《WIFI Audio(F8198)嵌入式端公版接口说明》

版本号：V2.3

2018 年 6 月

目录

版本说明

版本号	日期	作者	修改内容
V1.8	2018-05-03	罗云进	修改文档格式
V1.9	2018-05-17	罗云进	新增 2.13.3 查询当前连接路由器的 SSID 2.13.4 获取设备信息
V1.9	2018-05-17	胡维	新增 2.9 音乐播放与控制接口
V2.0	2018-06-05	罗云进	修改 2.13.4 增加 DEVICEINFO 结构体成员 audioname（ssid）
V2.3	2018-06-011	罗云进	新增 2.1.6 设置升级状态标志位 更改 2.1.4 OTA 升级接口

1 概述

本文档所述所有接口中没有特殊依赖库相关说明的接口均在公用库 libcchip 中实现，使用时请在 Makefile 中添加编译选项:-lcchip,并在源码中添加头文件:#include <libcchip/libcchip.h> 。对于 CGI 程序调用改库中的接口请在进程初始化中调用 log_init(“l=00000”);以确保关掉所有打印调试信息。使用库 libcchip 时，为保证能通过编译,请确保 Makefile 的 DEPENDS 中包含：+alsa-lib +wdk +libpthread +librt +libmad +libcchip 这几个包 ，并确保依赖库列表中包含：-lasound -lmad -lcdb -lpthread -lrt -ldl -lm 这几个项。

本章描述内容如下表所述

标题	内容
1.1 API 函数参考域	介绍 API 函数参考域说明
1.2 数据类型参考域	介绍数据类型参考域

1.1 函数参考域

表 1-1 API 函数参考域说明

参考域	作用
描述	描述 API 的功能。
语法	显示 API 的语法样式。
参数	列出 API 的参数、参数说明及其属性。
返回值	列出 API 的返回值及其返回值说明。
需求	列出本 API 要包含的头文件和 API 编译时要链接的库文件。
注意	使用 API 时应注意的事项。
举例	使用 API 的实例。

1.2 数据类型参考域

表 1-2 数据类型参考域说明

参考域	作用
说明	简单描述结构体所实现的功能。
定义	列出结构体的定义。
成员	列出数据结构的成员及含义。
注意事项	列出使用数据类型时应注意的事项。
相关数据类型和接口	列出与本数据类型相关联的其他数据类型和接口。

2 API 参考

本章描述内容如下表所示

标题	内容
2.1 升级相关	包含 wifi、bt、conexant 升级
2.2 app 显示相关	App 上需要显示的信息
2.3 闹钟/定时功能	包括闹钟的创建、删除、查询
2.5 开机自动播放音乐列表功能	打开或关闭开机自动播放
2.6 定时关机功能接口	包括打开关闭功能
2.7 AmazonAlexa 接口	亚马逊相关信息：登录、语种等
2.8 multiroom 相关接口	多房间功能
2.9 音乐播放与控制	音乐播放控制功能：静音、声道设置等
2.10 录音相关	录音相关：开始、停止、获取等
2.11 蓝牙相关	获取蓝牙设备信息
2.12 设备网络连接配置	网络连接配置:连接断开 wifi
2.13 设备控制与参数设置	设备相关

2.1 升级相关

升级相关实现设备（wifi 和蓝牙）app 升级，网页升级。使用 app 对设备进行升级时，需先获取升级标志位，判断是否需要升级。升级过程中需要读取状态标志位，防止再次点击进行升级操作。该功能模块提供以下 API：

- xzxhtml_wifiup：网页升级 wifi
- xzxhtml_btup：网页升级蓝牙
- conexant_update：网页升级科胜讯
- app_update_noparmeter：OTA 升级接口
- get_wifi_bt_update_flage：获取升级时状态标志
- set_wifi_bt_update_flage 设置升级状态标志位
- get_wifi_bt_burn_progress: 获取升级时烧录进度

2.1.1 网页升级 wifi

xzxhtml_wifiup

【描述】
网页升级 wifi

【语法】
int xzxhtml_wifiup(char *pPath);

【参数】

参数名称（值）	描述	输入/输出
pPath	char *pPath wifi 固件先由 web 推送到设备，然后传入固件路径到函数	输入

【返回值】

返回值	描述
0	成功
-1	失败

【需求】
标准头文件

【注意事项】
网页升级 wifi 时，必须先由 web 推送固件至设备端，然后把固件所在路径作为参数。
升级时读取进度以及状态位详见 2.17/2.15

2.1.2 网页升级蓝牙

xzxhtml_btup

【描述】
网页升级蓝牙

【语法】
int xzxhtml_btup(char *pPath);

【参数】

参数名称（值）	描述	输入/输出
pPath	char *pPath bt 固件先由 web 推送到设备，然后传入固件路径到函数	输入

【返回值】

返回值	描述
0	成功
-1	失败

【需求】
标准头文件

【注意事项】
网页升级蓝牙时，必须先由 web 推送固件至设备端，然后把固件所在路径作为参数。
升级时读取进度以及状态位详见 2.17/2.15

2.1.3 网页升级科胜讯

conexant_update

【描述】
网页科胜讯升级

【语法】
int conexant_update(char *sfs,char *bin);

【参数】

参数名称（值）	描述	输入/输出
sfs	sfs 文件路径	输入
bin	bin 文件路径	输入

【返回值】

返回值	描述
0	成功
-1	失败

【需求】
标准头文件

【注意事项】
网页升级科胜讯时，必须先由 web 推送固件至设备端，然后把 sfs 和 bin 文件所在路径作为参数。

2.1.4 OTA 升级接口

app_update_noparmeter

【描述】
默认调用 API，参数为 NULL，使用手机 app 来升级 wifi 以及蓝牙设备，也可以带参数，把 URL 作为参数传入。
【语法】
int app_update_noparmeter(char *pPath);

【参数】

参数名称（值）	描述	输入/输出
NULL	默认传 NULL	输入
pPath	参数必须是有效的 URL，并且是要升级固件的当前目录，如：http://120.77.233.10/test_wifi/	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】
标准头文件

