



Music player

接口设计说明书

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. 文档介绍.....	5
1.1. 文档目的.....	5
1.2. 参考文献.....	5
[1].....	5
1.3. 关键词.....	5
2. 功能概述.....	5
3. 关键结构体、枚举类型及参数.....	6
3.1. 回调接口结构体.....	6
3.2. 参数设置结构体.....	7
3.3. music player 总控制句柄.....	7
3.4. 文件扫描参数.....	8
3.5. 播放器错误码表.....	9
4. 播放类接口内部流程设计.....	10
4.1. 基础流程.....	10
4.2. 流程说明.....	10
4.3. 接口举例说明.....	11
5. 详细接口说明.....	12
void music_player_destroy(void).....	12
bool music_player_creat(void *priv, struct __player_parm *parm).....	12
int music_player_end_deal(int parm).....	13
int music_player_decode_start(FILE *file, struct audio_dec_breakpoint *dbp).....	13
bool music_player_get_playing_breakpoint(struct __breakpoint *bp, u8 flag).....	13
u16 music_player_get_file_total(void).....	13
u16 music_player_get_file_cur(void).....	14
u16 music_play_get_fileindir_number(void).....	14
u16 music_play_get_dir_cur(void).....	14
u16 music_play_get_dir_total(void).....	14
FILE *music_player_get_file_hdl(void).....	15
u32 music_player_get_file_sclust(void).....	15
char *music_player_get_dev_cur(void).....	15
char *music_player_get_dev_next(void).....	15
char *music_player_get_dev_prev(void).....	16
int music_player_get_play_status(void).....	16
int music_player_get_dec_cur_time(void).....	16
int music_player_get_dec_total_time(void).....	17
u8 music_player_get_repeat_mode(void).....	17
char *music_player_get_cur_music_dev(void).....	17
bool music_player_get_record_play_status(void).....	17
int music_player_pp(void).....	18
void music_player_stop(u8 fsn_release).....	18
void music_player_ff(int step).....	18
void music_player_fr(int step).....	18

int music_player_set_repeat_mode(u8 mode).....	19
int music_player_change_repeat_mode(void).....	19
int music_player_delete_playing_file(void).....	19
int music_player_play_prev(void).....	19
int music_player_play_next(void).....	20
int music_player_play_first_file(char *logo).....	20
int music_player_play_last_file(char *logo).....	20
int music_player_play_auto_next(void).....	20
int music_player_play_folder_prev(void).....	21
int music_player_play_folder_next(void).....	21
int music_player_play_devcie_prev(struct __breakpoint *bp).....	21
int music_player_play_devcie_next(struct __breakpoint *bp).....	21
int music_player_play_by_breakpoint(char *logo, struct __breakpoint *bp).....	22
int music_player_play_by_number(char *logo, u32 number).....	22
int music_player_play_by_sclust(char *logo, u32 sclust).....	22
int music_player_play_by_path(char *logo, const char *path).....	23
int music_player_play_record_folder(char *logo, struct __breakpoint *bp).....	23

1. 文档介绍

1.1. 文档目的

Music player 播放器接口为设备播放（music 模式）提供 api 接口，可以支持 U 盘/SD 等设备音乐播放，并且提供不同播放方式，例如：断点、簇号、序号等，所有接口以展开式的方式开放出来，为用户在二次开发提供灵活发挥的空间。

1.2. 参考文献

[1].

1.3. 关键词

缩写、术语	解 释
Music_player	播放器
Dev_manager	设备管理器
file_manager	文件管理器

2. 功能概述

music player 播放器模块接口实现以下功能：

- (1) 支持播放器启动及退出
- (2) 提供播放类接口
 - 1) 播放上一曲
 - 2) 播放下一曲
 - 3) 播放第一曲
 - 4) 播放最后一曲
 - 5) 自动播放下一曲
 - 6) 播放上一个文件夹
 - 7) 播放下一个文件夹
 - 8) 播放上一设备
 - 9) 播放下一设备
 - 10) 断点播放

