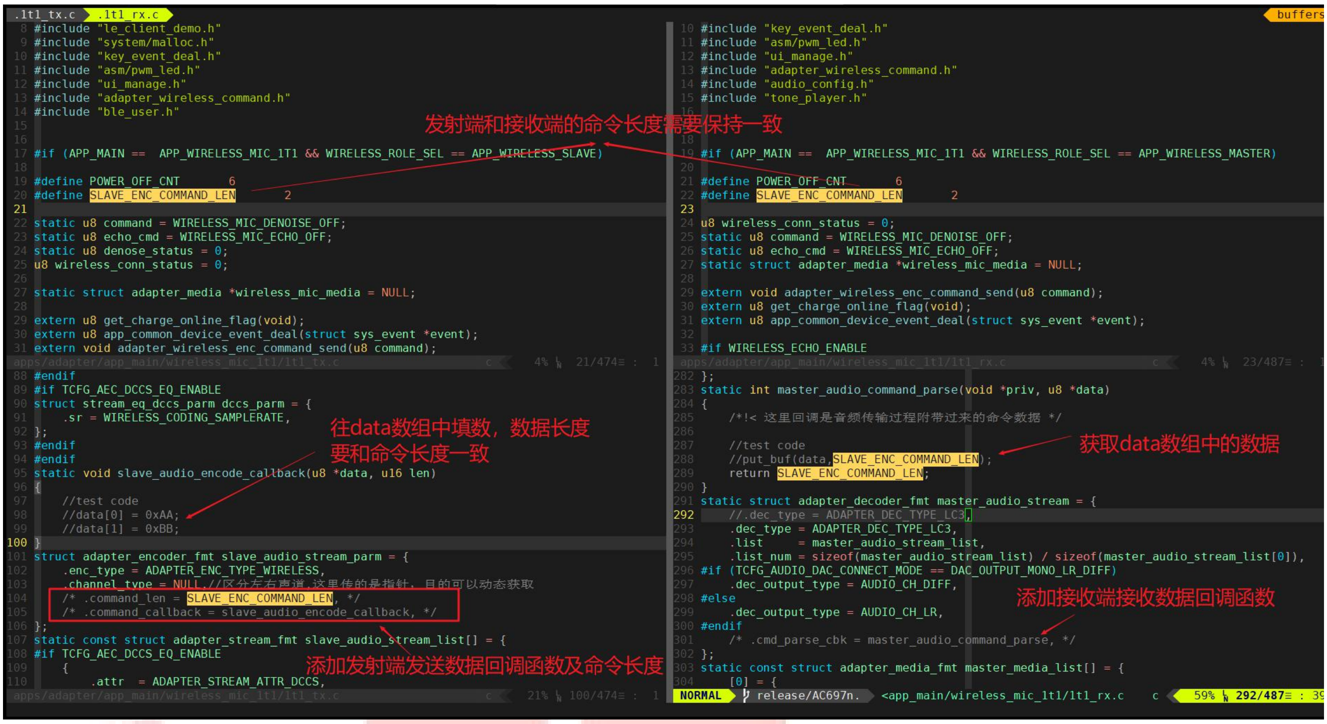
无线麦 SDK 用户自定义数据收发接口说明文档

1，添加发射端发送给接收端自定义数据

一发一收 SDK



```
.1t1_tx.c  .1t1_rx.c
#include "le_client_demo.h"
#include "system/malloc.h"
#include "key_event_deal.h"
#include "asm/pwm_led.h"
#include "ui_manage.h"
#include "adapter_wireless_command.h"
#include "ble_user.h"


#if (APP_MAIN ==  APP_WIRELESS_MIC_1T1 && WIRELESS_ROLE_SEL == APP_WIRELESS_SLAVE)

#define POWER_OFF_CNT       6
#define SLAVE_ENC_COMMAND_LEN       2

static u8 command = WIRELESS_MIC_DENOISE_OFF;
static u8 echo_cmd = WIRELESS_MIC_ECHO_OFF;
static u8 denose_status = 0;
u8 wireless_conn_status = 0;

static struct adapter_media *wireless_mic_media = NULL;

extern u8 get_charge_online_flag(void);
extern u8 app_common_device_event_deal(struct sys_event *event);
extern void adapter_wireless_enc_command_send(u8 command);
apps/adapter/app_main/wireless_mic_1t1/1t1_tx.c         c    4%   21/474≡ :  1
#endif
#if TCFG_AEC_DCCS_EQ_ENABLE
struct stream_eq_dccs_parm dccs_parm = {
    .sr = WIRELESS_CODING_SAMPLERATE,
};
#endif
#endif
static void slave_audio_encode_callback(u8 *data, u16 len)
{
    //test code
    //data[0] = 0xAA;
    //data[1] = 0xBB;
}
struct adapter_encoder_fmt slave_audio_stream_parm = {
    .enc_type = ADAPTER_ENC_TYPE_WIRELESS,
    .channel_type = NULL,//区分左右声道,这里传的是指针, 目的可以动态获取
    /* .command_len = SLAVE_ENC_COMMAND_LEN, */
    /* .command_callback = slave_audio_encode_callback, */
};
static const struct adapter_stream_fmt slave_audio_stream_list[] = {
#if TCFG_AEC_DCCS_EQ_ENABLE
    {
        .attr  = ADAPTER_STREAM_ATTR_DCCS,
apps/adapter/app_main/wireless_mic_1t1/1t1_tx.c         c   21%   100/474≡ :  1
```

```