【注意事项】
升级 wifi 和蓝牙，由升级状态标志位决定（详见 2.16），升级过程中读取下载和烧录进度详见 2.17/2.18

2.1.5 获取升级状态标志位

get_wifi_bt_update_flage

【描述】
设备是否需要升级状态标志位

【语法】
int get_wifi_bt_update_flage(void);

【参数】

参数名称（值）	描述	输入/输出
无		

【返回值】

返回值	描述
0	wifi 和蓝牙都是最新，不需要升级
1	wifi 不需要升级，蓝牙需要升级
2	wifi 需要升级，蓝牙不需要升级
3	wifi 和蓝牙都需要升级
4	wifi 正在升级中
5	蓝牙正在升级中

6	wifi 固件下载中
7	蓝牙固件下载中
8	网页升级 wifi 中
9	网页升级蓝牙中

【需求】
标准头文件

【注意事项】
app 进行设备升级前，首先读取状态位，查看是否需要升级，升级过程中，查看状态位，确保设备不重复升级。

2.1.6 设置升级状态标志位

set_wifi_bt_update_flage

【描述】
网页升级下载前设置标志位

【语法】
int set_wifi_bt_update_flage(int status);

【参数】

参数名称（值）	描述	输入/输出
status	8：网页升级 wifi 中 9：网页升级蓝牙中，具体标志详看 2.1.5	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】
标准头文件

【注意事项】
网页升级 post 固件到设备时设置，防止重复升级

2.1.7 获取升级烧录进度

get_wifi_bt_burn_progress

【描述】
使用手机 app 来升级 wifi 以及蓝牙设备

【语法】
Int get_wifi_bt_burn_progress(void);

【参数】

参数名称（值）	描述	输入/输出
无		

【返回值】

返回值	描述
0~100	成功，返回进度
-1	失败

【需求】
标准头文件

【注意事项】
无论是网页升级还是 app 升级，都可以调用这个 API 获取升级烧录进度

2.1.8 获取升级下载进度

get_wifi_bt_download_progress

【描述】
使用手机 app 来升级 wifi 以及蓝牙设备

【语法】
Int get_wifi_bt_download_progress(void);

【参数】

参数名称（值）	描述	输入/输出
无		

【返回值】

返回值	描述
0~100	成功，返回进度
-1	失败

【需求】

标准头文件

【注意事项】
App 对设备进行升级时，调用这个 API 获取下载进度

2.2 app 显示相关

App 显示设备相关信息，提供以下 API：

- get_battery_level：电量显示
- is_charge_plug：是否插入电源标志
- get_conexant_version：科胜讯版本号

2.2.1 电量显示接口

get_battery_level

【描述】
App 显示电量接口

【语法】
int get_battery_level(void);

【参数】

参数名称（值）	描述	输入/输出
无		

【返回值】

返回值	描述
0-10	对应电量 0%-100%

【需求】
标准头文件

【注意事项】
App 获取有电池的设备电量时调用此接口，如果没有默认 0；

2.2.2 判断是否插入电源

is_charge_plug

【描述】

判断设备是否插入电源

【语法】
int is_charge_plug(void);

【参数】

参数名称（值）	描述	输入/输出
无		

【返回值】

返回值	描述
0	未插入电源返回整数 0
1	插入电源时返回整数 1
-1	读取失败返回-1

【需求】
标准头文件

【注意事项】
App 判断设备是否插入电源

2.2.3 app 显示 Conexant 版本号

get_conexant_version

【描述】
获取科胜讯版本号

【语法】
char *get_conexant_version(void);

【参数】

参数名称（值）	描述	输入/输出
无		

【返回值】

返回值	描述
字符串版本号	成功返回字符串版本号如 612001
NULL	读取失败返回 NULL

【需求】
标准头文件

【注意事项】
App 获取科胜讯版本号，如果没有科胜讯设备，显示 NULL,完成之后不需要释放 free

2.2.4 远场唤醒次数

get_wakeup_times

【描述】
获取远场唤醒次数

【语法】
int get_wakeup_times(void);

【参数】

参数名称（值）	描述	输入/输出
无		

【返回值】

返回值	描述
int 型数据	返回远场唤醒次数

【需求】
标准头文件

【注意事项】

2.2.5 设置提示音语言

set_language

【描述】
设置提示语音语言

【语法】
int set_language(int language);

【参数】

参数名称（值）	描述	输入/输出

0	为语音提示禁用	输入
1	汉语	输入
2	英语	输入
3	法语（暂不可用）	输入
4	西班牙语（暂不可用）	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】
标准头文件

【注意事项】

2.3 闹钟功能

闹钟定时功能，提供以下 API：

- create_music_alarm：创建闹钟
- delete_music_alarm：删除闹钟
- get_music_alarm_info：获取创建闹钟信息

2.3.1 创建闹钟接口

create_music_alarm

【描述】
创建闹钟

【语法】
int create_music_alarm(int NumID, int State, int Hour, int Min, int Wday, char *UrlAddr);

【参数】

参数名称（值）	描述	输入/输出
NumID	闹钟编号 1~9	输入
State	使能状态 0—Disable 1--Enable	输入
Hour	小时	输入
Min	分钟	输入
Wday	星期(二进制从低到高位表示周一到周日用十六进制转换)	输入
UrlAddr	混合列表路径(/tmp/PlayMixedListJson.json)包含单曲或专	输入

	辑或混合的单曲和专辑列表	

【返回值】

返回值	描述
0	成功
-1	失败

【需求】
标准头文件

【注意事项】

2.3.2 删除闹钟接口

delete_music_alarm

【描述】
删除闹钟

【语法】
int delete_music_alarm(int NumID);

【参数】

参数名称（值）	描述	输入/输出
NumID	闹钟编号 1~9	输入

【返回值】

返回值	描述
0	成功
-1	失败

【需求】
标准头文件

【注意事项】

2.3.3 查询闹钟接口

get_music_alarm_info

【描述】
获取创建闹钟信息

【语法】
char *get_music_alarm_info(void);

