



Dev manager

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. 设备管理器结构体.....	6
3.2. 设备节点 const 信息结构体.....	6
3.3. 设备节点结构体.....	7
3.4. 设备配置参数.....	7
4. 设备管理流程框架.....	10
5. 如何将自定义设备加入到设备链表.....	11
6. 详细接口说明.....	12
int dev_manager_add(char *logo).....	13
int dev_manager_del(char *logo).....	13
struct __dev *dev_manager_check(struct __dev *dev).....	13
struct __dev *dev_manager_check_by_logo(char *logo).....	13
u32 dev_manager_get_total(u8 valid).....	14
struct __dev *dev_manager_find_first(u8 valid).....	14
struct __dev *dev_manager_find_last(u8 valid).....	14
struct __dev *dev_manager_find_prev(struct __dev *dev, u8 valid).....	14
struct __dev *dev_manager_find_next(struct __dev *dev, u8 valid).....	15
struct __dev *dev_manager_find_active(u8 valid).....	15
struct __dev *dev_manager_find_spec(char *logo, u8 valid).....	15
struct __dev *dev_manager_find_by_index(u32 index, u8 valid).....	16
void dev_manager_scan_disk_release(struct vfscan *fsn).....	16
struct vfscan *dev_manager_scan_disk(struct __dev *dev, const char *path, const char *parm, u8 cycle_mode, int (*callback)(void)).....	16
void dev_manager_set_valid(struct __dev *dev, u8 flag).....	17
void dev_manager_set_valid_by_logo(char *logo, u8 flag).....	17
void dev_manager_set_active(struct __dev *dev).....	17
void dev_manager_set_active_by_logo(char *logo).....	18
char *dev_manager_get_logo(struct __dev *dev).....	18
char *dev_manager_get_phy_logo(struct __dev *dev).....	18
char *dev_manager_get_rec_logo(struct __dev *dev).....	18
char *dev_manager_get_root_path(struct __dev *dev).....	19
char *dev_manager_get_root_path_by_logo(char *logo).....	19
char *dev_manager_get_bs_root_path(struct __dev *dev).....	19
char *dev_manager_get_bs_root_path_by_logo(char *logo).....	20
int dev_manager_online_check_by_logo(char *logo, u8 valid).....	20
int dev_manager_online_check(struct __dev *dev, u8 valid).....	20

static void dev_manager_task(void *p).....	20
void dev_manager_init(void).....	21



1. 文档介绍

1.1. 文档目的

Dev_manager 设备管理接口为设备为设备上下线及查询设备状态等操作提供 api 接口，可以支持如 U 盘/SD 等设备挂载、查找、激活等操作。

1.2. 参考文献

[1].

1.3. 关键词

缩写、术语	解 释
Dev_manager	设备管理器
valid	有效设备标志，有效指的是设备中有可播放文件，music_player 会查询有效设备进行播放，跳过无效设备。
logo	设备逻辑盘符，如：sd0/udisk0 等
active	最后活动活动设备，dev_manager_find_active 表示查找最后活动设备，dev_manager_set_active 可以将需要指定的设备设置为最后活动设备， 同时注意设备上线的时候默认会将设备设置为最后活动设备。
_rec	SDK 默认在需要支持录音文件夹区分设备后追加 “_rec” (例如：sd0 对应的录音文件播放设备为“sdk0_rec”), 并提供 dev_manager_get_rec_logo 接口获取录音播放设备及 dev_manager_get_phy_logo 接口获取录音播放设备对应的音乐设备。

2. 功能概述

Dev_manager 设备接口实现以下功能：

- (1) 支持设备上下线
- (2) 支持设备查找
 - 1) 查找第一个设备
 - 2) 查找最后一个设备
 - 3) 查找当前设备上一个设备
 - 4) 查找当前设备下一个设备
 - 5) 查找最后活动设备

