



Music 应用 详细设计说明书

珠海市杰理科技股份有限公司
Zhuhai Jieli Technologyco.,LTD

版权所有，未经许可，禁止外传

修改记录

版本	更新日期	描述
V1.0	2020/08/11	初稿



目录

1. 引 言.....	4
1.1. 编写目的.....	4
1.2. 参考资料.....	4
1.3. 术语和缩写词.....	4
2. 总体设计.....	4
2.1. 需求概述.....	4
2.2. 总体架构设计.....	5
2.3. 详细流程框图.....	6
2.4. 关键数据结构说明.....	6
3. 应用系统事件.....	7
3.1. 事件分类.....	7
3.2. 按键事件处理.....	7
3.3. 设备事件处理.....	7
4. 解码器错误处理.....	8
4.1. 错误处理说明.....	8
4.2. Music player 错误码表.....	8
5. 扫盘打断机制.....	9
5.1. 机制说明.....	9
5.2. 机制实现流程图.....	10
6. 断点说明.....	10
6.1. 断点控制句柄创建.....	10
6.2. 断点信息控制句柄释放.....	10
6.3. 当前播放歌曲断点获取及保存.....	10
6.4. 从 vm 获取设备断点信息并断点播放.....	11
7. 设备提示音实现.....	11
7.1. 设备提示音功能描述.....	11
7.2. 功能配置.....	11
8. 详细接口注释.....	11
static void music_player_play_success(void *priv, int parm).....	11
static void music_player_play_end(void *priv, int parm).....	12
static int music_player_scandisk_break(void).....	12
static void music_tone_play_end_callback(void *priv, int flag).....	12
void music_player_err_deal(int err).....	12
static int music_key_event_opr(struct sys_event *event).....	13
static int music_sys_event_handler(struct sys_event *event).....	13
void music_task_set_parm(u8 type, int val).....	13
static void music_player_play_start(void).....	13
static void music_task_start().....	14
static void music_task_close().....	14
int music_app_check(void).....	14
void app_music_task().....	14

1. 引言

1.1. 编写目的

该文档为基于 AC696N soundbox 平台开发 Music 应用的人员提供相应的设计开发文档。也可以为测试 Music 应用的测试人员提供参考。

文档中详细定义了 Music 应用的总体功能、系统的接口和数据属性；对程序的基本结构、功能模块以及各个程序的名称进行了划分，以便于 Music 应用的详细设计和二次开发。

1.2. 参考资料

[1]

1.3. 术语和缩写词

缩写和术语	解 释
sclust	文件簇号
breakpoint	歌曲断点信息，包括文件信息和解码信息
valid dev	有效播放设备，这个有效是指设备有可播放文件的意思
FF	快进
FR	快退
scanbreak	打断正在扫描设备的过程
repeat	播放循环模式
music player	音乐播放器
logo	设备逻辑盘符
evt_logo	设备上线时对应的设备逻辑盘符
vm	记忆存储区

