XRADIO Audio Developer Guide

Revision 2.3

Aug 11, 2020

Declaration

THIS DOCUMENTATION IS THE ORIGINAL WORK AND COPYRIGHTED PROPERTY OF XRADIO TECHNOLOGY ("XRADIO"). REPRODUCTION IN WHOLE OR IN PART MUST OBTAIN THE WRITTEN APPROVAL OF XRADIO AND GIVE CLEAR ACKNOWLEDGEMENT TO THE COPYRIGHT OWNER.

THE PURCHASED PRODUCTS, SERVICES AND FEATURES ARE STIPULATED BY THE CONTRACT MADE BETWEEN XRADIO AND THE CUSTOMER. PLEASE READ THE TERMS AND CONDITIONS OF THE CONTRACT AND RELEVANT INSTRUCTIONS CAREFULLY BEFORE USING, AND FOLLOW THE INSTRUCTIONS IN THIS DOCUMENTATION STRICTLY. XRADIO ASSUMES NO RESPONSIBILITY FOR THE CONSEQUENCES OF IMPROPER USE (INCLUDING BUT NOT LIMITED TO OVERVOLTAGE, OVERCLOCK, OR EXCESSIVE TEMPERATURE).

THE INFORMATION FURNISHED BY XRADIO IS PROVIDED JUST AS A REFERENCE OR TYPICAL APPLICATIONS, ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS DOCUMENT DO NOT CONSTITUTE A WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. XRADIO RESERVES THE RIGHT TO MAKE CHANGES IN CIRCUIT DESIGN AND/OR SPECIFICATIONS AT ANY TIME WITHOUT NOTICE.

NOR FOR ANY INFRINGEMENTS OF PATENTS OR OTHER RIGHTS OF THE THIRD PARTIES WHICH MAY RESULT FROM ITS USE. NO LICENSE IS GRANTED BY IMPLICATION OR OTHERWISE UNDER ANY PATENT OR PATENT RIGHTS OF XRADIO.THIRD PARTY LICENCES MAY BE REQUIRED TO IMPLEMENT THE SOLUTION/PRODUCT. CUSTOMERS SHALL BE SOLELY RESPONSIBLE TO OBTAIN ALL APPROPRIATELY REQUIRED THIRD PARTY LICENCES. XRADIO SHALL NOT BE LIABLE FOR ANY LICENCE FEE OR ROYALTY DUE IN RESPECT OF ANY REQUIRED THIRD PARTY LICENCE. XRADIO SHALL HAVE NO WARRANTY, INDEMNITY OR OTHER OBLIGATIONS WITH RESPECT TO MATTERS COVERED UNDER ANY REQUIRED THIRD PARTY LICENCE.

Revision History

Version	Date	Summary of Changes
1.0	2017-12-21	Initial Version
2.0	2019-10-22	Use Player_app and Recorder_app to play and record, instead of using cedarx directly
2.1	2019-11-2	update to template
2.2	2019-11-18	add play pcm data and record pcm data by audio driver
2.3	2020-08-11	fix misdescription

Table 1- 1 Revision History

Contents

Tables

1 概述

1.1 介绍

Xradio 支持音频播放和录制。对于纯音频数据的 pcm 音频，可以直接使用音频驱动进行播放和录制；对于需要进行编解码的音频，则可使用 Cedarx 进行播放和录制。

CedarX 是一套多媒体中间件，主要负责音乐媒体的播放和录制。内部集成了流媒体处理、封装处理、编解码等复杂逻辑，对外提供简单的控制接口便于使用。目前支持如下：

- 目前支持 HTTP、HTTPS、File（TF/SD 卡）、Flash、音频数据流、自定义音频源播放
- 目前支持 MP3、AMR、AAC、M4A、M3U8、WAV 播放
- 目前支持 AMR、PCM 录音