- 11) 序号播放
- 12) 簇号播放
- 13) 路径播放
- 14) 一键切换录音文件夹播放
- (3) 操作类接口
 - 1) 播放/暂停
 - 2) 快进/快退
 - 3) 设置播放循环模式
 - 4) 停止播放
- (4) 状态获取类接口、
 - 1) 获取播放断点
 - 2) 获取文件总数
 - 3) 获取当前播放文件号
 - 4) 获取当前播放文件所在文件夹下的文件总数
 - 5) 获取当前播放文件所在文件夹
 - 6) 获取文件夹总数
 - 7) 获取当前播放文件的文件句柄
 - 8) 获取当前播放文件的文件簇号
 - 9) 获取当前播放设备
 - 10) 获取当前播放的音乐设备（如果当前播放的是录音设备，返回对应的音乐设备）
 - 11) 获取当前播放的下一个设备
 - 12) 获取当前播放的上一个设备
 - 13) 获取播放状态
 - 14) 获取当前播放歌曲时间
 - 15) 获取当前播放歌曲总时间
 - 16) 获取当前播放循环模式
 - 17) 获取当前是否是录音设备播放状态
- (5) 其他
 - 1) 删除当前正在播放的设备

3. 关键结构体、枚举类型及参数

3.1. 回调接口结构体

```
struct __player_cb {  
    ///解码成功回调  
    void (*start)(void *priv, int parm);  
};
```

```

///解码结束回调
void (*end)(void *priv, int parm);
///扫盘打断回调
int (*fns_break)(void);
};

```

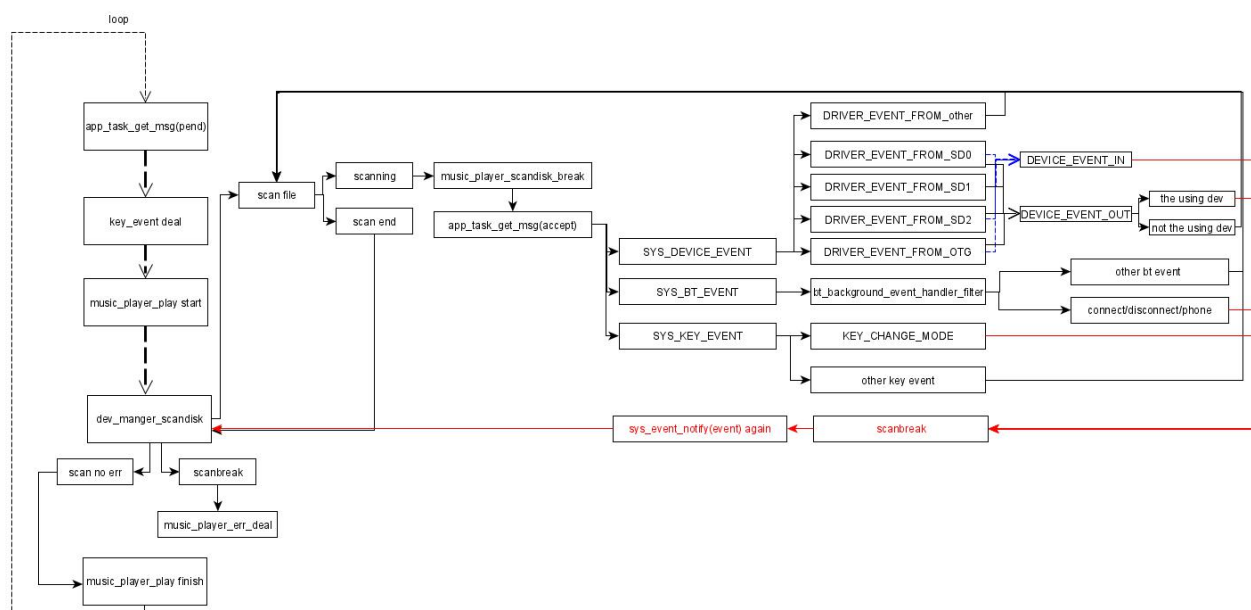
说明:

播放器回调有应用层在调用 `music_player_creat` 启动播放器时传入。

解码成功回调：应用层可以在这里做一些解码状态获取，如：歌曲序号、文件名，及显示等。

解码结束回调：可以根据方案需求引导不同的流程走向。

扫盘打断回调：扫盘打断回调为长时间扫描设备（主要是针对文件特别多的设备）提供打断机制，具体实现方式，请参考文档【music 应用详细设计说明】，打断原理如下流程（流程结合了 music 应用，即 music.c 中实现）：



3.2. 参数设置结构体

```

struct __player_parm {
    struct __player_cb *cb;
    ///其他扩展
};

```

3.3. music player 总控制句柄

```

struct __music_player {
    struct __dev          *dev;//当前播放设备节点
    struct vfscan         *fns;//设备扫描句柄
};

```

版权所有，侵权必究

```
FILE                *file;//当前播放文件句柄
void                *priv;//music 回调函数，私有参数
struct __player_parm parm;//回调及参数配置
};
```

3. 4. 文件扫描参数

```
static const char scan_parm[] = "-t"
#ifdef (TCFG_DEC_MP3_ENABLE)
    "MP1MP2MP3"
#endif
#ifdef (TCFG_DEC_WMA_ENABLE)
    "WMA"
#endif
#ifdef (TCFG_DEC_WAV_ENABLE || TCFG_DEC_DTS_ENABLE)
    "WAVDTS"
#endif
#ifdef (TCFG_DEC_FLAC_ENABLE)
    "FLA"
#endif
#ifdef (TCFG_DEC_APE_ENABLE)
    "APE"
#endif
#ifdef (TCFG_DEC_M4A_ENABLE)
    "M4AMP4AAC"
#endif
#ifdef (TCFG_DEC_AMR_ENABLE)
    "AMR"
#endif
#ifdef (TCFG_DEC_DECRYPT_ENABLE)
    "SMP"
#endif
#ifdef (TCFG_DEC_MIDI_ENABLE)
    "MID"
#endif
    "-sn -r"
#ifdef (TCFG_RECORD_FOLDER_DEV_ENABLE)
    "-m"
    REC_FOLDER_NAME
#endif
;
```

说明:

版权所有，侵权必究

scan_parm 在扫描 dev_manager_scan_disk 调用是使用， 参数设置了文件后缀如：MP3/WAV/WMA 等后缀的歌曲将在设备扫描时将被扫描到， 没有的后缀， 将被过滤（被过滤的歌曲是不会被播放的）。同时该参数也设置了一些特殊的内容，如录音文件夹内容过滤等。

