Light Lightness Model Application Note

V 0.0.1

2019/1/29

修订历史（Revision History）

日期	版本	修改	作者	Reviewer
2018/09/26	Draft v0.0.1	初稿	hector	

目 录

表目录

图目录

未找到图形项目表。

词汇表

缩写	含义

1 背景介绍

Light Lightness Model 主要是用来控制灯的亮度，Light Lightness Model 是一个组合态的 Model，包括灯的亮度，线性尺度上的亮度，最近亮度，默认亮度等状态。Light Lightness Model 是灯控相关 Model 中比较简单的一个，比较合用来做灯控相关 Model 入门的学习。

2 状态

Light Lightness 是一个组合态，包括灯的实际亮度，线性尺度上的亮度，最近亮度，默认亮度等状态。

2.1 Light Lightness Linear

线性亮度表示灯在线性尺度上的亮度，此状态和实际亮度绑定在一起，参数如表 2-1。此亮度等同于测量到的光强(Y)。

表 2-1 Light Lightness Linear 状态

值	描述
0x0000	灯没发光
0x0001-0xFFFE	灯发光的亮度
0xFFFF	灯发光的最高亮度

2.1.1 绑定 Light Lightness Actual 状态

灯光的线性亮度和实际亮度绑定在一起，当两者中一个改变了，另一个也随着改变，转换公式如下：

$$Light\ Lightness\ Actual = 65535\sqrt{\frac{Light\ \ Lightness\ \ Linear}{65535}}$$

2.2 Light Lightness Actual

实际亮度表示灯在视觉上的亮度，此状态与 Generic Level 和 Generic OnOff 状态绑定在一起，参数如表 2-2。

表 2-2 Light Lightness Actual 状态

值	描述
0x0000	灯没发光
0x0001-0xFFFE	视觉上灯的亮度
0xFFFF	视觉上灯的最高亮度

视觉光强(L)和测量的光强(Y)的转换公式如下：

$$L = 65535\sqrt{\frac{Y}{65535}}$$

2.2.1 绑定 Light Lightness Linear 状态

灯光的线性亮度和实际亮度绑定在一起，当两者中一个改变了，另一个也随着改变，转换公式如下：

$$Light\ Lightness\ Linear = \text{Ceiling}(65535((\frac{Light\ \ Lightness\ \ Actual}{65535})^2)$$

