see Chapter 6 for more info.

2. Decoding the Uplink Payload

An uplink, keep alive signal, is pr default sent every 30 minutes from the device. It is a 24- or 25-byte array structured as follows:

Byte	*Structure*	*Content*
0-3	Signed int 32 Bit	RSSI from device perspective (dBm)
4-7	Signed int 32 Bit	SNR from device perspective (dB)
8-11	Unsigned int 32 Bit	Downlinks received, since last join. Payload messages only.
12-15	Unsigned int 32 Bit	Uploads sent, since last join. Payload messages only.
16	Byte	Last color(Red)
17	Byte	Last color(Blue)
18	byte	Last color(Green)
19	Byte	Time On (1/10 seconds)
20	Byte	Time Off (1/10 seconds)
21	Byte	SW revision
22	Byte	HW revision
23	Byte	ADR state (1=on, 0=off)
24	Byte	High Brightness mode (1=on, 0=off)

The last byte “High Brightness mode” is only included if the device is set to this mode. High Brightness mode is available on HW 1.5 and above. Please see Chapter 4 for more information.

Example of a 25-byte decoded uplink payload:

c4ffffff2100000004000000030000009999 00ff00440f0101	2	4	6	8		
c4ffffff	c4	ff	ff	ff	-60	*RSSI from device perspective*
21000000	21	00	00	00	33	*SNR from device perspective*
04000000	04	00	00	00	4	*Downlinks received, since last join*
03000000	03	00	00	00	3	*Uploads sent, since last join*
999900	153	153	0		#990099	*Last color(R,B,G)*
ff	255				255	*Time On*
00	0				0	*Time Off*
44	44				68	*sw revision*
0f	0f				15	*HW revision*
01	01				1	*ADR state*
01	01				1	*High Brightness mode*

Example of a structure in a C type language:

```
[StructLayout(LayoutKind.Sequential, Pack = 1)]
struct busylightuplinkmessagestruct
```

```
{
    Int32 rssi;
    Int32 snr;
    UInt32 messages_received;
    UInt32 messages_send;
    byte lastcolor_red;
    byte lastcolor_blue;
    byte lastcolor_green;
    byte lastcolor_ontime;
    byte lastcolor_offtime;
    byte sw_rev;
    byte hw_rev;
    byte adr_state;
}
```

Please note: If you are using pure C, you need to adjust the data types according to your data type sizes, which depends on the compiler and processor properties. This specific example is C#, where the size of the members can be defined explicitly. The Structure layout annotation tells the compiler how to arrange the members to directly map the byte array to the struct.

Payload decoder from TTN device Repository can be found in Appendix1:

3. LoRaWan version and LoRa FPort

Busylight IoT is based on LoRaWAN MAC version 1.0.3 and uses FPort 15

4. High Brightness Mode

Busylight IoT with HW-version 1.5 or higher is per default configured for indoor use with smooth brightness colors. If the Busylight IoT is used in very bright and sunny areas a **High Brightness mode** can be set on the device.

- High brightness mode is set with the hex command 0701
- Default brightness mode is set with hex command 0700.

The setting is persistent and will not change if the device is rebooted.

5. Power Consumption

- Power Supply: SPD3303X
- Amp-meter: DVM345DI
- DUT: JSE #1 – device EUI: 2020204135260602
- Voltage setting: 5.000V
- Waiting for join (faint yellow): 9.4mA/0.05W
- Joined (green): 20.4mA/0.1W
- LED’s off consumption: 16.5mA/0.08W

Color vs. intensity consumption table:

Power% (Hex)		10% (26)		20% (51)		40% (102)		60% (153)		80% (204)		100% (255)	
Unit		*mA*	*Watts*	*mA*	*Watts*	*mA*	*Watts*	*mA*	*Watts*	*mA*	*Watts*	*mA*	*Watts*
Red		20.4	0.1	24.9	0.12	33.3	0.17	41.9	0.21	50.5	0.25	59.1	0.30
Green		20.3	0.1	24.5	0.12	32.5	0.16	40.5	0.20	48.5	0.24	56.6	0.28
Blue		20.0	0.1	24.2	0.12	32.9	0.16	39.7	0.20	47.5	0.24	55.3	0.28
Yellow		24.8	0.12	33.0	0.16	48.4	0.24	62.3	0.31	75.7	0.38	87.2	0.44
Cyan		23.9	0.12	31.3	0.16	45.2	0.23	58.5	0.29	70.9	0.35	81.5	0.41
Purple		24.0	0.12	30.3	0.15	43.4	0.22	57.1	0.29	71.4	0.36	85.3	0.43
White		27.9	0.14	37.4	0.19	56.3	0.28	75.0	0.38	90.1	0.45	113.2	0.57

6. Functional Commands

2-byte commands to control the Busylight can be sent directly to the device.
Commands can be sent to Busylight after the join is registered in the NS and the first uplink with payload received.