3. 5. 播放器错误码表

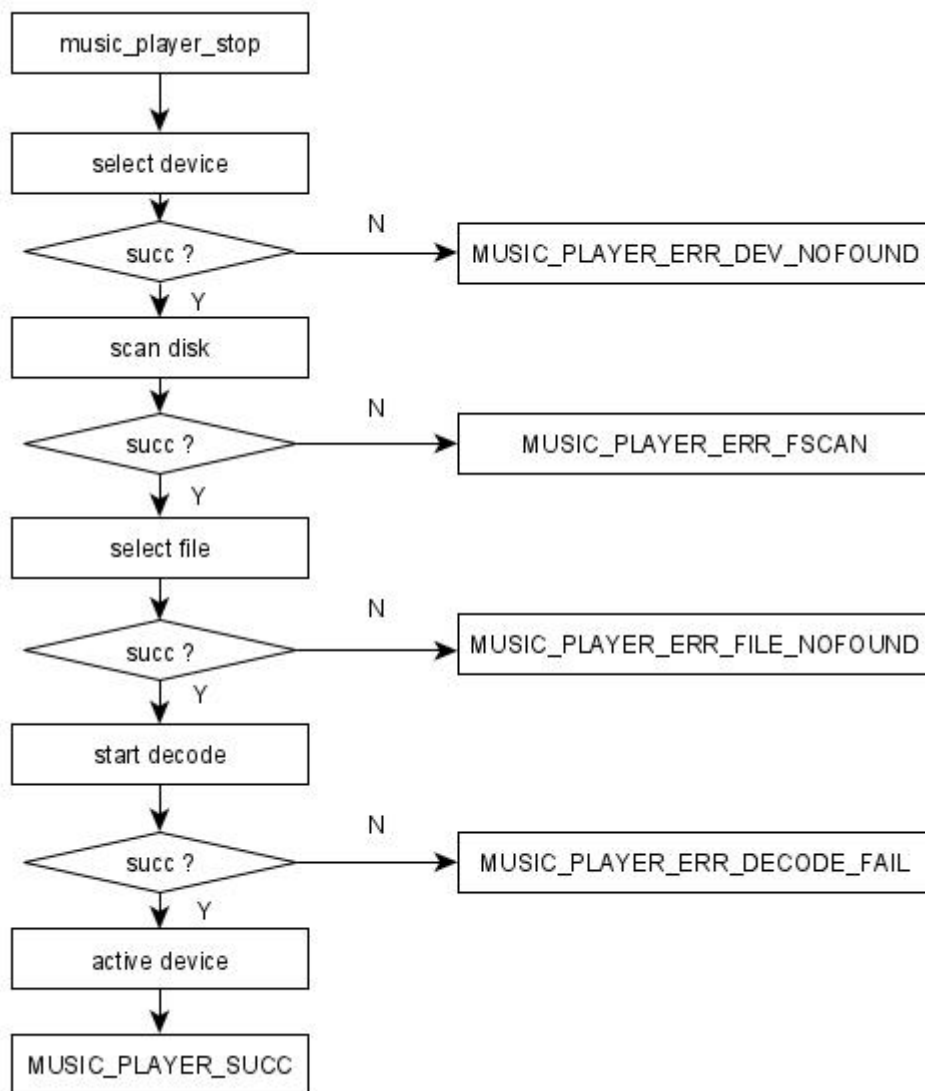
```
enum {  
    MUSIC_PLAYER_ERR_NULL = 0x0,    ///没有错误， 不需要做任何  
    MUSIC_PLAYER_SUCC      = 0x1,    ///播放成功， 可以做相关显示、断点记忆等处理  
    MUSIC_PLAYER_ERR_PARM,           ///参数错误  
    MUSIC_PLAYER_ERR_POINT,          ///参数指针错误  
    MUSIC_PLAYER_ERR_NO_RAM,          ///没有 ram 空间错误  
    MUSIC_PLAYER_ERR_DECODE_FAIL,     ///解码器启动失败错误  
    MUSIC_PLAYER_ERR_DEV_NOFOUND,     ///没有找到指定设备  
    MUSIC_PLAYER_ERR_DEV_OFFLINE,     ///设备不在线错误  
    MUSIC_PLAYER_ERR_DEV_READ,        ///设备读错误  
    MUSIC_PLAYER_ERR_FSCAN,           ///设备扫描失败  
    MUSIC_PLAYER_ERR_FILE_NOFOUND,    ///没有找到文件  
};
```

说明：

具体错误处理统一外抛到应用层根据实际需求引导不同流程的走向，SDK 默认在 music.c 集中处理。

4. 播放类接口内部流程设计

4.1. 基础流程



4.2. 流程说明

- (1) 解码停止 music_player_stop 的时候要判断接下来的播放是否需要重新扫盘决定参数是传 1 还是 0
- (2) 选择设备建议判断是否是当前正在使用的设备，如果是不需要重新选择，只需要通过 dev_manager_online_check 判断设备是否还在线
- (3) 如果需要重新选择设备或者有特殊需求就需要重新扫盘，如果要重新扫盘(1)中描述的 stop 参数必须要是 1

- (4) 选择文件 file_manager_select, 根据不同的需求通过设置不同的参数进行文件选择
- (5) 启动解码器, 需要提供的参数为文件控制句柄及断点信息, 如果没有断点填 NULL, 如果有指定断点, 播放的时候会从指定的断点位置开始播放
- (6) 所有错误返回值由上层应用层处理 (sdk 默认在 music.c 中处理), 具体提含义情况 music_player 错误码表