2. 总体设计

2.1. 需求概述

本应用主要是基于 AC696N soundbox 的 SDK 系统开发包来实现音乐播放功能。

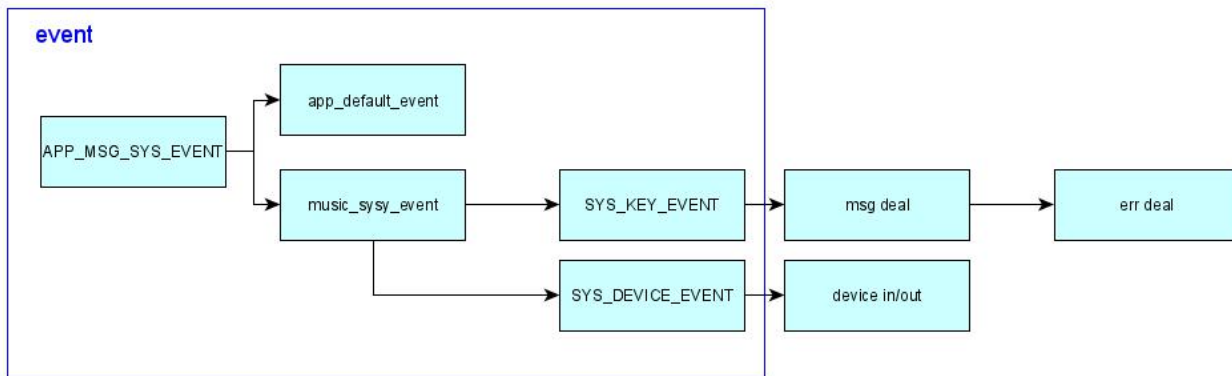
Music 应用主要实现的功能包括：

版权所有，侵权必究

4

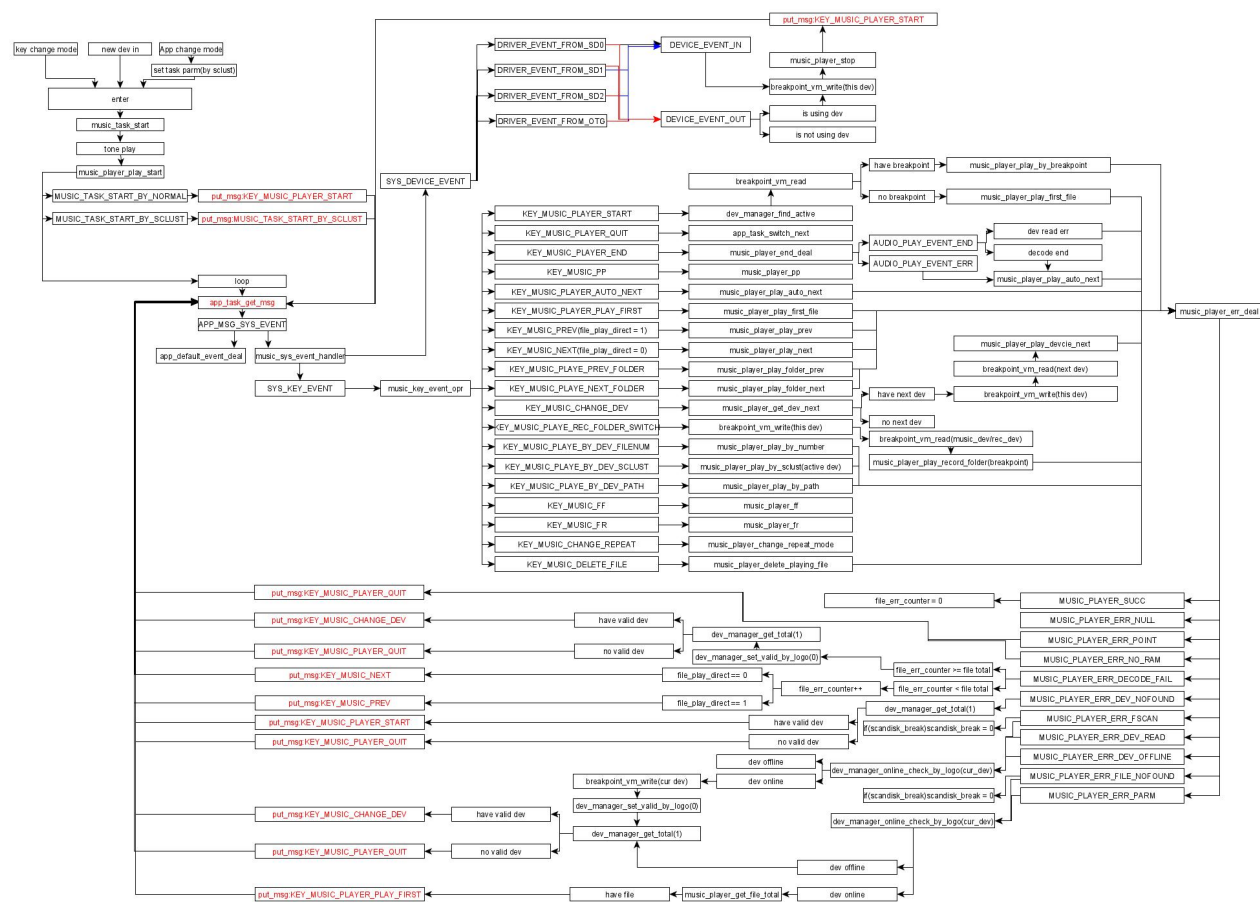
- (1) 实现 MP3、WMA、WAV、FLAC、APE、M4A、AMR、DTS 等歌曲的播放（可以在板卡配置项中选配）。
- (2) 提供播放/暂停、快进快退、上下曲、设备切换、循环模式切换、路径/序号/簇号播放、录音文件夹过滤、录音/音乐切换播放、文件删除等操作
- (3) 支持断点存取
- (4) 支持播放信息获取，如：当前文件序号、簇号、断点信息、文件句柄、设备句柄等