                                                                    buffers
#include "key_event_deal.h"
#include "asm/pwm_led.h"
#include "ui_manage.h"
#include "adapter_wireless_command.h"
#include "audio_config.h"
#include "tone_player.h"


#if (APP_MAIN ==  APP_WIRELESS_MIC_1T1 && WIRELESS_ROLE_SEL == APP_WIRELESS_MASTER)

#define POWER_OFF_CNT       6
#define SLAVE_ENC_COMMAND_LEN       2

u8 wireless_conn_status = 0;
static u8 command = WIRELESS_MIC_DENOISE_OFF;
static u8 echo_cmd = WIRELESS_MIC_ECHO_OFF;
static struct adapter_media *wireless_mic_media = NULL;

extern void adapter_wireless_enc_command_send(u8 command);
extern u8 get_charge_online_flag(void);
extern u8 app_common_device_event_deal(struct sys_event *event);

#if WIRELESS_ECHO_ENABLE
apps/adapter/app_main/wireless_mic_1t1/1t1_rx.c         c    4%   23/487≡ :  1
};
static int master_audio_command_parse(void *priv, u8 *data)
{
    /*!< 这里回调是音频传输过程附带过来的命令数据 */

    //test code
    //put_buf(data,SLAVE_ENC_COMMAND_LEN);
    return SLAVE_ENC_COMMAND_LEN;
}
static struct adapter_decoder_fmt master_audio_stream = {
    //.dec_type = ADAPTER_DEC_TYPE_LC3,
    .dec_type = ADAPTER_DEC_TYPE_LC3,
    .list     = master_audio_stream_list,
    .list_num = sizeof(master_audio_stream_list) / sizeof(master_audio_stream_list[0]),
#if (TCFG_AUDIO_DAC_CONNECT_MODE == DAC_OUTPUT_MONO_LR_DIFF)
    .dec_output_type = AUDIO_CH_DIFF,
#else
    .dec_output_type = AUDIO_CH_LR,
#endif
    /* .cmd_parse_cbk = master_audio_command_parse, */
};
static const struct adapter_media_fmt master_media_list[] = {
    [0] = {
NORMAL  release/AC697n.  <app_main/wireless_mic_1t1/1t1_rx.c    c   59%   292/487≡ : 39
```