4.3. 接口举例说明

此处以**指定设备断点播放**作为特例说明展开式事例说明:

```
int music_player_play_by_breakpoint(char *logo, struct __breakpoint *bp)
{
    //判断参数是否有效
    if(bp == NULL) {
        return MUSIC_PLAYER_ERR_PARM;
    }
    if(logo == NULL) {
        //设备没有指定
        music_player_stop(0); //停止解码, 注意参数
        //检查当前设备是否在线
        if(dev_manager_online_check(__this->dev, 1) == 0) {
            return MUSIC_PLAYER_ERR_DEV_OFFLINE;
        }
        //使用当前设备, 不需要找设备, 不需要扫描
    } else {
        music_player_stop(1); //停止解码, 注意参数
        //查找指定设备
        __this->dev = dev_manager_find_spec(logo, 1/*1 表示有效设备*/);
        if(__this->dev == NULL) {
            return MUSIC_PLAYER_ERR_DEV_NOFOUND;
        }
        //找到了设备, 下面开始扫盘
        __this->fsn = dev_manager_scan_disk(__this->dev, NULL, scan_parm, cycle_mode, __this->parm.cb->fsn_break);
    }
    if(__this->fsn == NULL) {
        return MUSIC_PLAYER_ERR_FSCAN;
    }
    //扫盘成功后, 下面就是文件选择了
    __this->file = file_manager_select(__this->fsn, FSEL_BY_SCLUST, bp->sclust); //根据文件簇号查找断点文件
    if(__this->file == NULL) {
        return MUSIC_PLAYER_ERR_FILE_NOFOUND;
    }
    //文件选择好, 接下来就是启动解码器
```

```
int err = music_player_decode_start(__this->file, &(bp->dbp));
if (err == MUSIC_PLAYER_SUCC) {
    //播放成功，将当前设备激活，下次查找活动设备，就是当前激活的设备了
    dev_manager_set_active(__this->dev);
    log_i("[%s %d] ok\n", __FUNCTION__, __LINE__);
}
//此处断点播放执行完毕，所有的返回值，请根据错误码表进行处理
return err;
}
```

5. 详细接口说明

```
/**-----*/
/**@brief    music_player 释放接口
    @param
    @return
    @note
*/
/**-----*/
```

void music_player_destroy(void)

```
/**-----*/
/**@brief    music_player 创建接口
    @param
    @return
    @note
*/
/**-----*/
```

bool music_player_creat(void *priv, struct __player_parm *parm)

```
/**-----*/
/**@brief    music_player 播放结束处理
    @param    parm: 结束参数，暂时没用
    @return
    @note
*/
/**-----*/
```

int music_player_end_deal(int parm)

```
/**-----*/  
/**@brief    music_player 解码器启动接口  
    @param  
        file:  
            文件句柄  
        dbp:  
            断点信息  
    @return   music_player 错误码  
    @note  
*/  
/*-----*/
```

int music_player_decode_start(FILE *file, struct audio_dec_breakpoint *dbp)

```
/**-----*/  
/**@brief    music_player 获取当前播放设备断点信息  
    @param  
        bp:  
            断点缓存，外部调用提供  
        flag:  
            1: 需要获取歌曲断点信息及文件信息， 0: 只获取文件信息  
    @return   成功与否  
    @note  
*/  
/*-----*/
```

bool music_player_get_playing_breakpoint(struct __breakpoint *bp, u8 flag)

```
/**-----*/  
/**@brief    music_player 获取当前播放设备文件总数  
    @param  
    @return   文件总数  
    @note  
*/  
/*-----*/
```

u16 music_player_get_file_total(void)

```
/**-----*/
```

版权所有，侵权必究

13

```
/**@brief    music_player 获取当前播放文件序号
    @param
    @return    当前文件序号
    @note
*/
/*-----*/
```

u16 music_player_get_file_cur(void)

```
/*-----*/
/**@brief    music_player 获取当前播放文件所在文件夹的文件总数
    @param
    @return    文件总数
    @note
*/
/*-----*/
```

u16 music_play_get_fileindir_number(void)

```
/*-----*/
/**@brief    music_player 获取当前播放文件所在文件夹
    @param
    @return    当前文件夹序号
    @note
*/
/*-----*/
```

u16 music_play_get_dir_cur(void)

```
/*-----*/
/**@brief    music_player 获取文件夹总数
    @param
    @return    文件夹总数
    @note
*/
/*-----*/
```

u16 music_play_get_dir_total(void)

```
/*-----*/
/**@brief    music_player 获取文件句柄
```