2.2. 总体架构设计



模块名称	功能简述	对应文件
event	处理系统产生的各种事件	music.c
msg_deal	按键消息处理（music_key_event_opr） 负责调用 music_player 提供的 API 接口，实现歌曲播放、切换、操作等，参考章节 2.3 的 详细流程框图	music.c
device in/out	设备上下线处理	music.c
err_deal	错误处理（播放产生的错误，参考 music_player 错误码表），参考章节 2.3 的 详细流程框图	music.c

2.3. 详细流程框图



2.4. 关键数据结构说明

///模式参数结构体

```
struct __music_task_parm {
    u8 type;
    int val;
};
```

///断点信息结构体

```
struct __breakpoint {
    ///文件信息
    u32 fsize;
    u32 sclust;
    ///解码信息
    struct audio_dec_breakpoint dbp;
};
```

```
///music 模式控制结构体
struct __music {
    struct __music_task_parm task_parm;
    u16 file_err_counter;//错误文件统计
    u8 file_play_direct;//0:下一曲, 1: 上一曲
    u8 scandisk_break;//扫描设备打断标志

};
```

3. 应用系统事件

3.1. 事件分类

系统事件分类:

- (1) 按键事件 (SYS_KEY_EVENT)
主要是各种按键触发的事件及软件流程 post 的事件
- (2) 设备事件 (SYS_DEVICE_EVENT)
主要是各种设备上下线事件, 本应用主要响应以下事件 (in/out):
DRIVER_EVENT_FROM_SD0
DRIVER_EVENT_FROM_SD1
DRIVER_EVENT_FROM_SD2
DEVICE_EVENT_FROM_OTG

以应用情景对事件分类:

- (1) 应用模式内部事件 (music_sys_event_handler)
- (2) 公共事件 (app_default_event_deal)
公共事件同样会响应一些例如公共按键 (模式切换/音量加减等)、设备上下线 (linein/pc 上下线等)、蓝牙事件 (连接、通话、播放等)

3.2. 按键事件处理

参考 2.3 详细流程框图。

3.3. 设备事件处理

参考 2.3 详细流程框图, 以下是对设备事件 music_sys_event_handler 中的 SYS_DEVICE_EVENT 类事件特别说明:


```
case SYS_DEVICE_EVENT:
    switch ((u32)event->arg) {
        case DRIVER_EVENT_FROM_SD0:
        case DRIVER_EVENT_FROM_SD1:
        case DRIVER_EVENT_FROM_SD2:
            evt_logo = (char *)event->u.dev.value;
        case DEVICE_EVENT_FROM_OTG:
            if ((u32)event->arg == DEVICE_EVENT_FROM_OTG) {
                evt_logo = (char *)"udisk0";
            }
            logo = music_player_get_dev_cur();
            log_i("evt_logo =%s, logo = %s\n", evt_logo, logo);
            if (event->u.dev.event == DEVICE_EVENT_OUT) {
                if (logo == NULL) {
                    break; // 设备掉线，判断是否是正在使用的设备
                }
                if (logo && (0 == strcmp(logo, evt_logo))) {
                    ///相同的设备才响应
                    if (music_player_get_playing_breakpoint(breakpoint, 1) == true) {
                        breakpoint_vm_write(breakpoint, logo);
                    }
                    ///停止解码，防止设备掉线后还继续使用
                    music_player_stop(1);
                    ///重新选择活动设备播放
                    app_task_put_key_msg(KEY_MUSIC_PLAYER_START, 0); //卸载了设备再执行
                    log_i("KEY_MUSIC_PLAYER_START AFTER UMOUNT\n");
                }
            } else { // 新设备上线，播放新设备
                ///新设备上线，先记录当前设备断电， 然后播放活动设备
                if (music_player_get_playing_breakpoint(breakpoint, 1) == true) {
                    breakpoint_vm_write(breakpoint, logo);
                }
                app_task_put_key_msg(KEY_MUSIC_PLAYER_START, 0);
                log_i("KEY_MUSIC_PLAYER_START AFTER MOUNT\n");
                ///先挂载了设备再执行
            }
    }
}
```

4. 解码器错误处理

4.1. 错误处理说明

这里的出错误处理是指 music 应用调用 music_player 接口时返回值处理，二次开发可以根据实际需求根据不同错误码引导不同的流程设计，SDK 默认的错误处理请参考章节 2.3 详细流程框图。

4.2. Music player 错误码表

///解码错误码表

```
enum {
    MUSIC_PLAYER_ERR_NULL = 0x0,    ///没有错误， 不需要做任何
    MUSIC_PLAYER_SUCC      = 0x1,    ///播放成功， 可以做相关显示、断点记忆等处理
}
```