【参数】

参数名称（值）	描述	输入/输出
无		

【返回值】

返回值	描述
查询地址	成功返回查询地址，字符长度<64 Byte
NULL	失败

【需求】
标准头文件

【注意事项】
用完注意 free

【举例】

查询闹钟信息 json 文件位置: usr/script/playlists/alarm_list.json

Json 数据格式：

```
{
    "tasknum": 0,                                   //保留字段
    "musicnum": 5 ,                                 //闹钟总数
    "alam": [   ],                                  //保留字段
    "music": [
        {
            "num": 1,                               //专辑播放编号
            "state": 1,                             //专辑播放使能
            “time”:”8:30:1f”                        //专辑播放时间/重复星期
            “url”:”http://ximalaya.music.com”       //专辑 URL 地址
            “platfrom”:”ximalaya”                   //平台名 ximalaya

        },
        {
            "num": 2,                               //专辑播放编号
            "state": 1,                             //专辑播放使能
            “time”:”8:30:1f”                        //专辑播放时间/重复星期
            “url”:” /usr/script/playlists/timermusicinfo02.json”   //播放列表 JSON 文件
        },
```

```
        {
            "num": 3,                                   //专辑播放编号
            "state": 1,                                 //专辑播放使能
            “time”:”8:30:1f”                            //专辑播放时间/重复星期
            “url”:” /usr/script/playlists/timermusicinfo03.json”   //播放列表 JSON 文件
        },
        {
            "num": 4,                                   //专辑播放编号
            "state": 1,                                 //专辑播放使能
            “time”:”8:30:1f”                            //专辑播放时间/重复星期
            “url”:” http://ximalaya.music.com”          //专辑 URL 地址
            “platfrom”:”ximalaya”                       //平台名 ximalaya

        },
        {
            "num": 5,                                   //混合音乐播放编号
            "state": 1,                                 //混合音乐播放使能
            “time”:”13:27:1”                            //混合音乐播放时间/重复星期
            “url”:” /usr/script/playlists/PlayMixedListJson05.json” //混合音乐播放列表
JSON 文件
        },

    ]
}
```

2.4 收藏歌曲列表相关

收藏歌曲列表提供以下 API：

- add_collectionSong：收藏歌曲 URL
- del_collectionSong：删除收藏歌曲
- inquiry_collectionSong：查询 1 至 9 快捷键中收藏的歌曲

2.4.1 添加收藏歌曲列表

add_collectionSong

【描述】
收藏歌曲 URL。

【语法】

```
int add_collectionSong(int keyNum);
```

【参数】

参数名称（值）	描述	输入/输出
keyNum	1~9 ， 1 至 9 个列表	输入

【返回值】

返回值	描述
0	成功
-1	失败

【需求】
标准头文件

【注意事项】
将“/tmp/CollectionListJson.json”中的 json 格式数据整理歌曲。

【举例】

2.4.2 删除收藏歌曲列表

del_collectionSong

【描述】
删除收藏歌曲。

【语法】
int del_collectionSong(int keyNum);

【参数】

参数名称（值）	描述	输入/输出
keyNum	1~9 ， 1 至 9 个列表	输入

【返回值】

返回值	描述
0	成功
-1	失败

【需求】
标准头文件

【注意事项】

删除收藏在 1 至 9 快捷键文件中歌曲。

【举例】

2.4.3 查询收藏歌曲列表

inquiry_collectionSong

【描述】
查询 1 至 9 快捷键中收藏的歌曲。

【语法】
int inquiry_collectionSong(int keyNum,char **pjsonstring);

【参数】

参数名称（值）	描述	输入/输出
keyNum	1~9 ， 1 至 9 个列表	输入
pjsonstring	用于存放堆空间地址，需要 free	输出

【返回值】

返回值	描述
0	成功
-1	失败

【需求】
标准头文件

【注意事项】
查看收藏在（1~9）文件中歌曲。

【举例】

```
char *pjsonstring;
inquiry_collectionSong(1,&pjsonstring);   //数据存放在 pjsonstring 中
.....
free(pjsonstring); //释放空间
```

说明：
1> 查询收藏歌曲列表如何使用？
答：调用 API（inquiry_collectionSong），返回 0，成功；-1，失败。
ret=inquiry_collectionSong(1,&pjsonstring);
字符串指针 pjsonstring 中存放着堆空间地址，堆空间地址指向查询数据。格式就是

CollectionListJson.json 文件中的格式，使用完需要释放它。

2>. CollectionListJson.json 文件数据是怎样的格式？是怎么用的？

答：

postplaylist 将获取到的歌曲链接数据整理成如下数据格式,放到嵌入式路径文件：“/tmp/CollectionListJson.json”中，然后调用 API(add_collectionSong(int keyNum)),添加就完成了。

重要 json 成员：

字符串	描述
CollectionNum	文件存放在第 KeyNum 个收藏列表中
num	Mixedlist 数据成员数量
type	元素类型 1：音乐 2：专辑
content_url	歌曲链接 或 专辑链接

数据格式：


```
{
        "ret":  "OK",
        "CollectionNum":  1, //文件在第 KeyNum 个列表中
        "content":  {
                "num":  3,
                "mixedlist":  [
                        {
                                "type":  1,     //元素类型 1：音乐 2：专辑
                                "content":  {
                                        "index":  1,
                                        "title":  "手放开——MCMK",
                                        "artist":  "MCMK",
                                        "album":  "爱",
                                        "content_url":  "http://fdfs.xmcdn.com/group3/M03/05/22/wKgDsVIHx2bB393DABAivzur0J0919.mp3",
                                        "cover_url":  "http://fdfs.xmcdn.com/group3/M00/04/D0/wKgDslIHxznz2G4kAACAiIwe-z4202_mobile_large.jpg",
                                        "platform":  "ximalaya",
                                        "column":  12345,
                                        "program":  67890
                                }
                        },
                        {
                                "type":  1,
                                "content":  {
                                        "index":  2,
                                        "title":  "手放开——MCMK",
                                        "artist":  "MCMK",
```

```
                                        "album":  "爱",
                                        "content_url":  "http://fdfs.xmc
    dn.com/group3/M03/05/22/wKgDsVIHx2bB393DABAivzur0J0919.mp3",
                                        "cover_url":  "http://fdfs.xmcdn.
    com/group3/M00/04/D0/wKgDslIHxznz2G4kAACAiIwe-z4202_mobile_large.jpg",
                                        "platform":  "ximalaya",
                                        "column":  12345,
                                        "program":  67890
                                }
                        },
                        {
                                "type":  2,
                                "content":  {
                                        "index":  3,
                                        "title":  "爱",
                                        "artist":  "MCMK",
                                        "album":  "爱",
                                        "content_url":  "http://3rd.xima
    laya.com/albums/200410/tracks?i_am=test20150518&page=1&per_page=20&is",
                                        "cover_url":  "http://fdfs.xmcdn.
    com/group3/M01/05/2A/wKgDsVIHzrDxzauCAACAiIwe-z4488_mobile_large.jpg",
                                        "platform":  "ximalaya",
                                        "column":  12345,
                                        "program":  67890
                                }
                        }
                ]
        }
    }
```