发射端和接收端的命令长度需要保持一致

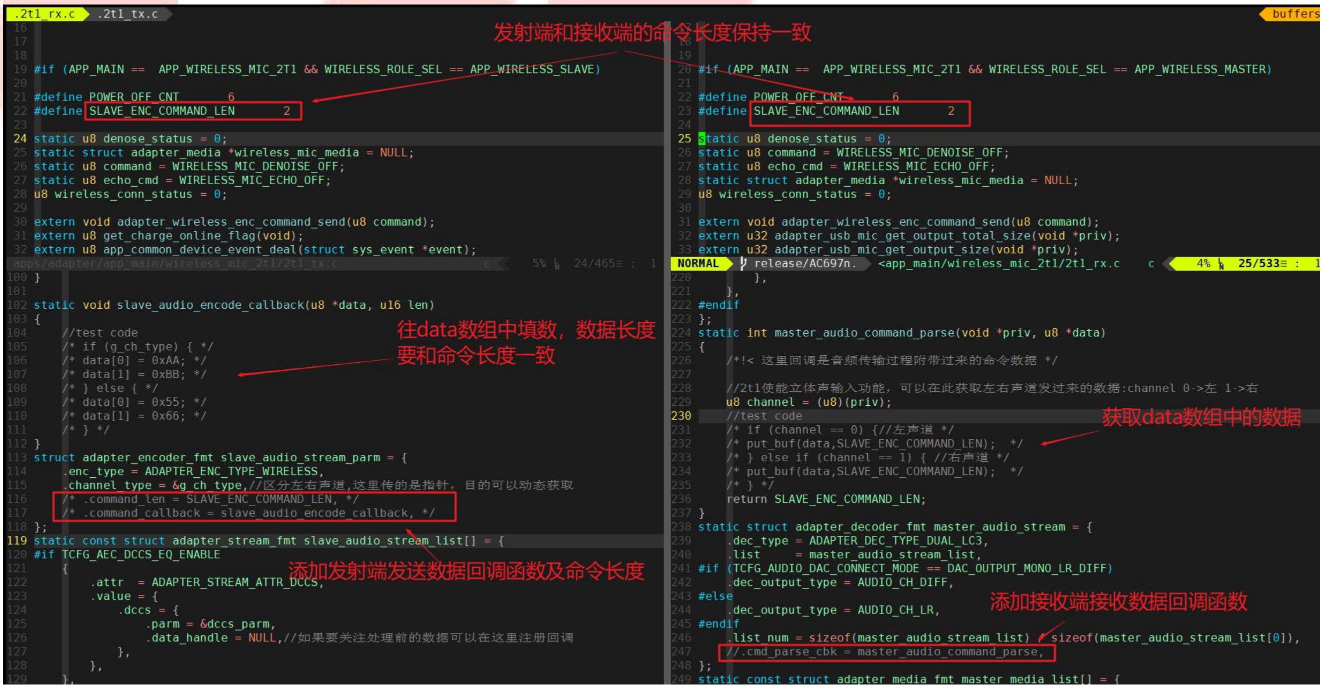
往data数组中填数，数据长度要和命令长度一致

获取data数组中的数据

添加接收端接收数据回调函数

添加发射端发送数据回调函数及命令长度

两发一首 SDK



```
.2t1_rx.c  .2t1_tx.c



#if (APP_MAIN ==  APP_WIRELESS_MIC_2T1 && WIRELESS_ROLE_SEL == APP_WIRELESS_SLAVE)

#define POWER_OFF_CNT       6
#define SLAVE_ENC_COMMAND_LEN       2

static u8 denose_status = 0;
static struct adapter_media *wireless_mic_media = NULL;
static u8 command = WIRELESS_MIC_DENOISE_OFF;
static u8 echo_cmd = WIRELESS_MIC_ECHO_OFF;
u8 wireless_conn_status = 0;

extern void adapter_wireless_enc_command_send(u8 command);
extern u8 get_charge_online_flag(void);
extern u8 app_common_device_event_deal(struct sys_event *event);
apps/adapter/app_main/wireless_mic_2t1/2t1_tx.c         c    5%   24/465≡ :  1
}

static void slave_audio_encode_callback(u8 *data, u16 len)
{
    //test code
    /* if (g_ch_type) { */
    /* data[0] = 0xAA; */
    /* data[1] = 0xBB; */
    /* } else { */
    /* data[0] = 0x55; */
    /* data[1] = 0x66; */
    /* } */
}
struct adapter_encoder_fmt slave_audio_stream_parm = {
    .enc_type = ADAPTER_ENC_TYPE_WIRELESS,
    .channel_type = &g_ch_type,//区分左右声道,这里传的是指针, 目的可以动态获取
    /* .command_len = SLAVE_ENC_COMMAND_LEN, */
    /* .command_callback = slave_audio_encode_callback, */
};
static const struct adapter_stream_fmt slave_audio_stream_list[] = {
#if TCFG_AEC_DCCS_EQ_ENABLE
    {
        .attr  = ADAPTER_STREAM_ATTR_DCCS,
        .value = {
            .dccs = {
                .parm = &dccs_parm,
                .data_handle = NULL,//如果要关注处理前的数据可以在这里注册回调
            },
        },
    },
```

```
                                                                    buffers

#if (APP_MAIN ==  APP_WIRELESS_MIC_2T1 && WIRELESS_ROLE_SEL == APP_WIRELESS_MASTER)

#define POWER_OFF_CNT       6
#define SLAVE_ENC_COMMAND_LEN       2

static u8 denose_status = 0;
static u8 command = WIRELESS_MIC_DENOISE_OFF;
static u8 echo_cmd = WIRELESS_MIC_ECHO_OFF;
static struct adapter_media *wireless_mic_media = NULL;
u8 wireless_conn_status = 0;

extern void adapter_wireless_enc_command_send(u8 command);
extern u32 adapter_usb_mic_get_output_total_size(void *priv);
extern u32 adapter_usb_mic_get_output_size(void *priv);
NORMAL  release/AC697n.  <app_main/wireless_mic_2t1/2t1_rx.c    c    4%   25/533≡ :  1
        },
    },
#endif
};
static int master_audio_command_parse(void *priv, u8 *data)
{
    /*!< 这里回调是音频传输过程附带过来的命令数据 */

    //2t1使能立体声输入功能, 可以在此获取左右声道发过来的数据:channel 0->左 1->右
    u8 channel = (u8)(priv);
    //test code
    /* if (channel == 0) {//左声道 */
    /* put_buf(data,SLAVE_ENC_COMMAND_LEN);  */
    /* } else if (channel == 1) { //右声道 */
    /* put_buf(data,SLAVE_ENC_COMMAND_LEN);  */
    /* } */
    return SLAVE_ENC_COMMAND_LEN;
}
static struct adapter_decoder_fmt master_audio_stream = {
    .dec_type = ADAPTER_DEC_TYPE_DUAL_LC3,
    .list     = master_audio_stream_list,
#if (TCFG_AUDIO_DAC_CONNECT_MODE == DAC_OUTPUT_MONO_LR_DIFF)
    .dec_output_type = AUDIO_CH_DIFF,
#else
    .dec_output_type = AUDIO_CH_LR,
#endif
    .list_num = sizeof(master_audio_stream_list) / sizeof(master_audio_stream_list[0]),
    //.cmd_parse_cbk = master_audio_command_parse,
};
static const struct adapter_media_fmt master_media_list[] = {
```