版权所有，侵权必究

8


```
MUSIC_PLAYER_ERR_PARM,           ///  
MUSIC_PLAYER_ERR_POINT,         ///  
MUSIC_PLAYER_ERR_NO_RAM,        ///  
MUSIC_PLAYER_ERR_DECODE_FAIL,   ///  
MUSIC_PLAYER_ERR_DEV_NOFOUND,   ///  
MUSIC_PLAYER_ERR_DEV_OFFLINE,   ///  
MUSIC_PLAYER_ERR_DEV_READ,      ///  
MUSIC_PLAYER_ERR_FSCAN,         ///  
MUSIC_PLAYER_ERR_FILE_NOFOUND,  ///  
};
```

5. 扫盘打断机制

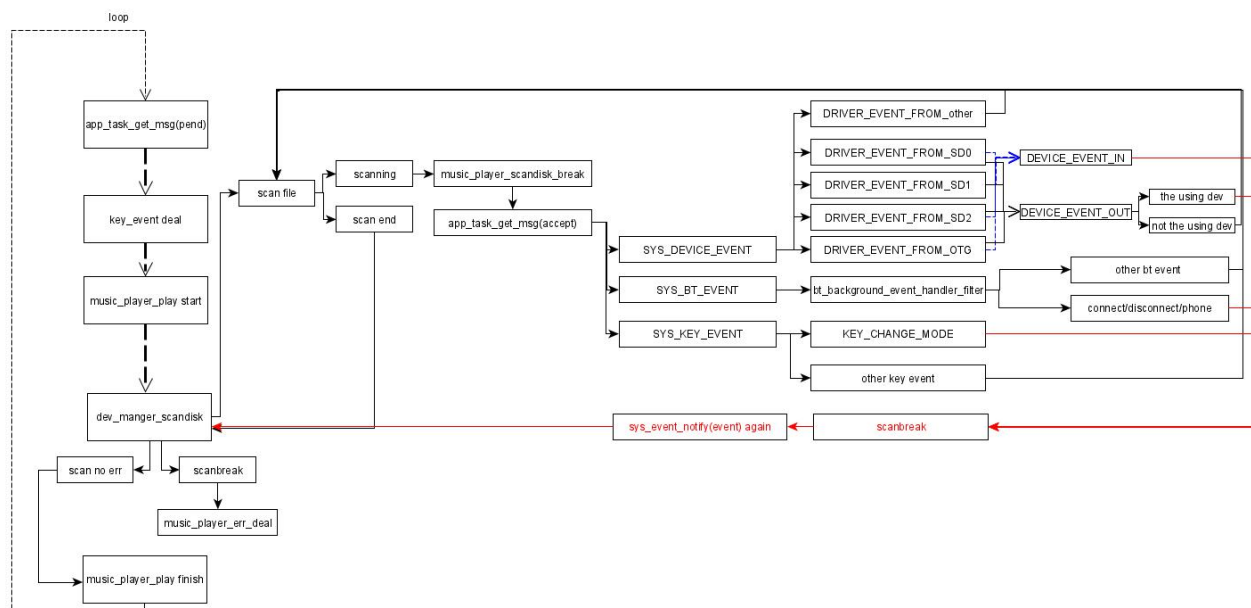
5.1. 机制说明

设备在 music_player 调用 scandisk 接口的时候，会注册 music_player_scandisk_break 接口（解决长时间扫盘打断问题），在扫描过程会被调用，music_player_scandisk_break 函数内部会对消息进行查询，SDK 默认会对一些例如按键切换模式、设备插拔、蓝牙连接等触发的事件进行扫盘打断动作，如二次开发有特殊情境需要进行打断，可以在解析到特定的事件后，将 scandisk_break 标志位设置为 1 即可，以模式切换按键为例：

```
150         __this->scandisk_break = 1;  
151     }  
152     if (__this->scandisk_break == 0) {  
153         dev_status_event_filter(event);  
154     }  
155     break;  
156 }  
157 break;  
158 case SYS_BT_EVENT:  
159     if (bt_background_event_handler_filter(event)) {  
160         __this->scandisk_break = 1;  
161     }  
162     break;  
163 case SYS_KEY_EVENT: ←  
164     switch (event->u.key.event) {  
165         case KEY_CHANGE_MODE: ←  
166             ///  
167             __this->scandisk_break = 1;  
168             break;  
169     }  
170     break;  
171 }  
172 break;  
173 }  
174 if (__this->scandisk_break) {  
175     ///  
176     sys_event_notify(event);  
177     printf("scandisk_break!!!!\n");  
178     return 1;  
179 } else {  
180     return 0;  
181 }  
182 }
```