- 6) 查找指定设备
- 7) 查找指定序号的设备
- (3) 支持设置设备信息
 - 1) 设置设备有效/无效
 - 2) 设置设备为活动设备
- (4) 支持获取设备信息
 - 1) 获取设备逻辑盘符
 - 2) 获取音乐设备逻辑盘符
 - 3) 获取录音播放设备逻辑盘符
 - 4) 获取设备根目录
 - 5) 获取设备文件浏览根目录
- (5) 支持设备在线检查
- (6) 其他
 - 1) 设备扫盘及扫盘释放

3. 关键结构体、枚举类型及参数

3.1. 设备管理器结构体

```
struct __dev_manager {  
    struct list_head    list;  
    OS_MUTEX            mutex;  
    OS_SEM              sem;  
};
```

说明:

该结构体主要起到设备链表管理的作用。

3.2. 设备节点 const 信息结构体

```
struct __dev_reg {  
    char *logo;//设备选择使用的逻辑盘符  
    char *name;//设备名称，底层匹配设备节点使用  
    char *storage_path;//设备路径，文件系统 mount 时使用  
    char *bs_storage_path;//设备浏览使用的设备路径，供文件浏览文件系统 mount 使用  
    char *root_path;//设备文件系统根目录  
    char *bs_root_path;//文件浏览设备文件系统根目录  
    char *fs_type;//文件系统类型,如: fat, sdfile  
};
```

版权所有，侵权必究

6

说明:

设备 const 信息, 在 dev_manager.c 中断 dev_reg[] 中配置。

3.3. 设备节点结构体

```
struct __dev {
    struct list_head    entry;
    struct __dev_reg    *parm;//设备参数
    volatile u8         valid:    1;//有效设备标记, 这里有效是指是否有可播放文件
    volatile u32        active_stamp;//活动设备时间戳, 通过时间戳记录当前最后活动设备
#ifdef (TCFG_LFN_EN)
    u8                  lfn_buf[512];//设备长文件名缓存
#endif
};
```

3.4. 设备配置参数

```
const struct __dev_reg dev_reg[] = {
    //内置 flash
    {
        /*logo*/          SDFILE_DEV,
        /*name*/           NULL,
        /*storage_path*/   SDFILE_MOUNT_PATH,
        /*bs_storage_path*/ NULL,
        /*root_path*/      SDFILE_RES_ROOT_PATH,
        /*bs_root_path*/   NULL,
        /*fs_type*/        "sdfile"
    },
    //内置录音
    {
        /*logo*/          "rec_sdfile",
        /*name*/           NULL,
        /*storage_path*/   "mnt/rec_sdfile",
        /*bs_storage_path*/ NULL,
        /*root_path*/      "mnt/rec_sdfile/C/",
        /*bs_root_path*/   NULL,
        /*fs_type*/        "rec_sdfile"
    },
    //sd0
    {
```