发射端和接收端的命令长度保持一致

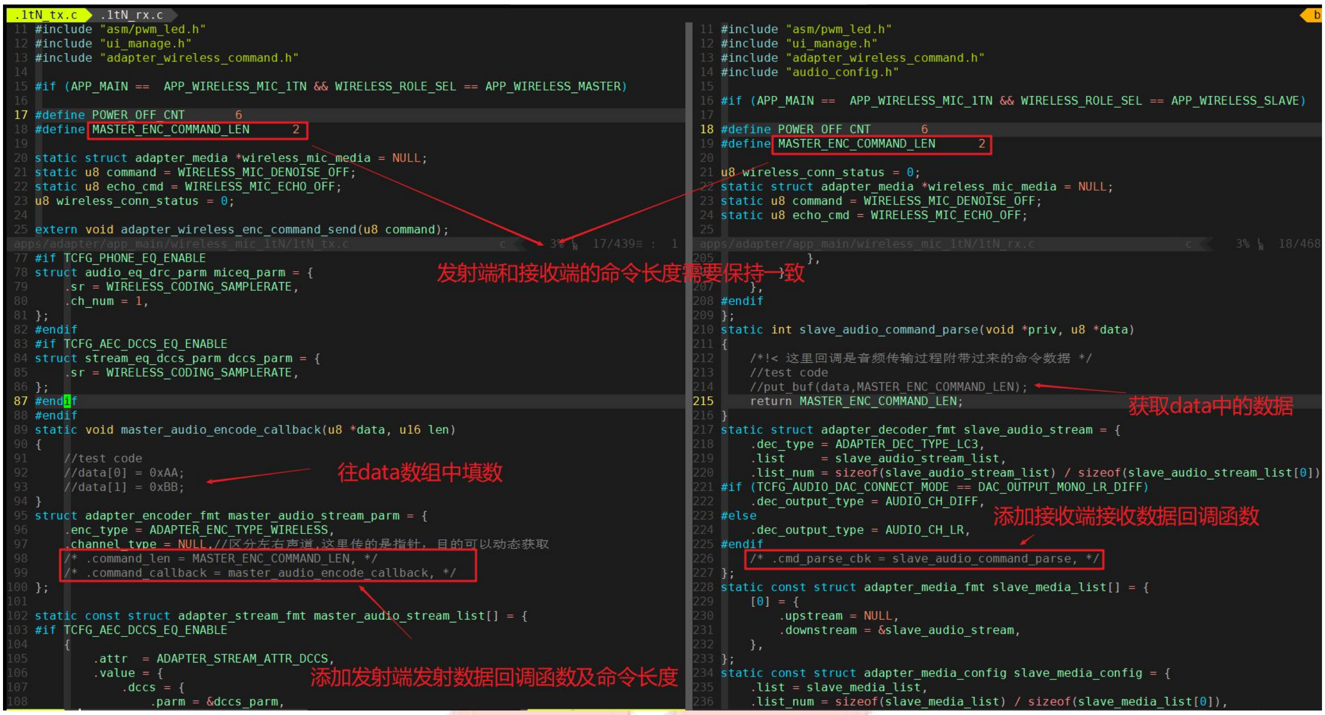
往data数组中填数，数据长度要和命令长度一致

获取data数组中的数据

添加接收端接收数据回调函数

添加发射端发送数据回调函数及命令长度

一发多收 SDK



.1tN_tx.c .1tN_rx.c

```c
#include "asm/pwm_led.h"
#include "ui_manage.h"
#include "adapter_wireless_command.h"

#if (APP_MAIN ==  APP_WIRELESS_MIC_1TN && WIRELESS_ROLE_SEL == APP_WIRELESS_MASTER)

#define POWER_OFF_CNT       6
#define MASTER_ENC_COMMAND_LEN      2

static struct adapter_media *wireless_mic_media = NULL;
static u8 command = WIRELESS_MIC_DENOISE_OFF;
static u8 echo_cmd = WIRELESS_MIC_ECHO_OFF;
u8 wireless_conn_status = 0;

extern void adapter_wireless_enc_command_send(u8 command);
```