2.2.2 绑定 Generic Level 状态

当灯光实际亮度和 Generic Level 绑定在一起时，他们之间相互转换的公式如下：

Light Lightness Actual = *Generic Level* + 32768

Generic Level = *Light Lightness Actual* – 32768

2.2.3 绑定 Generic OnOff 状态

当灯光实际亮度和 Generic OnOff 状态绑定时，在 Generic OnOff 状态改变时，实际亮度状态要做如下变化：

Generic OnOff 变为 0 时：

Light Lightness Actual = 0

Generic OnOff 变成 1，并且 Light Lightness Default 状态是 0 时：

Light Lightness Actual = Light Lightness Last

Genric OnOff 变成 1，并且 Light Lightness Default 状态不为 0 时：

Light Lightness Actual = Light Lightness Default

同样反向的绑定也是存在的，当实际亮度改变试，Generic OnOff 状态也需要做如下改变：

Light Lightness Actual 的值为 0 时：

Generic OnOff = 0x00

Light Lightness Actual 的值大于 0 时：

Generic OnOff = 0x01

2.2.4 绑定 Generic OnPowerUp 状态

当灯光实际亮度和 Generic OnPowerUp 状态绑定时，在系统启动的时刻，灯光实际亮度需要做如下改变：

Generic OnPowerUp 的值为 0：

Light Lightness Actual = 0

Generic OnPowerUp 的值为 1，并且 Light Lightness Default 的值不为 0：

Light Lightness Actual = Light Lightness Default

Generic OnPowerUp 的值为 1，并且 Light Lightness Default 的值为 0：

Light Lightness Actual = Light Lightness Last

Generic OnPowerUp 的值为 2：

Light Lightness Acutal = last known value before powered down

2.2.5 绑定 Light Lightness Range 状态

当灯光实际亮度和 Light Lightness Range 状态绑定时，在灯光实际亮度改变时需要做如下计算：

Light Lightness Actual 的值不为 0，并且小于 Light Lightness Range Min 的值时：

Light Lightness Actual = Light Lightness Range Min

Light Lightness Actual 的值不为 0，并且大于 Light Lightness Range Max 的值时：

Light Lightness Actual = Light Lightness Range Max

2.3 Light Lightness Last

此状态表示最后一次实际亮度的非 0 值，可以用来在开关，系统启动时对灯的实际亮度做一些改变。此状态的默认值为 0xFFFF。参数如表 2-3。

表 2-3 Light Lightness Last 状态

值	描述
0x0000	灯没发光
0x0001-0xFFFE	视觉上灯的亮度
0xFFFF	视觉上灯的最高亮度

2.4 Light Lightness Default

此状态代表灯实际亮度的默认值，可以用来在系统启动时对灯的实际亮度做一些改变。此状态的默认值为 0x0000。参数如表 2-4。

表 2-4 Light Lightness Default 状态

值	描述
0x0000	灯没发光
0x0001-0xFFFE	视觉上灯的亮度
0xFFFF	视觉上灯的最高亮度

2.5 Light Lightness Range

此状态限定了灯光亮度的范围，当灯光实际亮度改变时，需要在这个范围之内。参数表 2-5。

表 2-5 Light Lightness Range 状态

值	描述
0x0000	无效值
0x0001-0xFFFF	灯的亮度

3 SIG 消息

3.1 Light Lightness Get

此消息用来获取灯光的实际亮度值，远端收到此消息时会回一个 Light Lightness Status 消息。此消息没有参数。

3.2 Light Lightness Set

此消息用来设置灯光的实际亮度值，远端收到此消息时应当调节灯光的实际亮度，并返回 Light Lightness Status 消息来表明当前灯光亮度。参数如表 3-1 所示。

表 3-1 Light Lightness Set 消息

域	大小(字节)	注释
Lightness	2	目标灯光实际亮度值
TID	1	事务 ID，用来标记是新的消息还是重传的消息
Transition Time	1	Transition 的时间(可选)。此字段不存在时，灯光亮度应该立刻改变。存在时，灯光亮度应该在 Transition 的时间之内完成
Delay	1	消息执行的延时，5ms 步进(可选)，当 Transition 存在时，也会存在

3.3 Light Lightness Set Unacknowledged

此消息用来设置灯光的实际亮度值，远端收到此消息时应当调节灯光的实际亮度，此消息不需要回复。参数同表 3-1。

3.4 Light Lightness Status

此消息用来报告当前灯光的实际亮度值。参数如表 3-2。

表 3-2 Light Lightness 消息

域	大小(字节)	注释
Present Lightness	1	当前灯光实际亮度
Target Lightness	1	目标灯光实际亮度

Remaining Time	1	切换到目标状态剩余时间(可选)，当 Target Lightness 字段存在时才会存在

3.5 Light Lightness Linear Get

此消息用来获取灯光的线性亮度值，远端收到此消息时会回一个 Light Lightness Linear Status 消息。此消息没有参数。

3.6 Light Lightness Linear Set

此消息用来设置灯光的线性亮度值，远端收到此消息时应当调节灯光的线性亮度，并返回 Light Lightness Linear Status 消息来表明当前灯光亮度。参数如表 3-3 所示。

表 3-3 Light Lightness Linear Set 消息

域	大小(字节)	注释
Lightness	2	目标灯光线性亮度值
TID	1	事务 ID，用来标记是新的消息还是重传的消息
Transition Time	1	Transition 的时间(可选)。此字段不存在时，灯光亮度应该立刻改变。存在时，灯光亮度应该在 Transition 的时间之内完成
Delay	1	消息执行的延时，5ms 步进(可选)，当 Transition 存在时，也会存在

3.7 Light Lightness Linear Set Unacknowledged

此消息用来设置灯光的亮度值，远端收到此消息时应当调节灯光的线性亮度，此消息不需要回复。参数同表 3-3。

3.8 Light Lightness Linear Status

此消息用来报告当前灯光的线性亮度值。参数如表 3-4。

表 3-4 Light Lightness Linear 消息

域	大小(字节)	注释

Present Lightness	1	当前灯光线性亮度
Target Lightness	1	目标灯光线性亮度
Remaining Time	1	切换到目标状态剩余时间(可选)，当 Target Lightness 字段存在时才会存在

3.9 Light Lightness Last Get

此消息用来获取远端的 Light Lightness Last 状态，当远端收到此消息时会回复 Light Lightness Last Status 消息。此消息没有参数。