2.5 开机自动播放音乐列表功能

开机自动播放音乐列表功能，提供以下 API：

- startingup_play_state：打开或关闭开机自动播放

2.5.1 打开/关闭开机自动播放音乐接口

startingup_play_state

【描述】

在连接外网正常下打开此功能音箱在开机时会从服务器上获取 APP 收藏音乐来自动播放。

【语法】
int startingup_play_state(int State);

【参数】

参数名称（值）	描述	输入/输出
State	0 关闭开机播放音乐;1 打开开机播放音乐	输入

【返回值】

返回值	描述
0	成功
-1	失败

【需求】
标准头文件

【注意事项】

【举例】

2.6 定时关机功能接口

定时关机功能接口，提供以下 API：

- enable_shutdown_set：打开定时关机功能
- disable_shutdown_set：关闭定时关机功能
- get_shutdown_remain_time：获取定时关机剩余时间

2.6.1 打开定时关机接口

enable_shutdown_set

【描述】
打开定时关机功能。

【语法】
int enable_shutdown_set(int Time);;

【参数】

参数名称（值）	描述	输入/输出
Time	1~3600s(单位:秒)	输入

【返回值】

返回值	描述

0	成功
-1	失败

【需求】
标准头文件

【注意事项】

【举例】

2.6.2 关闭定时关机接口

disable_shutdown_set

【描述】
打开定时关机功能。

【语法】
int disable_shutdown_set(void);

【参数】

参数名称（值）	描述	输入/输出
无		

【返回值】

返回值	描述
0	成功
-1	失败

【需求】
标准头文件

【注意事项】

【举例】

2.6.3 获取定时关机剩余时间接口

get_shutdown_remain_time

【描述】

获取定时关机剩余时间。

【语法】
int get_shutdown_remain_time(void);

【参数】

参数名称（值）	描述	输入/输出
无		

【返回值】

返回值	描述
0~3600	成功
-1	失败

【需求】
标准头文件

【注意事项】

【举例】

2.7 AmazonAlexa 接口

说明：关于 AmazonAlexa 接口需要依赖 libavsclient 库并包含#include<avsclient/libavsclient.h>头文件。
LOGIN_INFO_STRUCT 结构体：

```
typedef struct LOGIN_INFO{
  char *pCodeVerifier;
  char *pCodeChallenge;
  char *pCodeChallengeMethod;
  char *pProductId;              // 产品 ID
  char *pDeviceSerialNumber; // 相当于之前的 dsn
}LOGIN_INFO_STRUCT;
```

AmazonAlexa 提供以下 API：

- logInInfo_new：创建 amazon alexa 登陆时需要的参数
- OnSendSignInInfo：APP 登录成功后，将一些音箱登录亚马逊所需要的信息推送给音箱
- OnGetAVSSigninStatus：查询亚马逊登录状态
- OnSignUPAmazonAlexa：退出亚马逊登录
- OnSetAlexaLanguage：设置亚马逊语种
- OnGetAlexaLanguage：查询亚马逊语种

2.7.1 获取 AmazonAlexa 登录参数

logInInfo_new

【描述】
创建 amazon alexa 登陆时需要的参数。

【语法】
LOGIN_INFO_STRUCT* logInInfo_new();

【参数】

参数名称（值）	描述	输入/输出
无		

【返回值】

返回值	描述
	返回 amazon alexa 登陆时需要的参数

【需求】
库文件：libavsclient 库
头文件：#include<avsclient/libavsclient.h>
【注意事项】

【举例】

```
LOGIN_INFO_STRUCT* loginfo;
loginfo = logInInfo_new();
if (loginfo != NULL){
    printf("pCodeChallengeMethod:%s\n", loginfo->pCodeChallengeMethod);
    printf("pCodeVerifier:%s\n", loginfo->pCodeVerifier);
    printf("pCodeChallenge:%s\n", loginfo->pCodeChallenge);
    printf("pProductId:%s\n", loginfo->pProductId);
    printf("pDeviceSerialNumber:%s\n", loginfo->pDeviceSerialNumber);
    logInInfo_free(loginfo);
}
```

2.7.2 推送亚马逊登录信息

OnSendSignInInfo

【描述】
APP 登录成功后，将一些音箱登录亚马逊所需要的信息推送给音箱。

【语法】

int OnSendSignInInfo(char *codeVerifier, char *authorizationCode, char *redirectUri, char *clientId);

【参数】

参数名称（值）	描述	输入/输出
codeVerifier		输入
authorizationCode		输入
redirectUri		输入
clientId		输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

库文件：libavsclient 库

头文件：#include<avsclient/libavsclient.h>

【注意事项】

说明：上一版本推送亚马逊登录信息时只推送了 authorizationCode, redirectUri, clientId 三个参数，现在由于亚马逊登陆参数是直接在 postplaylist 里面生产获取的，需要 APP 增加 codeVerifier 参数的发送。

2.7.3 查询亚马逊登陆状态

OnGetAVSSigninStatus

【描述】

查询亚马逊登录状态。

【语法】

int OnGetAVSSigninStatus();