apps/adapter/app_main/wireless_mic_1tN/1tN_tx.c   c   3%   17/439≡ : 1

```c
#if TCFG_PHONE_EQ_ENABLE
struct audio_eq_drc_parm miceq_parm = {
    .sr = WIRELESS_CODING_SAMPLERATE,
    .ch_num = 1,
};
#endif
#if TCFG_AEC_DCCS_EQ_ENABLE
struct stream_eq_dccs_parm dccs_parm = {
    .sr = WIRELESS_CODING_SAMPLERATE,
};
#endif
#endif
static void master_audio_encode_callback(u8 *data, u16 len)
{
    //test code
    //data[0] = 0xAA;
    //data[1] = 0xBB;
}
struct adapter_encoder_fmt master_audio_stream_parm = {
    .enc_type = ADAPTER_ENC_TYPE_WIRELESS,
    .channel_type = NULL,//区分左右声道,这里传的是指针, 目的可以动态获取
    /* .command_len = MASTER_ENC_COMMAND_LEN, */
    /* .command_callback = master_audio_encode_callback, */
};

static const struct adapter_stream_fmt master_audio_stream_list[] = {
#if TCFG_AEC_DCCS_EQ_ENABLE
    {
        .attr  = ADAPTER_STREAM_ATTR_DCCS,
        .value = {
            .dccs = {
                .parm = &dccs_parm,
```

```c
#include "asm/pwm_led.h"
#include "ui_manage.h"
#include "adapter_wireless_command.h"
#include "audio_config.h"

#if (APP_MAIN ==  APP_WIRELESS_MIC_1TN && WIRELESS_ROLE_SEL == APP_WIRELESS_SLAVE)

#define POWER_OFF_CNT       6
#define MASTER_ENC_COMMAND_LEN      2

u8 wireless_conn_status = 0;
static struct adapter_media *wireless_mic_media = NULL;
static u8 command = WIRELESS_MIC_DENOISE_OFF;
static u8 echo_cmd = WIRELESS_MIC_ECHO_OFF;

```

apps/adapter/app_main/wireless_mic_1tN/1tN_rx.c   c   3%   18/468

```c
            },
        }
    },
#endif
};
static int slave_audio_command_parse(void *priv, u8 *data)
{
    /*!< 这里回调是音频传输过程附带过来的命令数据 */
    //test code
    //put_buf(data,MASTER_ENC_COMMAND_LEN);
    return MASTER_ENC_COMMAND_LEN;
}
static struct adapter_decoder_fmt slave_audio_stream = {
    .dec_type = ADAPTER_DEC_TYPE_LC3,
    .list     = slave_audio_stream_list,
    .list_num = sizeof(slave_audio_stream_list) / sizeof(slave_audio_stream_list[0]),
#if (TCFG_AUDIO_DAC_CONNECT_MODE == DAC_OUTPUT_MONO_LR_DIFF)
    .dec_output_type = AUDIO_CH_DIFF,
#else
    .dec_output_type = AUDIO_CH_LR,
#endif
    /* .cmd_parse_cbk = slave_audio_command_parse, */
};
static const struct adapter_media_fmt slave_media_list[] = {
    [0] = {
        .upstream = NULL,
        .downstream = &slave_audio_stream,
    },
};
static const struct adapter_media_config slave_media_config = {
    .list = slave_media_list,
    .list_num = sizeof(slave_media_list) / sizeof(slave_media_list[0]),
```