版权所有, 侵权必究

7

```
/*logo*/      "sd0",
/*name*/      "sd0",
/*storage_path*/  "storage/sd0",
/*bs_storage_path*/  SD0_BS_STORAGE_PATH,
/*root_path*/    "storage/sd0/C/",
/*bs_root_path*/  SD0_BS_ROOT_PATH,
/*fs_type*/      "fat"
},
//sd1
{
/*logo*/      "sd1",
/*name*/      "sd1",
/*storage_path*/  "storage/sd1",
/*bs_storage_path*/  SD1_BS_STORAGE_PATH,
/*root_path*/    "storage/sd1/C/",
/*bs_root_path*/  SD1_BS_ROOT_PATH,
/*fs_type*/      "fat"
},
//u 盘
{
/*logo*/      "udisk0",
/*name*/      "udisk0",
/*storage_path*/  "storage/udisk0",
/*bs_storage_path*/  UDISK0_BS_STORAGE_PATH,
/*root_path*/    "storage/udisk0/C/",
/*bs_root_path*/  UDISK0_BS_ROOT_PATH,
/*fs_type*/      "fat"
},
//sd0 录音文件夹分区
{
/*logo*/      "sd0_rec",
/*name*/      "sd0",
/*storage_path*/  "storage/sd0",
/*bs_storage_path*/  NULL,
/*root_path*/    "storage/sd0/C/"REC_FOLDER_NAME,
/*bs_root_path*/  NULL,
/*fs_type*/      "fat"
},
//sd1 录音文件夹分区
{
/*logo*/      "sd1_rec",
/*name*/      "sd1",
/*storage_path*/  "storage/sd1",
```



```
/*bs_storage_path*/    NULL,
/*root_path*/          "storage/sd1/C/"REC_FOLDER_NAME,
/*bs_root_path*/       NULL,
/*fs_type*/            "fat"
},
//u 盘录音文件夹分区
{
/*logo*/              "udisk0_rec",
/*name*/              "udisk0",
/*storage_path*/       "storage/udisk0",
/*bs_storage_path*/    NULL,
/*root_path*/          "storage/udisk0/C/"REC_FOLDER_NAME,
/*bs_root_path*/       NULL,
/*fs_type*/            "fat"
},
//外挂 fat 分区
{
/*logo*/              "fat_nor",
/*name*/              "fat_nor",
/*storage_path*/       "storage/fat_nor",
/*bs_storage_path*/    NULL,
/*root_path*/          "storage/fat_nor/C/",
/*bs_root_path*/       NULL,
/*fs_type*/            "fat"
},
//外挂 flash 资源分区
{
/*logo*/              "res_nor",
/*name*/              "res_nor",
/*storage_path*/       "storage/res_nor",
/*bs_storage_path*/    NULL,
/*root_path*/          "storage/res_nor/C/",
/*bs_root_path*/       NULL,
/*fs_type*/            "nor_sdfile"
},
///外挂录音分区
{
/*logo*/              "rec_nor",
/*name*/              "rec_nor",
/*storage_path*/       "storage/rec_nor",
/*bs_storage_path*/    NULL,
/*root_path*/          "storage/rec_nor/C/",
/*bs_root_path*/       NULL,
```

```

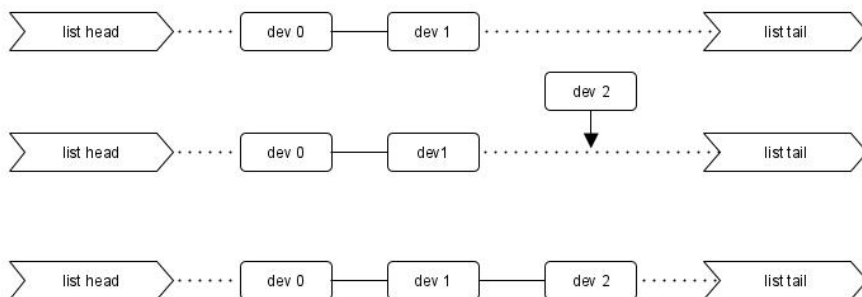
        /*fs_type*/          "nor_fs"
    },
    {
        /*logo*/             "nor_ui",
        /*name*/              "nor_ui",
        /*storage_path*/      "storage/nor_ui",
        /*bs_storage_path*/    NULL,
        /*root_path*/          "storage/nor_ui/C/",
        /*bs_root_path*/       NULL,
        /*fs_type*/            "nor_sdfile"
    },

    //<新加设备参数请在 reg end 前添加!!
    //<reg end
    {
        /*logo*/              NULL,
        /*name*/               NULL,
        /*storage_path*/       NULL,
        /*bs_storage_path*/     NULL,
        /*root_path*/          NULL,
        /*bs_root_path*/       NULL,
        /*fs_type*/            NULL
    },
};

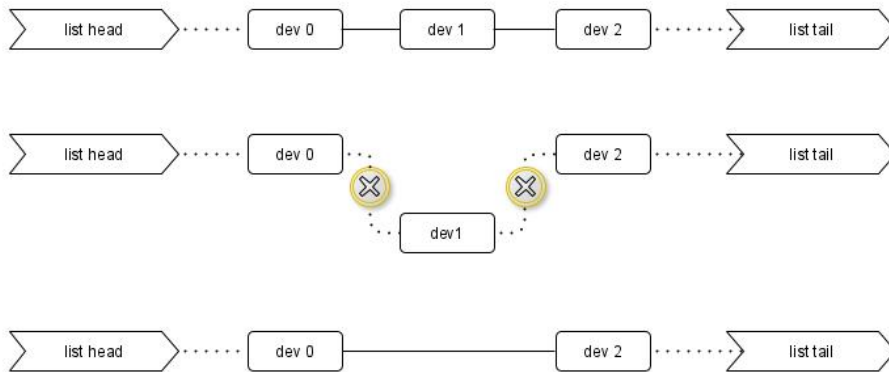
```