【参数】

参数名称(值)	描述	输入/输出
无		

【返回值】

返回值	描述
0	登陆失败或未登陆

1	成功
2	登陆中

【需求】
库文件：libavsclient 库
头文件：#include<avsclient/libavsclient.h>

【注意事项】

2.7.4 退出亚马逊登陆

OnSignUPAmazonAlexa

【描述】
退出亚马逊登录。

【语法】
Int OnSignUPAmazonAlexa(void);

【参数】

参数名称(值)	描述	输入/输出
无		

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】
库文件：libavsclient 库
头文件：#include<avsclient/libavsclient.h>

【注意事项】

2.7.5 设置亚马逊语种

OnSetAlexaLanguage

【描述】
设置亚马逊语种。

【语法】

```
int OnSetAlexaLanguage(char *pData);
```

【参数】

参数名称(值)	描述	输入/输出
pData	目前支持的参数有：en-AU, en-CA, en-GB, en-IN, en-US, de-DE, ja-JP	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

库文件：libavsclient 库

头文件：#include<avsclient/libavsclient.h>

【注意事项】

2.7.6 查询亚马逊语种

OnGetAlexaLanguage

【描述】

设置亚马逊语种。

【语法】

```
char *OnGetAlexaLanguage();
```

【参数】

参数名称(值)	描述	输入/输出
无		

【返回值】

返回值	描述
具体语种	成功返回具体的语种
NULL	失败

【需求】

库文件：libavsclient 库

头文件：#include<avsclient/libavsclient.h>

【注意事项】

2.8 multiroom 相关接口

2.8.1 调节指定从音箱音量

示例如下:
snapmulti.sh setVolume "00:11:22:33:01:04" "10"
snapmulti.sh 调用的脚本
setVolume 设置音量的命令参数
00:11:22:33:01:04 从音箱的 mac 地址
"10" 设置音箱值

2.8.2 断开指定从音箱

示例如下:
snapmulti.sh disconnect "00:11:22:33:01:04"
snapmulti.sh 调用的脚本
disconnect 断开从音箱的命令参数
00:11:22:33:01:04 从音箱的 mac 地址

2.8.3 设置指定音箱的声道

示例如下:
snapmulti.sh setChannel "00:11:22:33:01:04" "1" //1 左声道 2 右声道
snapmulti.sh 调用的脚本
setChannel 设置从音箱声道的命令参数
00:11:22:33:01:04 从音箱的 mac 地址
"1" //1 左声道 2 右声道

2.9 音乐播放与控制

PLAYER_STATUS 数据结构：

```
typedef struct{
    int mainmode;       // 0: 普通音箱 1: 主音箱 2:子音箱
    int nodetype;       // 声道， 0 表示立体声， 1 表示左声道， 2 表示右声道
    int sw;             // 播放列表中的歌曲切换模式： 0：顺序播放 1：循环单曲 2：
```

随机播放 3：列表循环

```
    State status;       // 当前播放状态 0:UNK 1:STOP 2:PLAY 3:PAUSE
    int curpos;         // 当前的位置
    int totlen;             // 总长度
    char* Title;        // 标题
    char* Artist;       // 艺术家
    char* Album;        // 专辑名
    int Year;           // 年份
    int Track;          // 轨道号
    char* Genre;        // 风格
    int TFflag;         // 是否插入 T 卡
    int plicount;       // 播放列表的总条数
    int plicurr;        // 当前音轨在播放列表中的 index
    int vol;            // 当前音量
    int mute;           // 当前是否静音
    char* uri;          // 当前 dlna 的 uri
    char* cover;        // 封面
    int sourcemode;     // 输入源模式： 0 表示 wifi 1 表示蓝牙 2 表示外接播放模式
    char* uuid;         // 设备 uuid
}PLAYER_STATUS;
```

音乐播放控制提供以下 API：

- MediaPlayerAllStatus: 获取当前设备状态
- MediaPlayerAllStatusFreeSturct:释放数据结构
- MediaPlayerPlay:开始播放
- MediaPlayerPause:暂停播放
- MediaPlayerStop:停止播放
- MediaPlayerNext:下一曲
- MediaPlayerPrevious:上一曲
- MediaPlayerSeek(int seconds);快进/快退
- MediaPlayerSetVolume(int iVolume);设置音量
- MediaPlayerGetVolume:查询音量
- MediaPlayerSetMute(bool mute);静音设置
- MediaPlayerMuteStatus:获取静音状态
- MediaPlayerSetPlayMode(int mode);设置播放模式
- MediaPlayerGetPlayMode:查询播放模式
- MediaPlayerSetEqualizerMode(int mode);设置均衡器模式
- MediaPlayerGetEqualizerMode:查询均衡器模式
- MediaPlayerSetChannel(int iChannel);声道设置
- MediaPlayerGetChannel:查询声道设置

2.9.1 获取当前设备播放状态

MediaPlayerAllStatus

【描述】
用户获取当前设备的播放状态，返回 PLAYER_STATUS 数据结构

【语法】
PLAYER_STATUS* MediaPlayerAllStatus();

【参数】

参数名称(值)	描述	输入/输出
无		

【返回值】

返回值	描述
PLAYER_STATUS*	返回当前设备的播放状态
NULL	获取失败

【需求】

【注意事项】
返回值需要使用 MediaPlayerAllStatusFreeSturct 接口释放。

【举例】

```
PLAYER_STATUS *playerStatus    = MediaPlayerAllStatus();
if (playerStatus != NULL) {
      MediaPlayerAllStatusFreeSturct(playerStatus);
}
```

2.9.2 释放播放状态数据结构

MediaPlayerAllStatusFreeSturct
【描述】
释放用户获取设备播放状态时产生的数据结构

【语法】
void MediaPlayerAllStatusFreeSturct(PLAYER_STATUS* playerStatus);

【参数】

参数名称(值)	描述	输入/输出
playerStatus	需要释放的数据结构	输入

【返回值】

返回值	描述
无	

【需求】

【注意事项】

【举例】

2.9.3 开始播放

MediaPlayerPlay

【描述】
开始播放音乐

【语法】
bool MediaPlayerPlay();