发射端和接收端的命令长度需要保持一致

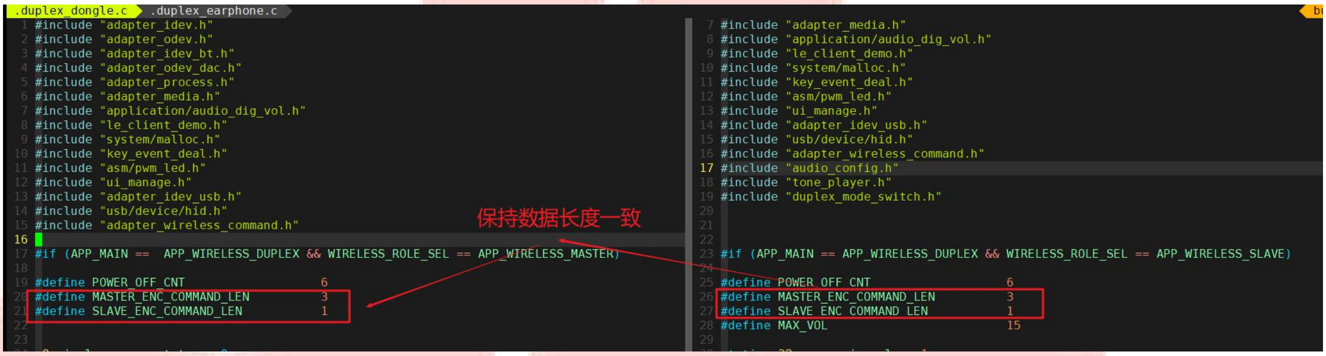
往data数组中填数

添加发射端发射数据回调函数及命令长度

获取data中的数据

添加接收端接收数据回调函数

双工 SDK

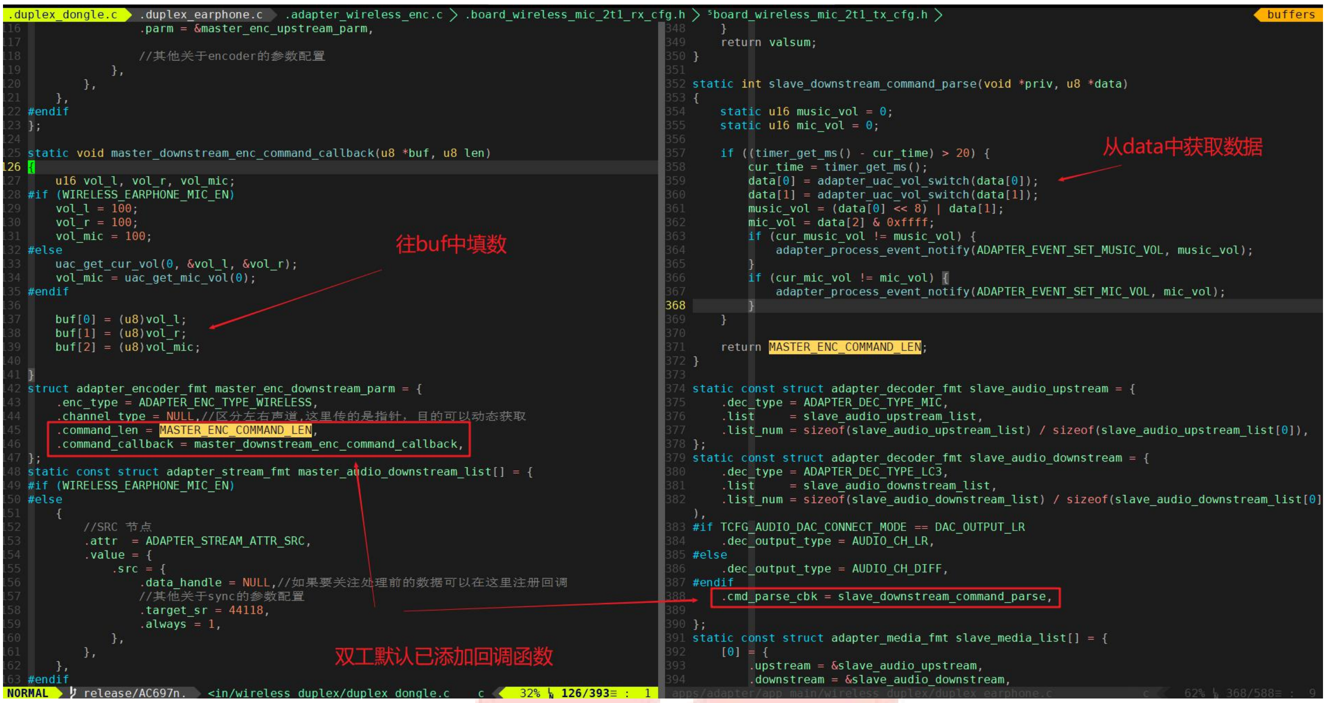


.duplex_dongle.c .duplex_earphone.c

```c
#include "adapter_idev.h"
#include "adapter_odev.h"
#include "adapter_idev_bt.h"
#include "adapter_odev_dac.h"
#include "adapter_process.h"
#include "adapter_media.h"
#include "application/audio_dig_vol.h"
#include "le_client_demo.h"
#include "system/malloc.h"
#include "key_event_deal.h"
#include "asm/pwm_led.h"
#include "ui_manage.h"
#include "adapter_idev_usb.h"
#include "usb/device/hid.h"
#include "adapter_wireless_command.h"

#if (APP_MAIN ==  APP_WIRELESS_DUPLEX && WIRELESS_ROLE_SEL == APP_WIRELESS_MASTER)

#define POWER_OFF_CNT                   6
#define MASTER_ENC_COMMAND_LEN          3
#define SLAVE_ENC_COMMAND_LEN           1

```

```c
#include "adapter_media.h"
#include "application/audio_dig_vol.h"
#include "le_client_demo.h"
#include "system/malloc.h"
#include "key_event_deal.h"
#include "asm/pwm_led.h"
#include "ui_manage.h"
#include "adapter_idev_usb.h"
#include "usb/device/hid.h"
#include "adapter_wireless_command.h"
#include "audio_config.h"
#include "tone_player.h"
#include "duplex_mode_switch.h"



#if (APP_MAIN == APP_WIRELESS_DUPLEX && WIRELESS_ROLE_SEL == APP_WIRELESS_SLAVE)

#define POWER_OFF_CNT                   6
#define MASTER_ENC_COMMAND_LEN          3
#define SLAVE_ENC_COMMAND_LEN           1
#define MAX_VOL                         15