版权所有，侵权必究

@param
@return 文件句柄
@note 需要注意文件句柄的生命周期

*/

/*-----*/

FILE *music_player_get_file_hdl(void)

/*-----*/

/**@brief music_player 获取文件簇号

@param

@return 文件簇号, -1:无效

@note

*/

/*-----*/

u32 music_player_get_file_sclust(void)

/*-----*/

/**@brief music_player 获取当前播放设备盘符

@param

@return 设备盘符

@note

*/

/*-----*/

char *music_player_get_dev_cur(void)

/*-----*/

/**@brief music_player 获取当前播放设备下一个设备

@param

@return 设备盘符

@note

*/

/*-----*/

char *music_player_get_dev_next(void)

/*-----*/

/**@brief music_player 获取当前播放设备上一个设备

版权所有，侵权必究

```
@param
@return 设备盘符
@note
*/
/*-----*/
```

char *music_player_get_dev_prev(void)

```
/*-----*/
/**@brief music_player 获取当前播放状态
@param
@return 返回值如:
        FILE_DEC_STATUS_STOP,//解码停止
        FILE_DEC_STATUS_PLAY,//正在解码
        FILE_DEC_STATUS_PAUSE,//解码暂停
@note
*/
/*-----*/
```

int music_player_get_play_status(void)

```
/*-----*/
/**@brief music_player 获取当前播放歌曲时间
@param
@return 当前播放时间
@note
*/
/*-----*/
```

int music_player_get_dec_cur_time(void)

```
/*-----*/
/**@brief music_player 获取当前播放歌曲总时间
@param
@return 当前播放总时间
@note
*/
/*-----*/
```


int music_player_get_dec_total_time(void)

```
/**-----*/  
/**@brief    music_player 获取当前播放循环模式  
    @param  
    @return    当前播放循环模式  
    @note  
*/  
/*-----*/
```

u8 music_player_get_repeat_mode(void)

```
/**-----*/  
/**@brief    music_player 获取当前播放对应的 music 设备  
    @param  
    @return    设备盘符  
    @note    播放录音区分时，可以通过该接口判断当前播放的音乐设备是什么以便做录音区分判断  
*/  
/*-----*/
```

char *music_player_get_cur_music_dev(void)

```
/**-----*/  
/**@brief    music_player 获取当前录音区分播放状态  
    @param  
    @return    true: 录音文件夹播放, false: 非录音文件夹播放  
    @note    播放录音区分时，可以通过该接口判断当前播放的是录音文件夹还是非录音文件夹  
*/  
/*-----*/
```

bool music_player_get_record_play_status(void)

```
/**-----*/  
/**@brief    music_player 播放/暂停  
    @param  
    @return  
    @note  
*/  
/*-----*/
```

int music_player_pp(void)

```
/**-----*/
/**@brief    music_player 解码停止
    @param
        fsn_release:
            1: 释放扫盘句柄
            0: 不释放扫盘句柄
    @return
    @note    如果释放了扫盘句柄，需要重新扫盘，否则播放失败
*/
/*-----*/
```

void music_player_stop(u8 fsn_release)

```
/**-----*/
/**@brief    music_player 快进
    @param    step: 快进步进
    @return
    @note
*/
/*-----*/
```

void music_player_ff(int step)

```
/**-----*/
/**@brief    music_player 快退
    @param    step: 快退步进
    @return
    @note
*/
/*-----*/
```

void music_player_fr(int step)

```
/**-----*/
/**@brief    music_player 设置播放循环模式
    @param    mode: 循环模式
                FCYCLE_ALL
                FCYCLE_ONE
                FCYCLE_FOLDER
```