【参数】

参数名称(值)	描述	输入/输出
无		

【返回值】

返回值	描述
true	成功
false	失败

【需求】

【注意事项】

2.9.4 暂停播放

MediaPlayerPause

【描述】
暂停当前音乐播放

【语法】
bool MediaPlayerPause();

【参数】

参数名称(值)	描述	输入/输出
无		

【返回值】

返回值	描述
true	成功
false	失败

【需求】

【注意事项】

2.9.5 停止播放

MediaPlayerStop

【描述】
停止当前音乐播放

【语法】
bool MediaPlayerStop();

【参数】

参数名称(值)	描述	输入/输出
无		

【返回值】

返回值	描述
true	成功
false	失败

【需求】

【注意事项】
如果调用停止播放接口再调用开始播放接口的话音乐会从头开始播放

2.9.6 下一曲

MediaPlayerNext

【描述】
播放下一曲音乐

【语法】
bool MediaPlayerNext();

【参数】

参数名称(值)	描述	输入/输出
无		

【返回值】

返回值	描述
true	成功
false	失败

【需求】

【注意事项】

2.9.7 上一曲

MediaPlayerPrevious

【描述】
播放上一曲音乐

【语法】
bool MediaPlayerPrevious();

【参数】

参数名称(值)	描述	输入/输出
无		

【返回值】

返回值	描述
true	成功
false	失败

【需求】

【注意事项】

2.9.8 快进/快退

MediaPlayerSeek

【描述】
用于控制播放音乐的进度条，单位为秒

【语法】
bool MediaPlayerSeek(int iSecond);

【参数】

参数名称(值)	描述	输入/输出
iSecond	移动到指定的音频偏移量	输入

【返回值】

返回值	描述
true	成功
false	失败

【需求】

【注意事项】

2.9.9 设置音量

MediaPlayerSetVolume

【描述】
设置播放器播放音量

【语法】
bool MediaPlayerSetVolume(int iVolume);

【参数】

参数名称(值)	描述	输入/输出
iVolume	需要设置的音量值 0-100	

【返回值】

返回值	描述
true	成功
false	失败

【需求】

【注意事项】

2.9.10 获取音量

MediaPlayerGetVolume

【描述】
获取播放器的播放音量

【语法】
int MediaPlayerGetVolume();

【参数】

参数名称(值)	描述	输入/输出
无		

【返回值】

返回值	描述
iRet	具体的音量等级 0-100

【需求】

【注意事项】

2.9.11 静音设置

MediaPlayerSetMute

【描述】
设置播放器是否静音

【语法】
bool MediaPlayerSetMute(bool mute);

【参数】

参数名称(值)	描述	输入/输出
mute	True 为静音，false 为取消静音	输入

【返回值】

返回值	描述
true	成功

false	失败

【需求】

【注意事项】

2.9.12 获取静音状态

MediaPlayerMuteStatus

【描述】
获取播放器的当前是否静音

【语法】
bool MediaPlayerMuteStatus();

【参数】

参数名称(值)	描述	输入/输出
无		

【返回值】

返回值	描述
Bool	true 为静音状态 false 为非静音状态

【需求】

【注意事项】

2.9.13 设置播放模式

MediaPlayerSetPlayMode

【描述】
设置播放器是否静音

【语法】
bool MediaPlayerSetPlayMode(int mode);

【参数】

参数名称(值)	描述	输入/输出
mode	0 列表循环，1 个单循环，2 个随机播放，3 个列表播放	输入

【返回值】

返回值	描述
true	成功
false	失败

【需求】

【注意事项】

2.9.14 查询播放模式

MediaPlayerGetPlayMode

【描述】
查询播放器的当前的播放模式，0 列表循环，1 个单循环，2 个随机播放，3 个列表播放

【语法】
int MediaPlayerGetPlayMode();

【参数】

参数名称(值)	描述	输入/输出
无		

【返回值】

返回值	描述
iMode	0 列表循环，1 个单循环，2 个随机播放，3 个列表播放

【需求】

【注意事项】

2.9.15 设置均衡器模式

MediaPlayerSetEqualizerMode

【描述】
设置均衡器模式。

【语法】
bool MediaPlayerSetEqualizerMode(int mode);

【参数】

参数名称(值)	描述	输入/输出
0	关闭均衡	输入
1	Classic 模式	输入
2	Popular 模式	输入
3	Jazzy 模式	输入
4	Vocal 模式	输入

【返回值】

返回值	描述
true	成功
false	失败

【需求】

【注意事项】

2.9.16 查询均衡器

MediaPlayerGetEqualizerMode

【描述】
查询均衡器模式。

【语法】
int MediaPlayerGetEqualizerMode();

【参数】

参数名称(值)	描述	输入/输出
无		

【返回值】

返回值	描述
0	均衡器模式
1	均衡器模式
2	均衡器模式
3	均衡器模式
4	均衡器模式

【需求】

【注意事项】

2.9.17 声道设置

MediaPlayerSetChannel

【描述】
声道设置。

【语法】
bool MediaPlayerSetChannel(int iChannel);

【参数】

参数名称(值)	描述	输入/输出
0	正常双声道	输入
1	左声道	输入
2	右声道	输入

【返回值】

返回值	描述
true	成功
false	失败

【需求】

【注意事项】

2.10 录音相关

录音相关提供以下 API：

- set_voice_channel：设置录音时间
- play_record_audio：播放录音文件
- play_url_audio：临时播放录音
- start_record：开始录音
- stop_record：停止录音
- start_fixed_record：定长录音
- get_record_file_URL：获取录音文件的 URL
- voice_activity_detection：语音活动检测
- voice_channel_compare：左右声道录音对比
- dot_normal_record：开始正常模式录音
- dot_normal_record_web_path：获取正常模式录音文件路径
- dot_mic_record：麦克模式录音
- dot_ref_record：获取麦克模式录音文件路径
- dot_ref_record：参考模式录音
- dot_ref_record_web_path：参考模式录音路径

- dot_stop_record：停止麦克模式录音

2.10.1 设置录音时间

set_record_time

【描述】
录制指定时间长度的音频，最长 12s。

【语法】
int set_record_time(int value);