```

保持数据长度一致



.duplex_dongle.c 〉 .duplex_earphone.c 〉 .adapter_wireless_enc.c 〉 .board_wireless_mic_2t1_rx_cfg.h 〉 ⁵board_wireless_mic_2t1_tx_cfg.h 〉 buffers

```c
                .parm = &master_enc_upstream_parm,

                //其他关于encoder的参数配置
            },
        },
    },
#endif
};

static void master_downstream_enc_command_callback(u8 *buf, u8 len)
{
    u16 vol_l, vol_r, vol_mic;
#if (WIRELESS_EARPHONE_MIC_EN)
    vol_l = 100;
    vol_r = 100;
    vol_mic = 100;
#else
    uac_get_cur_vol(0, &vol_l, &vol_r);
    vol_mic = uac_get_mic_vol(0);
#endif

    buf[0] = (u8)vol_l;
    buf[1] = (u8)vol_r;
    buf[2] = (u8)vol_mic;

}
struct adapter_encoder_fmt master_enc_downstream_parm = {
    .enc_type = ADAPTER_ENC_TYPE_WIRELESS,
    .channel_type = NULL,//区分左右声道,这里传的是指针, 目的可以动态获取
    .command_len = MASTER_ENC_COMMAND_LEN,
    .command_callback = master_downstream_enc_command_callback,
};
static const struct adapter_stream_fmt master_audio_downstream_list[] = {
#if (WIRELESS_EARPHONE_MIC_EN)
#else
    {
        //SRC 节点
        .attr  = ADAPTER_STREAM_ATTR_SRC,
        .value = {
            .src = {
                .data_handle = NULL,//如果要关注处理前的数据可以在这里注册回调
                //其他关于sync的参数配置
                .target_sr = 44118,
                .always = 1,
            },
        },
    },
#endif
```

NORMAL 〉 release/AC697n. 〉 <in/wireless_duplex/duplex_dongle.c c 〈 32% 126/393≡ : 1

往buf中填数

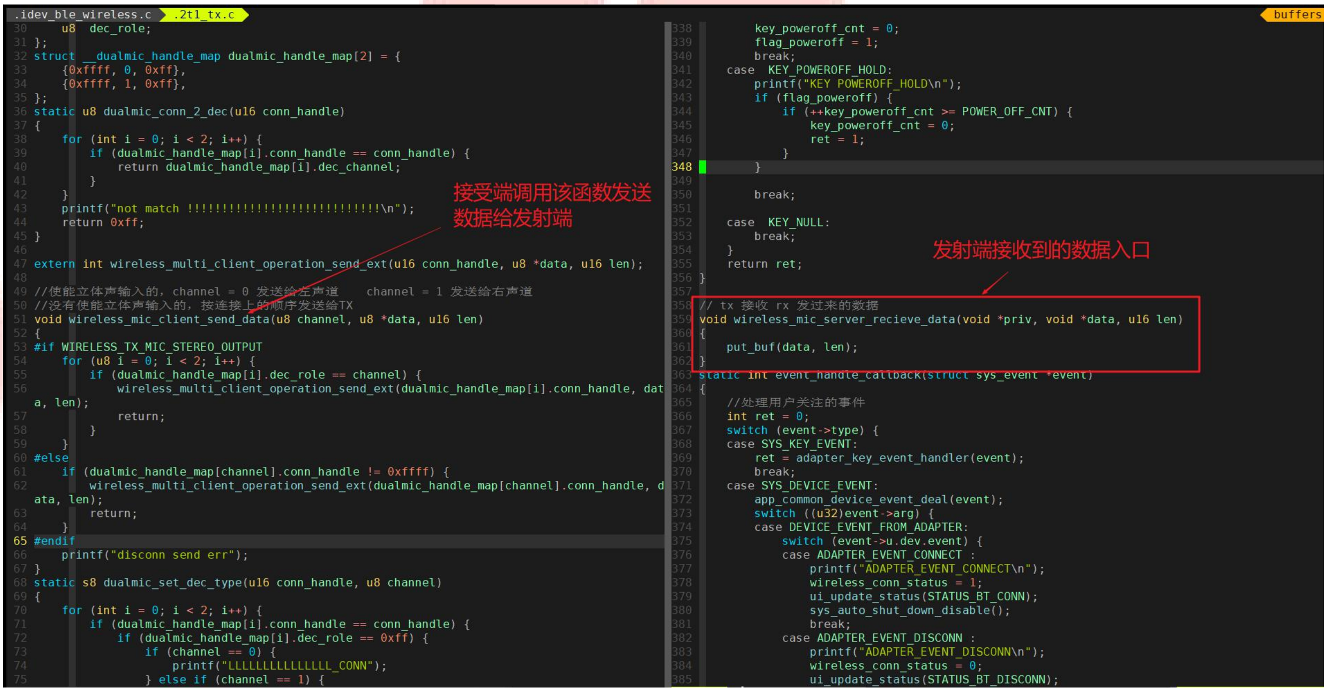
双工默认已添加回调函数

```c
    }
    return valsum;
}

static int slave_downstream_command_parse(void *priv, u8 *data)
{
    static u16 music_vol = 0;
    static u16 mic_vol = 0;

    if ((timer_get_ms() - cur_time) > 20) {
        cur_time = timer_get_ms();
        data[0] = adapter_uac_vol_switch(data[0]);
        data[1] = adapter_uac_vol_switch(data[1]);
        music_vol = (data[0] << 8) | data[1];
        mic_vol = data[2] & 0xffff;
        if (cur_music_vol != music_vol) {
            adapter_process_event_notify(ADAPTER_EVENT_SET_MUSIC_VOL, music_vol);
        }
        if (cur_mic_vol != mic_vol) {
            adapter_process_event_notify(ADAPTER_EVENT_SET_MIC_VOL, mic_vol);
        }
    }

    return MASTER_ENC_COMMAND_LEN;
}

static const struct adapter_decoder_fmt slave_audio_upstream = {
    .dec_type = ADAPTER_DEC_TYPE_MIC,
    .list     = slave_audio_upstream_list,
    .list_num = sizeof(slave_audio_upstream_list) / sizeof(slave_audio_upstream_list[0]),
};
static const struct adapter_decoder_fmt slave_audio_downstream = {
    .dec_type = ADAPTER_DEC_TYPE_LC3,
    .list     = slave_audio_downstream_list,
    .list_num = sizeof(slave_audio_downstream_list) / sizeof(slave_audio_downstream_list[0]
),
#if TCFG_AUDIO_DAC_CONNECT_MODE == DAC_OUTPUT_LR
    .dec_output_type = AUDIO_CH_LR,
#else
    .dec_output_type = AUDIO_CH_DIFF,
#endif
    .cmd_parse_cbk = slave_downstream_command_parse,

};
static const struct adapter_media_fmt slave_media_list[] = {
    [0] = {
        .upstream = &slave_audio_upstream,
        .downstream = &slave_audio_downstream,
```