3.10 Light Lightness Last Status

此消息用来报告 Light Lightness Last 的值，参数如表 3-5。

表 3-5 Light Lightness Last 消息

域	大小(字节)	注释
Lightness	2	Light Lightness Last 的值

3.11 Light Lightness Default Get

此消息用于获取默认的实际亮度值，远端收到此消息时会回复 Light Lightness Default Status 消息。此消息没有参数。

3.12 Light Lightness Default Set

此消息用于设置默认的实际亮度值，远端收到此消息时应当记录下此值，并回复 Light Lightness Default Status 消息，参数如表 3-6。

表 3-6 Light Lightness Default Set 消息

域	大小(字节)	注释
Lightness	2	默认的实际亮度值

3.13 Light Lightness Default Set Unacknowledged

此消息用于设置默认的实际亮度值，远端收到此消息时应当记录下此值，此消息不需要回复，参数同表 3-6。

3.14 Light Lightness Default Status

此消息用来报告 Light Lightness Default 的值，参数如表 3-7。

表 3-7 Light Lightness Default Status 消息

域	大小(字节)	注释
Lightness	2	默认的实际亮度值

3.15 Light Lightness Range Get

此消息用于获取实际亮度值的范围，远端收到此消息时会回复 Light Lightness Range Status 消息。此消息没有参数。

3.16 Light Lightness Range Set

此消息用于设置实际亮度值的范围，远端收到此消息时应当记录下此值，并回复 Light Lightness Range Status 消息，参数如表 3-8。

表 3-8 Light Lightness Range Set 消息

域	大小(字节)	注释
Range Min	2	亮度的最小值
Range Max	2	亮度的最大值

3.17 Light Lightness Range Set Unacknowledged

此消息用于设置默认的实际亮度值，远端收到此消息时应当记录下此值，此消息不需要回复，参数同表 3-8。

3.18 Light Lightness Range Status

此消息用来报告 Light Lightness Range 的值，参数如表 3-9。

表 3-9 Light Lightness Range Status 消息

域	大小(字节)	注释
Status Code	1	请求信息的状态码

Range Min	2	亮度最小值
Range Max	2	亮度最大值

4 Light Lightness Server

4.1 接口

表 4-1 Light Lightness Server 接口

函数	作用
light_lightness_server_reg	注册 Light Lightness Server Model
light_lightness_setup_server_reg	注册 Light Lightness Setup Server Model
light_lightness_publish	Publish 实际亮度值
light_lightness_linear_punlish	Publish 线性亮度值
light_lightness_linear_to_actual	线性亮度转换为实际亮度
light_lightness_actual_to_linear	实际亮度转换为线性亮度
light_lightness_to_generic_level	实际亮度值转换为 Generic Level 值
generic_level_to_light_lightness	Generic Level 值转换为实际亮度值
light_lightness_linear_to_generic_level	线性亮度值转换为 Generic Level 值
generic_level_to_light_lightness_linear	Generic Level 值转换为线性亮度值

4.2 消息

4.2.1 LIGHT_LIGHTNESS_SERVER_GET

当 Model 需要获取灯的实际亮度时，会发送此消息到 App 层。消息的参数如表 4-2。

表 4-2 Light Lightness Server Get 消息

```
typedef struct
{
    uint16_t lightness;
} light_lightness_server_get_t;
```

4.2.2 LIGHT_LIGHTNESS_SERVER_GET_LINEAR

当 Model 需要获取灯的线性亮度时，会发送此消息到 App 层。消息的参数同表 4-2。

4.2.3 LIGHT_LIGHTNESS_SERVER_GET_DEFAULT

当 Model 需要获取灯的默认亮度时，会发送此消息到 App 层。消息的参数同表 4-2。

4.2.4 LIGHT_LIGHTNESS_SERVER_GET_LAST

当 Model 需要获取灯的最后非 0 亮度时，会发送此消息到 App 层。消息的参数同表 4-2。

4.2.5 LIGHT_LIGHTNESS_SERVER_GET_RANGE

当 Model 需要获取灯的亮度范围时，会发送此消息到 App 层。消息的参数同表 4-3。

表 4-3 Light Lightness Server Get Range 消息

```
typedef struct
{
    uint16_t range_min;
    uint16_t range_max;
} light_lightness_server_get_range_t;
```

4.2.6 LIGHT_LIGHTNESS_SERVER_GET_DEFAULT_TRANSITION_TIME

Model spec 规定，当设置灯的亮度时如果不带 transition 参数，需要使用默认的 transiton 参数，当默认的 transiton 也不存在时，状态的改变立刻执行。所以 Model 会通过此消息从 App 层获取默认的 transition 参数，在没有默认 transition 参数的情况下，App 可以不处理这个消息。消息参数如表 4-4。

表 4-4 Light Lightness Server Get Default Transition Time 消息

```
typedef struct
{
    generic_transition_time_t trans_time;
} light_lightness_server_get_default_transition_time_t;
```