【参数】

参数名称(值)	描述	输入/输出
value	录音时间	输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

【注意事项】

2.10.2 播放录音文件

play_record_audio

【描述】
播放录音文件。

【语法】
int play_record_audio(void);

【参数】

参数名称(值)	描述	输入/输出
无		

【返回值】

返回值	描述

0	成功
非 0	失败

【需求】

【注意事项】

2.10.3 临时播放录音

play_url_audio

【描述】
临时播放录音文件，播放完之后恢复先前播放,URL 为录音文件的访问地址，录音文件格式为 mp3。

【语法】
int play_url_audio(const char *url);

【参数】

参数名称(值)	描述	输入/输出
url		输入

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

【注意事项】

2.10.4 开始录音

start_record

【描述】
开始录音,默认最长录音 12s。

【语法】
int start_record(void);

【参数】

参数名称(值)	描述	输入/输出
无		

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

【注意事项】

2.10.5 停止录音

stop_record

【描述】
立即停止录音。

【语法】
int stop_record(void);

【参数】

参数名称(值)	描述	输入/输出
无		

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

【注意事项】

2.10.6 定长录音

start_fixed_record

【描述】
录制 5s 音频。

【语法】
int start_fixed_record(void);

【参数】

参数名称(值)	描述	输入/输出
无		

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

【注意事项】

2.10.7 获取录音文件 URL

get_record_file_URL

【描述】
获取录音文件的 URL，可用此 URL 进行下载。

【语法】
const char *get_record_file_URL(void);

【参数】

参数名称(值)	描述	输入/输出
无		

【返回值】

返回值	描述
URL	成功返回录音文件的 URL
NULL	失败

【需求】

【注意事项】
使用完不需要 free

2.10.8 语音活动检测

voice_activity_detection

【描述】
语音活动检测。

【语法】
int voice_activity_detection(int value, char *pcardname);

【参数】

参数名称(值)	描述	输入/输出
Value	取值范围为 500000 到 2000000，值越大相应的检测是否有语音活动的范围越大	输入
pcardname	要测试的声卡名称，支持 default、plughw:1 和 plughw:0	输入

【返回值】

返回值	描述
0	成功
NULL	失败

【需求】

【注意事项】

2.10.9 左右声道录音对比

voice_channel_compare

【描述】
在录音成功之后，可调用该接口对比两个 Mic 的录音效果会否接近。

【语法】
void voice_channel_compare(char *filename, float *l_pro, float *r_pro);

【参数】

参数名称(值)	描述	输入/输出
filename	双声道录音文件绝对路径	输入

l_pro	浮点型结果，存放比较结果	输出
r_pro	浮点型结果，存放比较结果	输出

【返回值】

返回值	描述
无	

【需求】

【注意事项】

2.10.10 正常录音

dot_normal_record

【描述】
开始正常模式录音。

【语法】
int dot_normal_record(void);

【参数】

参数名称(值)	描述	输入/输出
无		

【返回值】

返回值	描述
0	成功
-1	失败

【需求】

【注意事项】

2.10.11 获取正常模式录音文件 URL

dot_normal_record_web_path

【描述】
获取正常模式录音文件路径。

【语法】
char* dot_normal_record_web_path(void);

【参数】

参数名称（值）	描述	输入/输出
无		

【返回值】

返回值	描述
路径	成功返回正常模式录音文件在 web server 的路径

【需求】

【注意事项】
使用完不需要 free

2.10.12 麦克录音

dot_mic_record

【描述】
开始麦克模式录音。

【语法】
int dot_mic_record(void);

【参数】

参数名称（值）	描述	输入/输出
无		

【返回值】

返回值	描述
0	成功
-1	失败

【需求】

【注意事项】

2.10.13 获取麦克录音文件 URL

dot_mic_record_web_path

【描述】
获取麦克模式录音文件路径。

【语法】
char* dot_mic_record_web_path(void);

【参数】

参数名称（值）	描述	输入/输出
无		

【返回值】

返回值	描述
路径	成功返回麦克模式录音文件在 web server 的路径

【需求】

【注意事项】

2.10.14 参考信号录音

dot_ref_record

【描述】
开始参考模式录音。

【语法】
int dot_ref_record(void);

【参数】

参数名称（值）	描述	输入/输出
无		

【返回值】

返回值	描述
0	成功

-1	失败

【需求】

【注意事项】

2.10.15 获取参考信号录音文件 URL

dot_ref_record_web_path

【描述】
获取录音文件 url。

【语法】
char* dot_ref_record_web_path(void);

【参数】

参数名称(值)	描述	输入/输出
无		

【返回值】

返回值	描述
路径	成功返回参考模式录音文件在 web server 的路径

【需求】

【注意事项】
用完不需要 free

2.10.16 停止模式录音

dot_stop_record

【描述】
停止麦克模式录音。

【语法】
int dot_stop_record();

【参数】

参数名称(值)	描述	输入/输出
无		

【返回值】

返回值	描述
0	成功
-1	失败

【需求】

【注意事项】

2.11 蓝牙相关

2.11.1 搜索周边蓝牙设备

bt_search

【描述】
开始搜索周边蓝牙设备。

【语法】
int bt_search(void);

【参数】

参数名称(值)	描述	输入/输出
无		

【返回值】

返回值	描述
0	成功
-1	失败

【需求】

【注意事项】

2.11.2 获取蓝牙设备信息

bt_list

【描述】
获取蓝牙设备信息列表。

【语法】
int bt_list(char **bt_list);

【参数】

参数名称（值）	描述	输入/输出
bt_list	指向获取到的蓝牙设备信息列表,该指针用完后需要 free bt_list 字符串中每个条目格式为 XX+112233445566,XX+NAME,XX 为序号(00 01 02..),后面是蓝牙地址，序号第 1 位为 1 表示该条目对应的蓝牙设备为当前连接的设备	输出

【返回值】

返回值	描述
0	成功
-1	失败

【需求】

【注意事项】

2.11.3 连接蓝牙设备

bt_connect

【描述】
连接蓝牙设备。

【语法】
int bt_connect(char *bt_addr);