apps/adapter/app_main/wireless_duplex/duplex_earphone.c c 〈 62% 368/588≡ : 9

从data中获取数据

2，两发一收接收端发送数据给发射端



.idev_ble_wireless.c 〉 .2t1_tx.c 〉 buffers

```c
    u8  dec_role;
};
struct __dualmic_handle_map dualmic_handle_map[2] = {
    {0xffff, 0, 0xff},
    {0xffff, 1, 0xff},
};
static u8 dualmic_conn_2_dec(u16 conn_handle)
{
    for (int i = 0; i < 2; i++) {
        if (dualmic_handle_map[i].conn_handle == conn_handle) {
            return dualmic_handle_map[i].dec_channel;
        }
    }
    printf("not match !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!\n");
    return 0xff;
}

extern int wireless_multi_client_operation_send_ext(u16 conn_handle, u8 *data, u16 len);

//使能立体声输入的, channel = 0 发送给左声道    channel = 1 发送给右声道
//没有使能立体声输入的, 按连接上的顺序发送给TX
void wireless_mic_client_send_data(u8 channel, u8 *data, u16 len)
{
#if WIRELESS_TX_MIC_STEREO_OUTPUT
    for (u8 i = 0; i < 2; i++) {
        if (dualmic_handle_map[i].dec_role == channel) {
            wireless_multi_client_operation_send_ext(dualmic_handle_map[i].conn_handle, dat
a, len);
            return;
        }
    }
#else
    if (dualmic_handle_map[channel].conn_handle != 0xffff) {
        wireless_multi_client_operation_send_ext(dualmic_handle_map[channel].conn_handle, d
ata, len);
        return;
    }
#endif
    printf("disconn send err");
}
static s8 dualmic_set_dec_type(u16 conn_handle, u8 channel)
{
    for (int i = 0; i < 2; i++) {
        if (dualmic_handle_map[i].conn_handle == conn_handle) {
            if (dualmic_handle_map[i].dec_role == 0xff) {
                if (channel == 0) {
                    printf("LLLLLLLLLLLLLLL_CONN");
                } else if (channel == 1) {
```

接受端调用该函数发送数据给发射端

```c
        key_poweroff_cnt = 0;
        flag_poweroff = 1;
        break;
    case  KEY_POWEROFF_HOLD:
        printf("KEY POWEROFF_HOLD\n");
        if (flag_poweroff) {
            if (++key_poweroff_cnt >= POWER_OFF_CNT) {
                key_poweroff_cnt = 0;
                ret = 1;
            }
        }

        break;

    case  KEY_NULL:
        break;
    }
    return ret;
}

// tx 接收 rx 发过来的数据
void wireless_mic_server_recieve_data(void *priv, void *data, u16 len)
{
    put_buf(data, len);
}
static int event_handle_callback(struct sys_event *event)
{
    //处理用户关注的事件
    int ret = 0;
    switch (event->type) {
    case SYS_KEY_EVENT:
        ret = adapter_key_event_handler(event);
        break;
    case SYS_DEVICE_EVENT:
        app_common_device_event_deal(event);
        switch ((u32)event->arg) {
        case DEVICE_EVENT_FROM_ADAPTER:
            switch (event->u.dev.event) {
            case ADAPTER_EVENT_CONNECT :
                printf("ADAPTER_EVENT_CONNECT\n");
                wireless_conn_status = 1;
                ui_update_status(STATUS_BT_CONN);
                sys_auto_shut_down_disable();
                break;
            case ADAPTER_EVENT_DISCONN :
                printf("ADAPTER_EVENT_DISCONN\n");
                wireless_conn_status = 0;
                ui_update_status(STATUS_BT_DISCONN);
```

发射端接收到的数据入口