CedarX 包含两个模块——XPlayer 播放、XRecord 录音，两个模块互相独立。XPlayer 负责把音频媒体转化成驱动可播放的 PCM 格式，驱动拿到 PCM 数据后则转化到喇叭上，播放出最终我们听到的声音。XRecord 负责把驱动录下来的 PCM 数据转化成指定格式数据，存储在 SD/TF 卡。

player_app 和 recorder_app 模块是对 XPlayer 播放模块和 XRecord 录音模块的封装，目的是使其播放/录音接口更加简单易用。

下文主要介绍如何使用 player_app、recorder_app 以及音频驱动模块进行音频播放与音频录制。

2 Cedarx 功能描述

2.1 播放功能选择

目前 Cedarx 支持多种音频来源和音频格式的播放，在播放前需要先选择需要的播放功能。播放功能的选择由下述接口实现。目前这些接口在开发板启动阶段由函数 platform_cedarx_init()调用，用户可自行选择是否调用相应的接口，来获取对应的播放功能。一共分为音频来源、音频格式、解码器三类接口。

注：下述接口只能被调用一次，不可重复多次调用。

Table 2-1 音频来源选择接口描述

function	detail
CedarxStreamListInit;	声明：int CedarxStreamListInit(void); 目的：初始化音频来源支持列表。该函数必须被调用，且必须在注册 https、http 等音频源前被调用 参数：无 返回值：返回状态 (0: 成功)
CedarxStreamRegisterHttps;	声明：int CedarxStreamRegisterHttps(void); 目的：注册 https 音频源。播放 https 音频时该函数需要被调用 参数：无 返回值：返回状态 (0: 成功)
CedarxStreamRegisterSsl;	声明：int CedarxStreamRegisterSsl(void); 目的：注册 ssl 音频源。ssl 是 https 的辅助音频源，播放 https 音频时该函数需要被调用 参数：无 返回值：返回状态 (0: 成功)
CedarxStreamRegisterFlash;	声明：int CedarxStreamRegisterFlash(void); 目的：注册 flash 音频源。播放 flash 音频时该函数需要被调用 参数：无 返回值：返回状态 (0: 成功)
CedarxStreamRegisterFile;	声明：int CedarxStreamRegisterFile(void); 目的：注册 file 音频源。播放 sd/tf 卡里的音频（即 file 音频）时该函数需

	要被调用 参数：无 返回值：返回状态 (0: 成功)
CedarxStreamRegisterFifo;	声明：int CedarxStreamRegisterFifo(void); 目的：注册 fifo 音频源。播放音频数据流的音频时该函数需要被调用 参数：无 返回值：返回状态 (0: 成功)
CedarxStreamRegisterHttp;	声明：int CedarxStreamRegisterHttp(void); 目的：注册 http 音频源。播放 http 音频时该函数需要被调用 参数：无 返回值：返回状态 (0: 成功)
CedarxStreamRegisterTcp;	声明：int CedarxStreamRegisterTcp(void); 目的：注册 tcp 音频源。tcp 是 http 的辅助音频源，播放 http 音频时该函数需要被调用 参数：无 返回值：返回状态 (0: 成功)
CedarxStreamRegisterCustomer;	声明：int CedarxStreamRegisterCustomer(void); 目的：注册 customer 音频源。customer 是用户自定义音频来源，用户需要定制音频来源时该函数需要被调用 参数：无 返回值：返回状态 (0: 成功)

Table 2-2 音频格式选择接口描述

function	detail
CedarxParserListInit;	声明：int CedarxParserListInit(void); 目的：初始化音频格式支持列表。该函数必须被调用，且必须在注册 mp3、amr 等播放格式前被调用 参数：无 返回值：返回状态 (0: 成功)

CedarxParserRegisterM3U;	声明：int CedarxParserRegisterM3U(void); 目的：注册 m3u8 音频格式。播放 m3u8 音频时该函数需要被调用 参数：无 返回值：返回状态 (0: 成功)
CedarxParserRegisterM4A;	声明：int CedarxParserRegisterM4A(void); 目的：注册 m4a 音频格式。播放 m4a 音频时该函数需要被调用 参数：无 返回值：返回状态 (0: 成功)
CedarxParserRegisterAAC;	声明：int CedarxParserRegisterAAC(void); 目的：注册 aac 音频格式。播放 aac 音频时该函数需要被调用 参数：无 返回值：返回状态 (0: 成功)返回值：返回状态 (0: 成功)
CedarxParserRegisterAMR;	声明：int CedarxParserRegisterAMR(void); 目的：注册 amr 音频格式。播放 amr 音频时该函数需要被调用 参数：无 返回值：返回状态 (0: 成功)
CedarxParserRegisterMP3;	声明：int CedarxParserRegisterMP3(void); 目的：注册 mp3 音频格式。播放 mp3 音频时该函数需要被调用 参数：无 返回值：返回状态 (0: 成功)
CedarxParserRegisterWAV;	声明：int CedarxParserRegisterWAV(void); 目的：注册 wav 音频格式。播放 wav 音频时该函数需要被调用 参数：无 返回值：返回状态 (0: 成功)