FCYCLE_RANDOM

@return 循环模式

@note

*/

/*-----*/

int music_player_set_repeat_mode(u8 mode)

/*-----*/

/**@brief music_player 切换循环模式

@param

@return 循环模式

@note

*/

/*-----*/

int music_player_change_repeat_mode(void)

/*-----*/

/**@brief music_player 删除当前播放文件,并播放下一曲

@param

@return 播放错误码

@note

*/

/*-----*/

int music_player_delete_playing_file(void)

/*-----*/

/**@brief music_player 播放上一曲

@param

@return 播放错误码

@note

*/

/*-----*/

int music_player_play_prev(void)

/*-----*/

/**@brief music_player 播放下一曲

@param

版权所有，侵权必究

19

@return 播放错误码

@note

*/

/*-----*/

int music_player_play_next(void)

/*-----*/

/**@brief music_player 播放第一曲

@param

@return 播放错误码

@note

*/

/*-----*/

int music_player_play_first_file(char *logo)

/*-----*/

/**@brief music_player 播放最后一曲

@param

@return 播放错误码

@note

*/

/*-----*/

int music_player_play_last_file(char *logo)

/*-----*/

/**@brief music_player 自动播放下一曲

@param

@return 播放错误码

@note

*/

/*-----*/

int music_player_play_auto_next(void)

/*-----*/

/**@brief music_player 上一个文件夹

@param

@return 播放错误码

版权所有，侵权必究

20

```
@note
*/
/*-----*/

int music_player_play_folder_prev(void)

/*-----*/
/**@brief    music_player 下一个文件夹
    @param
    @return    播放错误码
    @note
*/
/*-----*/

int music_player_play_folder_next(void)

/*-----*/
/**@brief    music_player 上一个设备
    @param    bp: 断点信息
    @return    播放错误码
    @note
*/
/*-----*/

int music_player_play_devcie_prev(struct __breakpoint *bp)

/*-----*/
/**@brief    music_player 下一个设备
    @param    bp: 断点信息
    @return    播放错误码
    @note
*/
/*-----*/

int music_player_play_devcie_next(struct __breakpoint *bp)

/*-----*/
/**@brief    music_player 断点播放指定设备
    @param
        logo: 逻辑盘符, 如: sd0/sd1/udisk0
```

bp: 断点信息
@return 播放错误码
@note
*/
/*-----*/

int music_player_play_by_breakpoint(char *logo, struct __breakpoint *bp)

/*-----*/
/**@brief music_player 序号播放指定设备
@param
logo: 逻辑盘符, 如: sd0/sd1/udisk0
number: 指定播放序号
@return 播放错误码
@note
*/
/*-----*/

int music_player_play_by_number(char *logo, u32 number)

/*-----*/
/**@brief music_player 簇号播放指定设备
@param
logo: 逻辑盘符, 如: sd0/sd1/udisk0
sclust: 指定播放簇号
@return 播放错误码
@note
*/
/*-----*/

int music_player_play_by_sclust(char *logo, u32 sclust)

/*-----*/
/**@brief music_player 路径播放指定设备
@param
logo: 逻辑盘符, 如: sd0/sd1/udisk0, 设置为 NULL, 为默认当前播放设备
path: 指定播放路径
@return 播放错误码
@note
*/
/*-----*/

int music_player_play_by_path(char *logo, const char *path)

```
/**-----*/  
/**@brief    music_player 录音区分切换播放  
    @param  
        logo: 逻辑盘符, 如: sd0/sd1/udisk0, 设置为 NULL, 为默认当前播放设备  
        bp: 断点信息  
    @return   播放错误码  
    @note     通过指定设备盘符, 接口内部通过解析盘符是否"_rec"  
              来确定是切换到录音播放设备还是非录音播放设备  
*/  
/*-----*/
```

int music_player_play_record_folder(char *logo, struct __breakpoint *bp)