Command, byte1	Parameter byte2	Example HEX	Function	Persi-stent	FW support
0xAA	00	AA00	Restart and reset to factory settings.	-	2.6-
0x01	00	0100	Enable ADR (enabled as default)	no	2.6-
0x02	00	0200	Disable ADR	no	2.6-
0x03	Number	0306	Connection count. Number of missed uplink reply-counts before restart. Default is 6.	no	6.1-
0x04	Time in minutes	040F	Customized time between uplinks in minutes. default 30 minutes (example is 0F=15 minutes).	no	2.6-
0x05	01	0501	Request keep alive uplink from device. Will be sent once.	-	5.6-
0x06	00	0600	Disable auto uplink after downlink (default)	no	5.6-
0x06	01	0601	Enable auto uplink after downlink	no	5.6-
0x07	00	0700	Disable high brightness mode (default). Only HW1.5 and up	yes	5.6-
0x07	01	0701	Enable high brightness mode (only HW1.5 and up).	yes	5.6-
0x09	FF+5 byte	**09FF**353535FF00	Change default light after join. Default is dim green 000035FF00. Example is dim white 353535FF00. Payload FFFFFFFFFF will reset to default.	yes	6.7- (not released, ask for access)

7. Technical Commands

Command byte0	Parameter byte1-16	Example HEX	Function	FW support
0x09	01	**09**01	Ask for CPU temperature in uplink (only HW1.5 and up).	6.1-
0x09	02	**09**02	Ask for actual temperature throttle (only HW1.5 and up).	6.1-
0x09	03	**09**03	Ask for last reset cause (only HW1.5 and up). 0x80 BACKUP 0x40 SOFTWARE 0x20 WDT (Watchdog timer) 0x10 External Reset 0x04 Brownout 3.3V 0x02 Brownout 1.2V 0x01 Power on Reset	6.1-
0x09	04	**09**04	Ask for ADR status (only HW1.5 and up).	6.1-
0x09	05	**09**05	Ask for RSSI from device perspective	6.1-
0x09	06	**09**06	Ask for SNR from device perspective	6.1-
0xBB	00	**BB**00	FW upgrade mode. Busylight goes in Boot Mode on next power up. • Attach USB to windows PC – device will flash in light blue • Device opens as device “desklight” in file explorer • Copy firmware uf2 file to the device(supplied by Plenom) • Light will flash quickly, restart, go yellow and try to connect. • All done.	2.6-
0x10	8byte JoinEUI	**10**0102030 405060708	Change of JoinEUI*: 9byte (1+8) command Sets new AppEUI/JoinEUI in the device. Await write procedure. Device will flash pink, turn blue and respond with UL FF10 when done. Adjust LNS and repower device to rejoin	6.1-
0x11	16byte AppKey	**11**0102030 405060708 090a0b0c0 d0e0f10	Change of AppKey*: 17byte (1+16) command Sets new AppKey in the device. Await write procedure. Device will flash pink, turn blue and respond with UL FF11 when done. Adjust LNS and repower device to rejoin	6.1-

*** Important**: Please note that changing JoinEUI and AppKeys needs to be done very carefully. You need to keep track of your changes at all time. If IDs and keys are lost, you cannot connect and restore. Plenom is not liable for this process.

8. Contact us

If you have questions or need support, please contact us here:

Appendix 1 – Payload formatter TTN

```
function decodeUplink(input) {
 if (input.bytes.length == 24)
 {
 return {
  data: {
   RSSI: byteArrayToLong(input.bytes, 0),
   SNR: byteArrayToLong(input.bytes, 4),
   messages_received: byteArrayToLong(input.bytes, 8),
   messages_send: byteArrayToLong(input.bytes, 12),
   lastcolor_red: input.bytes[16],
   lastcolor_blue: input.bytes[17],
   lastcolor_green: input.bytes[18],
   lastcolor_ontime: input.bytes[19],
   lastcolor_offtime: input.bytes[20],
   sw_rev: input.bytes[21],
   hw_rev: input.bytes[22],
   adr_state: input.bytes[23]
  },
  warnings: [],
  errors: []
 };
 }
 else if (input.bytes.length == 25)
 {
 return {
  data: {
   RSSI: byteArrayToLong(input.bytes, 0),
   SNR: byteArrayToLong(input.bytes, 4),
   messages_received: byteArrayToLong(input.bytes, 8),
   messages_send: byteArrayToLong(input.bytes, 12),
   lastcolor_red: input.bytes[16],
   lastcolor_blue: input.bytes[17],
   lastcolor_green: input.bytes[18],
   lastcolor_ontime: input.bytes[19],
   lastcolor_offtime: input.bytes[20],
   sw_rev: input.bytes[21],
   hw_rev: input.bytes[22],
   adr_state: input.bytes[23],
   high_brightness_mode: input.bytes[24]
  },
  warnings: [],
  errors: []
 };
 }
 else if (input.bytes.length == 19)
 {
 return {
  data: {
   messages_received: byteArrayToLong(input.bytes, 0),
   messages_send: byteArrayToLong(input.bytes, 4),
   lastcolor_red: input.bytes[8],
   lastcolor_blue: input.bytes[9],
   lastcolor_green: input.bytes[10],
   lastcolor_ontime: input.bytes[11],
   lastcolor_offtime: input.bytes[1],
   sw_rev: input.bytes[13],
   hw_rev: input.bytes[14],
   sound_no: input.bytes[15],
```

```
    sound_volume: input.bytes[16],
    sound_duration:input.bytes[17],
    high_brightness_mode: input.bytes[18]
   },
   warnings: [],
   errors: []
  };
  }
 else if (input.bytes.length == 20)
  {
  return {
   data: {
    messages_received: byteArrayToLong(input.bytes, 0),
    messages_send: byteArrayToLong(input.bytes, 4),
    lastcolor_red: input.bytes[8],
    lastcolor_blue: input.bytes[9],
    lastcolor_green: input.bytes[10],
    lastcolor_ontime: input.bytes[11],
    lastcolor_offtime: input.bytes[1],
    sw_rev: input.bytes[13],
    hw_rev: input.bytes[14],
    sound_no: input.bytes[15],
    sound_volume: input.bytes[16],
    sound_duration:input.bytes[17],
    high_brightness_mode: input.bytes[18],
    controlbyte: input.bytes[19]
   },
   warnings: [],
   errors: []
  };
  }
  else if (input.bytes.length == 10)
  {
 return {
   data: {
    messages_send: byteArrayToLong(input.bytes, 0),
    lastcolor_red: input.bytes[4],
    lastcolor_blue: input.bytes[5],
    lastcolor_green: input.bytes[6],
    lastcolor_ontime: input.bytes[7],
    lastcolor_offtime: input.bytes[8],
    high_brightness_mode: input.bytes[9]
   },
   warnings: [],
   errors: []
  };
  }
  else if (input.bytes.length == 3)
  {
   if (input.bytes[0]==0x09)
   {
    switch (input.bytes[1])
    {
     case 1:
      return {
       data:
       {
        temperature: input.bytes[2],
       },
       warnings: [],
       errors: []
      };
```

```
      break;
    case 2:
      return {
        data:
        {
          temperature_throttle: input.bytes[2],
        },
        warnings: [],
        errors: []
      };
      break;
    case 3:
      return {
        data:
        {
          last_reset_cause: input.bytes[2],
        },
        warnings: [],
        errors: []
      };
      break;
    case 4:
      return {
        data:
        {
          adr_status: input.bytes[2],
        },
        warnings: [],
        errors: []
      };
      break;

  }
 }
}
else if (input.bytes.length == 6)
{
 if (input.bytes[0]==0x09)
 {
  switch (input.bytes[1])
  {
    case 5:
       return {
        data:
        {
          RSSI: byteArrayToLong(input.bytes, 2),
        },
        warnings: [],
        errors: []
      };
      break;
      case 6:
       return {
        data:
        {
          SNR: byteArrayToLong(input.bytes, 2),
        },
        warnings: [],
        errors: []
      };
      break;
  }
```

```
  }
}
else if (input.bytes.length == 8)
 {
  if (input.bytes[0]==0x09)
  {
    switch (input.bytes[1])
   {
    case 7:
       return  {
      data:
      {
       voltage: charArrayToFloat(input.bytes, 2),
       oncharger: input.bytes[5]-0x30,
       charging: input.bytes[6]-0x30,
       buttonstatus: input.bytes[7]-0x30
      },
      warnings: [],
      errors: []
     };
     break;
    }
  }
}
 else
 {
  return {data: {
    bytes: input.bytes,
   },
  warnings: [],
  errors: []
  }
 }
}


charArrayToFloat = function(/*byte[]*/charArray, /*int*/from) {
  let retval = ((charArray[from]-0x30)*100) + ((charArray[from+1]-0x30)*10) + (charArray[from+2]-0x30);
  return retval/100;
}

byteArrayToLong = function(/*byte[]*/byteArray, /*int*/from) {
  return byteArray[from] | (byteArray[from+1] << 8) | (byteArray[from+2] << 16) | (byteArray[from+3] << 24);
};

function encodeDownlink(input) {
 return {
  bytes:[(input.data.red & 0x00FF), (input.data.blue & 0x00FF), (input.data.green & 0x00FF), (input.data.ontime &
0x00FF),
  (input.data.offtime & 0x00FF)],
  fPort: 15,
  warnings: [],
  errors: []
 };
}

function decodeDownlink(input) {
if (input.bytes.length == 5)
 {
 return {
  data: {
   red: input.bytes[0],
```

```
    green: input.bytes[2],
    blue: input.bytes[1],
    ontime: input.bytes[3],
    offtime: input.bytes[4]
   },
   warnings: [],
   errors: []
  }
  }
 else if (input.bytes.length == 6)
  {
  return {
   data: {
    red: input.bytes[0],
    green: input.bytes[2],
    blue: input.bytes[1],
    ontime: input.bytes[3],
    offtime: input.bytes[4],
    immediate_uplink: input.byte[5]
   },
   warnings: [],
   errors: []
    }
  }
}
```