Table 2-3 解码器选择接口描述

function	detail
CedarxDecoderListInit;	声明：int CedarxDecoderListInit(void); 目的：初始化解码器支持列表。该函数必须被调用，且必须在注册 AAC、

	MP3 等解码器前被调用 参数：无 返回值：返回状态 (0: 成功)
CedarxDecoderRegisterAAC;	声明：int CedarxDecoderRegisterAAC(void); 目的：注册 AAC 解码器。播放 m4a 和 aac 音频时，该函数需要被调用。当 m3u8 里的播放列表存在 m4a 或 aac 时，该函数也需要被调用 参数：无 返回值：返回状态 (0: 成功)
CedarxDecoderRegisterAMR;	声明：int CedarxDecoderRegisterAMR(void); 目的：注册 AMR 解码器。播放 amr 音频时该函数需要被调用 参数：无 返回值：返回状态 (0: 成功)
CedarxDecoderRegisterMP3;	声明：int CedarxDecoderRegisterMP3(void); 目的：注册 MP3 解码器。播放 mp3 音频时该函数需要被调用 参数：无 返回值：返回状态 (0: 成功)
CedarxDecoderRegisterWAV;	声明：int CedarxDecoderRegisterWAV(void); 目的：注册 WAV 解码器。播放 wav 音频时该函数需要被调用 参数：无 返回值：返回状态 (0: 成功)

2.2 播放接口说明

音频播放需要先创建一个播放实体，创建的函数 api 如下：

Table 2-4 播放器创建接口描述

function	detail
player_create();	声明：player_base * player_create(); 目的：创建一个播放器实体 参数：无

	返回值：播放器实体

其中播放器实体 player_base 包含了该播放器支持的操作方法，用户可利用这些操作方法进行音频播放、暂停、停止等操作，结构体定义如下：

```
typedef struct player_base
{
    int (*play)(struct player_base *base, const char *url);
    int (*stop)(struct player_base *base);
    int (*pause)(struct player_base *base);
    int (*resume)(struct player_base *base);
    int (*seek)(struct player_base *base, int ms);
    int (*tell)(struct player_base *base);
    int (*size)(struct player_base *base);
    int (*setvol)(struct player_base *base, int vol);
    int (*getvol)(struct player_base *base);
    int (*mute)(struct player_base *base, bool is_mute);
    int (*is_mute)(struct player_base *base);
    int (*control)(struct player_base *base, player_cmd command, void *data);
    void (*set_callback)(struct player_base *base, app_player_callback cb, void *arg);
    aplayer_states (*get_status)(struct player_base *base);
} player_base;
```

其中各个方法的描述如下：

Table 2-5 player_base 接口描述

function	detail
play;	声明：int (*play)(struct player_base *base, const char *url); 目的：播放一个 url 音频 参数： base：创建的播放器实体; url：待播放的 url 返回值：返回状态
stop;	声明：int (*stop)(struct player_base *base);

	目的：停止播放 参数： base：创建的播放器实体 返回值：返回状态
pause;	声明：int (*pause)(struct player_base *base); 目的：暂停播放 参数： base：创建的播放器实体 返回值：返回状态
resume;	声明：int (*resume)(struct player_base *base); 目的：重新启动播放 参数： base：创建的播放器实体 返回值：返回状态
seek;	声明：int (*seek)(struct player_base *base, int ms); 目的：使播放器跳转到指定位置进行播放 参数： base：创建的播放器实体； ms：待跳转的位置 返回值：返回状态
tell;	声明：int (*tell)(struct player_base *base); 目的：获取当前的播放进度 参数： base：创建的播放器实体 返回值：当前的播放进度
size;	声明：int (*size)(struct player_base *base); 目的：获取当前播放音频的总长度 参数： base：创建的播放器实体 返回值：当前播放音频的总长度
setvol;	声明：int (*setvol)(struct player_base *base, int vol); 目的：设置音量 参数：base：创建的播放器实体； vol：音量值 返回值：返回状态
getvol;	声明：int (*getvol)(struct player_base *base);