4.2.7 LIGHT_LIGHTNESS_SERVER_SET

当 Model 收到远端的设置灯亮度消息时会发送此消息给 App 层。消息的参数如表 4-5，lightness 为目标亮度，total_time 为总的 transition 时间，当 total_time 中的 num_steps 为 0 时，转换应该立刻生效，remaining_time 为转换剩余时间，当 remaining_time 中的 num_steps 为 0 时，转换也应该立刻生效。此消息在 transition 的每一步结束时都会通知到 App 层，相应的参数中只有 remaining_time 中的 num_steps 会改变。

表 4-5 Light Lightness Server Set 消息

```
typedef struct
{
    uint16_t lightness;
    generic_transition_time_t total_time;
    generic_transition_time_t remaining_time;
} light_lightness_server_set_t;
```

如果 App 想实现灯的亮度渐变的效果的话，可以在 total_time 和 remaining_time 中的 num_steps 相等时算出每步变化的步长度，之后每一次进入此消息时做一次步长的变化。

4.2.8 LIGHT_LIGHTNESS_SERVER_SET_LINEAR

同 LIGHT_LIGHTNESS_SERVER_SET 消息。

4.2.9 LIGHT_LIGHTNESS_SERVER_SET_LAST

每当灯的亮度变化到一个非 0 值是，此消息都会通知到 App 层，App 应当把此值保存下来供后续使用。消息参数如表 4-6。

表 4-6 Light Lightness Server Set Last 消息

```
typedef struct
{
    uint16_t lightness;
} light_lightness_server_set_last_t;
```

4.2.10 LIGHT_LIGHTNESS_SERVER_SET_DEFAULT

当收到远端设置等的默认亮度值时，此消息会通知到 App 层，App 应当把此值保存下来供后续使用。消息参数如表 4-7。

表 4-7 Light Lightness Server Set Default 消息

```
typedef struct
{
    uint16_t lightness;
} light_lightness_server_set_default_t;
```

4.2.11 LIGHT_LIGHTNESS_SERVER_SET_RANGE

当收到远端设置灯光亮度范围消息时，此消息会通知到 App 层，App 应当把此值保存下来供后续使用。消息参数如表 4-8。

表 4-8 Light Lightness Server Set Range 消息

```
typedef struct
{
    uint16_t range_min;
    uint16_t range_max;
} light_lightness_server_set_range_t;
```

5 Generic OnOff Client

5.1 接口

表 5-1 Generic OnOff Client Model 接口

函数	作用
light_lightness_client_reg	注册 Light Lightness Client Model
light_lightness_get	获取远端的实际光亮度
light_lightness_set	设置远端的实际光亮度
light_lightness_linear_get	获取远端的线性光亮度
light_lightness_linear_set	设置远端的线性光亮度
light_lightness_last_get	获取远端的最近非 0 光亮度
light_lightness_default_get	获取远端的默认光亮度
light_lightness_default_set	设置远端的默认光亮度
light_lightness_range_get	获取远端光亮度的范围
light_lightness_range_set	设置远端光亮度的范围

5.2 消息

5.2.1 LIGHT_LIGHTNESS_CLIENT_STATUS

当 get 或者 set 远端的灯的亮度时，远端会返回 status 消息，Model 会把这个消息格式化后通知到 App 层。参数如表 5-2，其中 optional 为 TRUE 时 target_lightness 和 remaining_time 参数有效，反之亦然。

表 5-2 Light Lightness Client Status 消息

```
typedef struct
{
    uint16_t present_lightness;
    bool optional;
    uint16_t target_lightness;
    generic_transition_time_t remaining_time;
} light_lightness_client_status_t;
```

5.2.2 LIGHT_LIGHTNESS_CLIENT_STATUS_LINEAR

同 LIGHT_LIGHTNESS_CLIENT_STATUS 消息。

5.2.3 LIGHT_LIGHTNESS_CLIENT_STATUS_LAST

当 get 远端的 lightness last 时，远端会回复 lightness last status 消息，Model 会把这个消息格式化后通知到 App 层。参数如表 5-3。

表 5-3 Light Lightness Client Status Last 消息

```
typedef struct
{
    uint16_t lightness;
} light_lightness_client_status_last_t;
```

5.2.4 LIGHT_LIGHTNESS_CLIENT_STATUS_DEFAULT

当 get 或者 set 远端的 lightness default 时，远端会回复 lightness default status 消息，Model 会把这个消息格式化后通知到 App 层。参数如表 5-4。

表 5-4 Light Lightness Client Status Default 消息

```
typedef struct
{
    uint16_t lightness;
} light_lightness_client_status_default_t;
```