【参数】

参数名称（值）	描述	输入/输出
bt_addr	需要连接的蓝牙设备地址	输入

【返回值】

返回值	描述
0	成功
-1	失败

【需求】

【注意事项】

2.11.4 断开蓝牙设备

bt_disconnect

【描述】
断开蓝牙设备。

【语法】
int bt_disconnect();

【参数】

参数名称(值)	描述	输入/输出
无		

【返回值】

返回值	描述
0	成功
-1	失败

【需求】

【注意事项】

2.12 设备网络连接配置

2.12.1 获取当前环境下 wifi 的 SSID

get_aplist

【描述】

获取当前环境下所有路由器的信息，包括 ssid，信道等。

【语法】

int get_aplist(char **pAplist);

【参数】

参数名称（值）	描述	输入/输出
pAplist	apInfList **pAplist, 指向获取到的 wifi SSID 信息，SSID 信息以结构体数组的形式表示，该指针用完后需要 free(*pAplist)	输出

【返回值】

返回值	描述
返回获取到的 AP 个数	成功
-2	失败

【需求】

【注意事项】

参数 apInfList **pAplist 中的结构体：

```
typedef struct apInfList{
        char SSID[16];      //Wifi 可见名称
        char bssid[18];     //Mac 地址
        char rssi[4];       //信号强度
        char channel;       //Wifi 信道
        char auth;          //是否加密
        char encry;         //加密类型
        char extch;
    }apInfList;
```

示例：

```
int count=0, i = 0;
apInfList *aplist = NULL;
count = get_aplist(&aplist);
for(i = 0; i < count; i++){
    printf("aplist[%d]:%s\n", i, aplist[i].SSID);    //打印所有获取到的 WIFI SSID
}
free(aplist);
```

2.12.2 连接 wifi

connect_wifi

【描述】
连接相应的 wifi。

【语法】
int connect_wifi(char* ssid,char* psw);

【参数】

参数名称(值)	描述	输入/输出
ssid	ssid 字符串	输入
psw	密码字符串	输入

【返回值】

返回值	描述
0	成功
-1	失败

【需求】

【注意事项】

2.12.3 断开 wifi

disconnect_wifi

【描述】
断开网络连接。

【语法】
int disconnect_wifi(void);

【参数】

参数名称(值)	描述	输入/输出
无		

【返回值】

返回值	描述

0	成功
-1	失败

【需求】

【注意事项】

2.13 设备控制与参数设置

2.13.1 修改设备 ssid 名称

set_device_ssid

【描述】
修改本模块的 AP 模式下的 SSID 名称。

【语法】
int set_device_ssid(char *ssid);

【参数】

参数名称(值)	描述	输入/输出
ssid	目的 SSID 名称	输入

【返回值】

返回值	描述
0	成功
-1 或者-2	失败

【需求】

【注意事项】

2.13.2 设置设备 wifi 密码

set_device_pwd

【描述】

修改本模块的 AP 模式下的 wifi 密码。

【语法】
int set_device_pwd(char *pwd);

【参数】

参数名称(值)	描述	输入/输出
pwd	密码字符串	输入

【返回值】

返回值	描述
0	成功
-1 或者-2	失败

【需求】

【注意事项】

2.13.3 查询当前连接路由器的 SSID

search_connect_router_ssid

【描述】
查询当前连接的路由器。

【语法】
char * search_connect_router_ssid(void);

【参数】

参数名称(值)	描述	输入/输出
无		

【返回值】

返回值	描述
ssid 字符串	成功，例如返回 alink_test
NULL	失败

【需求】

【注意事项】

2.13.4 获取设备信息

get_device_info

【描述】
获取设备信息。

【语法】
int get_device_info(DEVICEINFO * aplist);

【参数】

参数名称（值）	描述	输入/输出
aplist	DEVICEINFO *aplist, 指向获取到的设备信息，设备信息以结构体数组的形式表示，该指针用完后需要 free(aplist)	输出

【返回值】

返回值	描述
0	成功
非 0	失败

【需求】

【注意事项】
最好按以下示例：

```
DEVICEINFO *aplist = NULL;
aplist = (DEVICEINFO *)calloc(1, sizeof(DEVICEINFO));
get_device_info(aplist);
```

【举例】
设备信息结构体为

```
typedef struct DEVICEINFO{
//    struct apInfList *next;
      char sys_language[8];        //系统语言
      char sup_language[64];     //支持语言
      char firmware[64];         //固件系列
      char release[32];          //发行版本号
      char mac_addr[32];         //设备 mac 地址
      char uuid[64];             //设备 uuid
      char project[64];          //项目名称
      char battery_level[2];          //电池电量
      char function_support[64];    //功能支持
      char bt_ver[16];                //蓝牙固件版本号
      char key_num[2];                     //按键个数
```

```
    char charge_plug[8];            //是否充电
    char audioname[32];             //设备 ssid


}DEVICEINFO;
```

使用示例：

```
    void device_info(void)
{
    int i = 0;
    DEVICEINFO *aplist = NULL;
    aplist = (DEVICEINFO *)calloc(1, sizeof(DEVICEINFO));
    int count = get_device_info(aplist);

    printf("sys_language [%d]:%s\n", i, aplist->sys_language);
    printf("sup_language[%d]:%s\n", i, aplist->sup_language);
    printf("firmware[%d]:%s\n", i, aplist->firmware);
    printf("release[%d]:%s\n", i, aplist->release);
    printf("mac_addr[%d]:%s\n", i, aplist->mac_addr);
    printf("uuid[%d]:%s\n", i, aplist->uuid);
    printf("project[%d]:%s\n", i, aplist->project);
    printf("battery_level[%d]:%s\n", i, aplist->battery_level);
    printf("function_support[%d]:%s\n", i, aplist->function_support);
    printf("bt_ver[%d]:%s\n", i, aplist->bt_ver);
    printf("key_num[%d]:%s\n", i, aplist->key_num);
    printf("charge_plug[%d]:%s\n", i, aplist->charge_plug);
    printf("ssid[%d]:%s\n", i, aplist->audioname);

    free(aplist);

}
```

3 数据类型