	目的：获取当前音量 参数： base：创建的播放器实体 返回值：返回音量值
mute;	声明：int (*mute)(struct player_base *base, bool is_mute); 目的：静音 参数： base：创建的播放器实体； is_mute：静音使能 返回值：返回状态
is_mute;	声明：int (*is_mute)(struct player_base *base); 目的：获取静音状态 参数： base：创建的播放器实体 返回值：返回是否静音
control;	声明：int (*control)(struct player_base *base, player_cmd command, void *data); 目的：控制播放器，目前不支持 参数： base：创建的播放器实体；command：命令；data：相应的参数 返回值：返回状态
set_callback;	声明： void (*set_callback)(struct player_base *base, app_player_callback cb, void *arg); 目的：设置回调函数 参数： base：创建的播放器实体；cb：设置的回调函数；arg:回调函数的参数 返回值：无
get_status;	声明：aplayer_states (*get_status)(struct player_base *base); 目的：获取播放器状态 参数： base：创建的播放器实体 返回值：返回播放器状态

2.3录音功能选择

Cedarx 支持多种音频存放地址和音频格式的录制。在录音之前，需要先选择需要的录音功能。录音功能的选择由下述接口实现。目前这些接口在开发板启动阶段由函数 platform_cedarx_init()调用，用户可自行选择是否调用相应的接口，来获取对应的录音功能。一共分为音频地址、音频格式、编码器三类接口。

注：下述接口只能被调用一次，不可重复多次调用

Table 2-6 音频地址选择接口描述

function	detail
CedarxWriterListInit;	声明：int CedarxWriterListInit(void); 目的：初始化音频地址支持列表。该函数必须被调用，且必须在注册 file、customer 等音频目的地址前被调用 参数：无 返回值：返回状态 (0: 成功)
CedarxWriterRegisterFile;	声明：int CedarxWriterRegisterFile(void); 目的：注册 file 音频地址。当存放录制的音频到 sd/tf 卡时该函数需要被调用 参数：无 返回值：返回状态 (0: 成功)
CedarxWriterRegisterCallback;	声明：int CedarxWriterRegisterCallback(void); 目的：注册 callback 音频地址。callback 音频地址是自定义存放地址的一种实现方式，用户需实现自定义存放的函数实现 参数：无 返回值：返回状态 (0: 成功)
CedarxWriterRegisterCustomer;	声明：int CedarxWriterRegisterCustomer(void); 目的：注册 customer 音频地址。customer 音频地址是自定义存放地址的另外一种实现方式，用户需实现自定义的 writer。推荐使用该方式实现自定义存放地址 参数：无 返回值：返回状态 (0: 成功)

Table 2-7 音频格式选择接口描述

function	detail
CedarxMuxerListInit;	声明：int CedarxMuxerListInit(void); 目的：初始化音频格式支持列表。该函数必须被调用，且必须在注册 pcm、amr 等音频格式前被调用 参数：无

	返回值：返回状态 (0: 成功)
CedarxMuxerRegisterAmr;	声明：int CedarxMuxerRegisterAmr(void); 目的：注册 amr 音频格式。录制 amr 音频时该函数需要被调用 参数：无 返回值：返回状态 (0: 成功)
CedarxMuxerRegisterPcm;	声明：int CedarxMuxerRegisterPcm(void); 目的：注册 pcm 音频格式。录制 pcm 音频时该函数需要被调用 参数：无 返回值：返回状态 (0: 成功)

Table 2-8 编码器选择接口描述

function	detail
CedarxEncoderListInit;	声明：int CedarxEncoderListInit(void); 目的：初始化编码器支持列表。该函数必须被调用，且必须在注册 AMR、PCM 等编码器前被调用 参数：无 返回值：返回状态 (0: 成功)
CedarxEncoderRegisterAmr;	声明：int CedarxEncoderRegisterAmr(void); 目的：注册 AMR 编码器。录制 amr 音频时，该函数需要被调用 参数：无 返回值：返回状态 (0: 成功)
CedarxEncoderRegisterPcm;	声明：int CedarxEncoderRegisterPcm(void); 目的：注册 PCM 编码器。录制 pcm 音频时该函数需要被调用 参数：无 返回值：返回状态 (0: 成功)