5.2.5 LIGHT_LIGHTNESS_CLIENT_STATUS_RANGE

当 get 或者 set 远端的 lightness range 时，远端会回复 lightness range status 消息，Model 会把这个消息格式化后通知到 App 层。参数如表 5-5。

表 5-5 Light Lightness Client Status Range 消息

```
typedef struct
{
    generic_stat_t status;
    uint16_t range_min;
    uint16_t range_max;
} light_lightness_client_status_range_t;
```

6 示例

使用 Light Lightness Model 时主要有以下步骤：

1. 注册 Model
2. 实现 model_data_cb 函数

6.1 Server Model 示例

表 6-1 Light LightnessServer Model 示例

```
/* lightness light models */
static mesh_model_info_t light_lightness_server;

uint16_t current_lightness = 0;
uint16_t lightness_last = 0xffff;

static int32_t light_lightness_server_data(const mesh_model_info_p pmodel_info, uint32_t type,
                                           void *pargs)
{
    switch (type)
    {
    case LIGHT_LIGHTNESS_SERVER_GET:
        {
            light_lightness_server_get_t *pdata = pargs;
            pdata->lightness = current_lightness ;
        }
        break;
    case LIGHT_LIGHTNESS_SERVER_SET:
        {
            light_lightness_server_set_t *pdata = pargs;
            if (0 == pdata->remaining_time.num_steps)
           {
         current_lightness = pdata->lightness;
            }
        }
        break;
    case LIGHT_LIGHTNESS_SERVER_SET_LAST:
        {
            light_lightness_server_set_last_t *pdata = pargs;
            lightness_last = pdata->lightness;
        }
        break;
```

```
    default:
        break;
    }

    return MODEL_SUCCESS;
}

void light_lightness_server_models_init(void)
{
    /* register light lightness models */
    light_lightness_server.model_data_cb = light_lightness_server_data;
    light_lightness_server_reg(0, &light_lightness_server);
}
```

6.2 Client Model 示例

表 6-2 Light Lightness Client Model 示例

```
mesh_model_info_t light_lightness_client;

static int32_t light_lightness_client_data(const mesh_model_info_p pmodel_info,
                                           uint32_t type, void *pargs)
{
    UNUSED(pmodel_info);
    switch (type)
    {
    case LIGHT_LIGHTNESS_CLIENT_STATUS:
        {
            light_lightness_client_status_t *pdata = pargs;
            if (pdata->optional)
            {
                data_uart_debug("light lightness client receive: present = %d, target = %d, remain time =
step(%d), resolution(%d)\r\n",
                                pdata->present_lightness,
                                pdata->target_lightness,
                                pdata->remaining_time.num_steps,
                                pdata->remaining_time.step_resolution);
            }
            else
            {
                data_uart_debug("light lightness client receive: present = %d\r\n", pdata->present_lightness);
            }
        }
```

```
            break;
        case LIGHT_LIGHTNESS_CLIENT_STATUS_LINEAR:
            {
                light_lightness_client_status_t *pdata = pargs;
                if (pdata->optional)
                {
                    data_uart_debug("light lightness client receive: present = %d, target = %d, remain time =
step(%d), resolution(%d)\r\n",
                                    pdata->present_lightness,
                                    pdata->target_lightness,
                                    pdata->remaining_time.num_steps,
                                    pdata->remaining_time.step_resolution);
                }
                else
                {
                    data_uart_debug("light lightness client receive: present = %d\r\n", pdata->present_lightness);
                }
            }
            break;
        case LIGHT_LIGHTNESS_CLIENT_STATUS_LAST:
            {
                light_lightness_client_status_last_t *pdata = pargs;
                data_uart_debug("light lightness client receive: lightness = %d\r\n", pdata->lightness);
            }
            break;
        case LIGHT_LIGHTNESS_CLIENT_STATUS_DEFAULT:
            {
                light_lightness_client_status_default_t *pdata = pargs;
                data_uart_debug("light lightness client receive: lightness = %d\r\n", pdata->lightness);
            }
            break;
        case LIGHT_LIGHTNESS_CLIENT_STATUS_RANGE:
            {
                light_lightness_client_status_range_t *pdata = pargs;
                data_uart_debug("light lightness client receive: status = %d, min = %d, max = %d\r\n",
                                pdata->status, pdata->range_min, pdata->range_max);
            }
            break;
        default:
            break;
        }

        return 0;
}
```

```
void light_client_models_init(void)
{
    light_lightness_client.model_data_cb = light_lightness_client_data;
    light_lightness_client_reg(0, &light_lightness_client);
}
```

参考文献

[1] Mesh Profile Specification

[2] Mesh Model Specification

[3] RTL8762C Mesh SDK User Guide

附录