4. 设备管理流程框架

dev add:



dev del:



5. 如何将自定义设备加入到设备链表

具体操作步骤:

(1) 配置 dev_reg, 加入自定义设备信息项, 包括:

- 1) 设备逻辑盘符 logo, 设备匹配及查找时使用
- 2) dev_name 设备名称, 设备上线时, 底层内部会匹配设备链表 (在对应板卡.c 文件中配置设备, 如下图), 如找不到对应的设备, 上线失败。

```
REGISTER_DEVICES(device_table) = {
    /* { "audio", &audio_dev_ops, (void *) &audio_data }, */

    /* #if TCFG_CHARGE_ENABLE */
    /* { "charge", &charge_dev_ops, (void *)&charge_data }, */
    /* #endif */

    #if TCFG_SD0_ENABLE
        { "sd0", &sd_dev_ops, (void *) &sd0_data},
    #endif

    #if TCFG_SD1_ENABLE
        { "sd1", &sd_dev_ops, (void *) &sd1_data},
    #endif

    #if TCFG_LINEIN_ENABLE
        { "linein", &linein_dev_ops, (void *)&linein_data},
    #endif
    #if TCFG_OTG_MODE
        { "otg", &usb_dev_ops, (void *) &otg_data},
    #endif
    #if TCFG_UDISK_ENABLE
        { "udisk0", &mass_storage_ops, NULL},
    #endif
    #if TCFG_RTC_ENABLE
        { "rtc", &rtc_dev_ops, (void *)&rtc_data},
    #endif
    #if TCFG_NORFLASH_DEV_ENABLE
        { "norflash", &norflash_dev_ops, (void *)&norflash_dev_data},
    #if TCFG_NOR_TONE
        { "nor_tone", &norflash_dev_ops, (void *)&norflash_dev_data},
    #endif
    #if TCFG_NOR_FS_ENABLE
        { "nor_fs", &norflash_dev_ops, (void *)&norflash_dev_data},
    #endif
    #endif
};
```

- 3) storage_path、bs_storage_path 设备路径, 设备 mount 底层内部使用, 其中如果没有文件浏览需求可以将对应设备的 bs_storage_path 设置为 NULL。

- 4) root_path、bs_root_path 设备扫盘 (scan_disk)、文件访问时使用(fopen)及其他文件操作, 如果没有文件浏览 bs_root_path 可以设置为 NULL。
- 5) fs_type 文件系统类型, 例如: fat/sdfile 等
- (2) 常驻设备不需要检测的, 可以在 dev_manager_task 函数内直接 add 到设备链表, 如:

```
static void dev_manager_task(void *p)
{
    int res = 0;
    int msg[8] = {0};
    ///过滤隐藏 和 .开头名名的文件
    hidden_file(1);
    ///初始化设备管理链表
    os_mutex_create(&__this->mutex);
    ///设备初始化,
    devices_init();

    #if SDFILE_STORAGE && TCFG_CODE_FLASH_ENABLE
        dev_manager_add(SDFILE_DEV);
    #endif
}
```

- (3) 需要检测设备上下线事件响应, 参考 dev_status.c 中的 dev_status_event_filter 处理流程
- (4) 设备选择, 在指定情景, 通过以下这些接口选择设备

```
dev_manager_find_first
dev_manager_find_last
dev_manager_find_prev
dev_manager_find_next
dev_manager_find_active
dev_manager_find_spec
dev_manager_find_by_index
```

- (5) 设置设备信息及获取设备信息
- (6) **注意:** 支持上下线的设备, 要时刻留意设备的生命周期, 切记不能在设备从设备链表 del 之后还继续使用。