2.4录音接口说明

录音前需要先创建一个播放实体，创建的函数 api 如下

Table 2-9 录音器创建接口描述

function	detail
recorder_create();	声明：recorder_base *recorder_create(); 目的：创建一个录音器实体 参数：无 返回值：录音器实体

其中录音器实体 recorder_base 包含了该录音器支持的操作方法，用户可利用这些操作方法进行录音，结构体定义如下：

```
struct recorder_base
{
    int (*start)(recorder_base *base, const char *url, const rec_cfg *cfg);
    int (*stop)(recorder_base *base);
};
```

其中各个方法的描述如下：

Table 2-10 recorder_base 接口描述

function	detail
start;	声明：int (*start)(recorder_base *base, const char *url, const rec_cfg *cfg); 目的：启动录音 参数： base：创建的录音器实体； url：录音数据的存放地址； cfg：录音配置信息 返回值：返回状态
stop;	声明：int (*stop)(recorder_base *base); 目的：停止录音 参数： base：创建的录音器实体 返回值：返回状态

3 音频驱动描述

音频驱动可用来播放和录制 pcm 数据。音频驱动为应用层提供音频流操作接口和音频控制接口两类接口。

3.1 音频流接口说明

Table 3-1 音频流接口说明

function	detail
snd_pcm_init;	声明：int snd_pcm_init(void); 目的：AUDIO PCM 层初始化；系统上电初始化时调用； 参数：无 返回值：返回状态（0：成功；其它：失败）
snd_pcm_deinit;	声明：int snd_pcm_deinit(void); 目的：AUDIO PCM 层反初始化 参数：无 返回值：返回状态（0：成功；其它：失败）
snd_pcm_open;	声明： int snd_pcm_open(Snd_Card_Num card_num, Audio_Stream_Dir stream_dir, struct pcm_config *pcm_cfg); 目的：打开一个 PCM 流 参数：card_num：声卡号；stream_dir：pcm 数据流方向；pcm_cfg：录音或播放的配置； 返回值：返回状态（0：成功；其它：失败）
snd_pcm_close;	声明： int snd_pcm_close(Snd_Card_Num card_num, Audio_Stream_Dir stream_dir); 目的：关闭一个 PCM 流 参数：card_num：声卡号；stream_dir：pcm 数据流方向； 返回值：返回状态（0：成功；其它：失败）
snd_pcm_write;	声明： int snd_pcm_write(Snd_Card_Num card_num, void *data, uint32_t count); 目的：往对应声卡写入播放 pcm 数据 参数：card_num：声卡号；data：写入 pcm 数据的 buffer；count：写入

	pcm 数据的大小； 返回值：返回已写入的数据量长度；
snd_pcm_read;	声 明 ： int snd_pcm_read(Snd_Card_Num card_num, void *data, uint32_t count); 目的：读取对应声卡录音 pcm 数据 参数：card_num：声卡号；data：读取 pcm 数据的 buffer；count：读取 pcm 数据的大小； 返回值：返回读取到的数据量长度；
snd_pcm_flush;	声明：int snd_pcm_flush(Snd_Card_Num card_num); 目的：将音频驱动缓存区里的 pcm 数据冲刷到硬件，应用层播放完毕时调用 参数：card_num：声卡号 返回值：返回状态（0：成功；其它：失败）

3.2 音频控制接口说明

Table 3-2 音频控制接口说明

function	detail
audio_manager_init;	声明：int audio_manager_init(void); 目的：AUDIO Manager 层初始化；系统上电初始化时调用； 参数：无 返回值：返回状态（0：成功；其它：失败）
audio_manager_deinit;	声明：int audio_manager_deinit(void); 目的：AUDIO Manager 层反初始化 参数：无 返回值：返回状态（0：成功；其它：失败）
audio_manager_handler;	声 明 ： int audio_manager_handler(Snd_Card_Num card_num, Audio_Manager_Cmd cmd, Audio_Device dev, uint32_t param); 目的：AUDIO Manager 层音频控制接口，如设置音量、路径、Mute/Unmute 等 参数：card_num：声卡号；cmd：Audio_Manager_Cmd 类型命令；dev：