因为此处消息已被获取，要重新发送后，在扫盘之后响应

5.2. 机制实现流程图



6. 断点说明

6.1. 断点控制句柄创建

```
///获取断点句柄， 后面所有断点读/写都需要用到
breakpoint = breakpoint_handle_creat();
```

6.2. 断点信息控制句柄释放

```
breakpoint_handle_destroy(&breakpoint);
```

6.3. 当前播放歌曲断点获取及保存

```
if (music_player_get_playing_breakpoint(breakpoint, 1/*1:获取所有信息, 0: 只获取文件信息*/) == true)
{
    breakpoint_vm_write(breakpoint, logo);
}
```

6.4. 从 vm 获取设备断点信息并断点播放

```
if (true == breakpoint_vm_read(breakpoint, logo)) {  
    err = music_player_play_by_breakpoint(logo, breakpoint);  
}
```

7. 设备提示音实现

7.1. 设备提示音功能描述

设备提示音指的是 music 模式设备切换过程中会播放对应设备的提示音，如果 SD 卡插入播报提示音--"SD 卡模式"

7.2. 功能配置

通过宏配置 MUSIC_DEVICE_TONE_EN 实现：

- 1: 支持设备提示音，原有音乐模式提示音不播报
- 2、不支持设备提示音，播放音乐模式提示音

8. 详细接口注释

```
/**-----*/  
/**@brief    music 解码成功回调  
    @param    priv:私有参数，  parm:暂时未用  
    @return  
    @note     此处可以做一些用户操作， 如断点保存， 显示， 获取播放信息等  
*/  
/**-----*/
```

```
static void music_player_play_success(void *priv, int parm)
```

```
/**-----*/  
/**@brief    music 解码结束回调处理  
    @param  
    @return  
    @note     此处统一将错误通过消息的方式发出， 在 key msg 中统一响应  
*/
```

版权所有，侵权必究

/*-----*/

static void music_player_play_end(void *priv, int parm)

/*-----*/

/**@brief music 播放器扫盘打断接口
 @param
 @return 1:打断当前扫描, 0:正常扫描
 @note 设备扫描耗时长的情况下, 此接口提供打断机制
*/

/*-----*/

static int music_player_scandisk_break(void)

/*-----*/

/**@brief music 模式提示音播放结束处理
 @param
 @return
 @note
*/

/*-----*/

static void music_tone_play_end_callback(void *priv, int flag)

/*-----*/

/**@brief music 模式解码错误处理
 @param err:错误码, 详细错误码描述请看 MUSIC_PLAYER 错误码表枚举
 @return
 @note
*/

/*-----*/

void music_player_err_deal(int err)

/*-----*/

/**@brief music 按键消息入口
 @param 无
 @return 1、消息已经处理, common 不再处理 0、消息发送到 common 处理
 @note
*/

/*-----*/

static int music_key_event_opr(struct sys_event *event)

```
/**-----*/
/**@brief    music 设备事件响应接口
    @param    无
    @return    1、当前消息已经处理，comomon 不再做处理 0、common 统一处理
    @note
*/
/*-----*/
```

static int music_sys_event_handler(struct sys_event *event)

```
/**-----*/
/**@brief    music 模式切换前参数设置
    @param    type:播放方式,暂时支持:
                MUSIC_TASK_START_BY_NORMAL: 首次播放按照正常断点播放
                MUSIC_TASK_START_BY_SCLUST: 首次播放按照簇号播放
    val:播放参数
    @return
    @note    首次播放执行参考 music_player_play_start 接口
*/
/*-----*/
```

void music_task_set_parm(u8 type, int val)

```
/**-----*/
/**@brief    music 模式首次播放
    @param    无
    @return
    @note    切换到音乐模式前可以通过接口 music_task_set_parm 设置参数,
                进入音乐模式后会按照对应参数播放
*/
/*-----*/
```

static void music_player_play_start(void)

```
/**-----*/
/**@brief    music 模式初始化处理
    @param    无
    @return
    @note
```

版权所有，侵权必究

13

*/

/*-----*/

static void music_task_start()

/*-----*/

/**@brief music 模式退出处理

@param 无

@return

@note

*/

/*-----*/

static void music_task_close()

/*-----*/

/**@brief music 在线检测 切换模式判断使用

@param 无

@return 1 设备在线 0 设备不在线

@note

*/

/*-----*/

int music_app_check(void)

/*-----*/

/**@brief music 主任务

@param 无

@return 无

@note

*/

/*-----*/

void app_music_task()