	AUDIO_Device 类型音频设备；param：命令参数； 返回值：返回状态（0：成功；其它：失败）
audio_maneger_ioctl;	声明： int audio_maneger_ioctl(Snd_Card_Num card_num, Codec_Ioctl_Cmd cmd, uint32_t cmd_param[], uint32_t cmd_param_len); 目的：AUDIO Manager 层 Ioctl 控制接口 参数：card_num：声卡号；cmd：Codec_Ioctl_Cmd 类型命令；cmd_param[]：命令参数数组指针；cmd_param_len: 命令参数数组长度； 返回值：返回状态（0：成功；其它：失败）
audio_manager_reg_read;	声明：int audio_manager_reg_read(Snd_Card_Num card_num, uint32_t reg); 目的：AUDIO Manager 层寄存器读接口封装，读取对应声卡的 codec 寄存器值 参数：card_num：声卡号；reg：寄存器地址； 返回值：返回读取到的寄存器值；
audio_manager_reg_write;	声明：int audio_manager_reg_write(Snd_Card_Num card_num, uint32_t reg, uint32_t val); 目的：AUDIO Manager 层寄存器写接口封装，往对应声卡的 codec 写入对应寄存器值 参数：card_num：声卡号；reg：寄存器地址；val：要写入的寄存器值； 返回值：返回状态（0：成功；其它：失败）
audio_manager_get_current_dev;	声明：int audio_manager_get_current_dev(Snd_Card_Num card_num); 目的：AUDIO Manager 层获取当前播放录音设备（不能获取驱动默认设备）； 参数：card_num：声卡号 返回值：当前播放录音设备 bit mask

4 使用说明

4.1 播放/录音功能选择示例

目前在 platform_init.c 里实现了弱函数 platform_cedarx_init，该函数列出了所有的录音功能的选择接口，且通过注释代码的方式默认只选择了部分播放功能。开发板初始化函数 platform_init 会调用到 platform_cedarx_init，而因为 platform_init.c 里的 platform_cedarx_init 函数被定义为一个弱函数，即如果外部没有定义实现函数 platform_cedarx_init，platform_init 会调用到 platform_init.c 里的弱函数 platform_cedarx_init；如果外部定义实现了函数 platform_cedarx_init，platform_init 就会去调用外部的 platform_cedarx_init。因此用户可在外部模块实现自己的 platform_cedarx_init。例如，可以在 main.c 中定义 platform_cedarx_init()如下：

```
#include “cedarx/cedarx.h”
void platform_cedarx_init(void)
{
    /* for media player */
    CedarxStreamListInit();
#if PRJCONF_NET_EN
    CedarxStreamRegisterHttps();
    CedarxStreamRegisterSsl();
    CedarxThreadStackSizeSet(DEMUX_THREAD, 8 * 1024);
    CedarxStreamRegisterHttp();
    CedarxStreamRegisterTcp();
#endif
    CedarxStreamRegisterFlash();
    CedarxStreamRegisterFile();
    CedarxStreamRegisterFifo();
    CedarxStreamRegisterCustomer();

    CedarxParserListInit();
    CedarxParserRegisterM3U();
    CedarxParserRegisterM4A();
    CedarxParserRegisterAAC();
    CedarxParserRegisterAMR();
    CedarxParserRegisterMP3();
    CedarxParserRegisterWAV();

    CedarxDecoderListInit();
    CedarxDecoderRegisterAAC();
    CedarxDecoderRegisterAMR();
    CedarxDecoderRegisterMP3();
    CedarxDecoderRegisterWAV();

    SoundStreamListInit();
    SoundStreamRegisterCard();
    SoundStreamRegisterReverb();

    /* for media recorder */
```

```
    CedarxWriterListInit();
    CedarxWriterRegisterFile();
    CedarxWriterRegisterCallback();
    CedarxWriterRegisterCustomer();

    CedarxMuxerListInit();
    CedarxMuxerRegisterAmr();
    CedarxMuxerRegisterPcm();

    CedarxEncoderListInit();
    CedarxEncoderRegisterAmr();
    CedarxEncoderRegisterPcm();
}
```

上述例子表示支持所有播放、录音功能。

4.2 播放/录音使用示例

播放使用请参考工程示例 example/audio_play；

录音使用请参考工程示例 example/audio_record；

同时录播请参考工程示例 example/audio_play_and_record