6. 详细接口说明

```
/**-----*/
/**@brief    设备增加接口
    @param    logo:逻辑盘符, 如: sd0/sd1/udisk0 等
    @return    0:成功, 非 0 是失败
    @note
*/
/**-----*/
```

int dev_manager_add(char *logo)

```
/**-----*/
/**@brief    设备删除接口
    @param    logo:逻辑盘符，如：sd0/sd1/udisk0 等
    @return    0:成功，非 0 是失败
    @note
*/
/*-----*/
```

int dev_manager_del(char *logo)

```
/**-----*/
/**@brief    通过设备节点检查设备是否在线
    @param    dev:设备节点
    @return    成功返回设备节点， 失败返回 NULL
    @note    通过设备节点检查设备是否在设备链表中
*/
/*-----*/
```

struct __dev *dev_manager_check(struct __dev *dev)

```
/**-----*/
/**@brief    通过盘符检查设备是否在线
    @param    logo:逻辑盘符，如:sd0/sd1/udisk0
    @return    成功返回设备节点， 失败返回 NULL
    @note    通过设备节点检查设备是否在设备链表中
*/
/*-----*/
```

struct __dev *dev_manager_check_by_logo(char *logo)

```
/**-----*/
/**@brief    获取设备总数
    @param    valid:
                1: 有效可播放设备, 0;所有设备,包括有可播放设备及无可播放设备
    @return    设备总数
    @note    根据使用情景决定接口参数
*/
/*-----*/
```

u32 dev_manager_get_total(u8 valid)

```
/**-----*/
/**@brief  获取设备列表第一个设备
    @param  valid:
                1: 有效可播放设备, 0;所有设备,包括有可播放设备及无可播放设备
    @return  成功返回设备设备节点,失败返回 NULL
    @note    根据使用情景决定接口参数
*/
/*-----*/
```

struct __dev *dev_manager_find_first(u8 valid)

```
/**-----*/
/**@brief  获取设备列表最后一个设备
    @param  valid:
                1: 有效可播放设备中查找
                0: 所有设备,包括有可播放设备及无可播放设备中查找
    @return  成功返回设备设备节点,失败返回 NULL
    @note    根据使用情景决定接口参数
*/
/*-----*/
```

struct __dev *dev_manager_find_last(u8 valid)

```
/**-----*/
/**@brief  获取上一个设备节点
    @param
        dev:当前设备节点
        valid:
                1: 有效可播放设备中查找,
                0: 所有设备,包括有可播放设备及无可播放设备中查找
    @return  成功返回设备设备节点,失败返回 NULL
    @note    根据当前设置的参数设备节点,找链表中的上一个设备
*/
/*-----*/
```

struct __dev *dev_manager_find_prev(struct __dev *dev, u8 valid)

```
/**-----*/
/**@brief  获取下一个设备节点
```

版权所有，侵权必究

14

```
@param
    dev:当前设备节点
    valid:
        1: 有效可播放设备中查找,
        0: 所有设备,包括有可播放设备及无可播放设备中查找
@return 成功返回设备设备节点,失败返回 NULL
@note 根据当前设置的参数设备节点, 找链表中的下一个设备
*/
/*-----*/
```

struct __dev *dev_manager_find_next(struct __dev *dev, u8 valid)

```
/*-----*/
/**@brief 获取当前活动设备节点
@param
    valid:
        1: 有效可播放设备中查找,
        0: 所有设备,包括有可播放设备及无可播放设备中查找
@return 成功返回设备设备节点,失败返回 NULL
@note
*/
/*-----*/
```

struct __dev *dev_manager_find_active(u8 valid)

```
/*-----*/
/**@brief 获取指定设备节点
@param
    logo: 指定逻辑盘符, 如: sd0/sd1/udisk0
    valid:
        1: 有效可播放设备中查找,
        0: 所有设备,包括有可播放设备及无可播放设备中查找
@return 成功返回设备设备节点,失败返回 NULL
@note
*/
/*-----*/
```

struct __dev *dev_manager_find_spec(char *logo, u8 valid)

```
/*-----*/
/**@brief 获取指定序号设备节点
```


@param

index: 指定序号, 指的是在设备链表中的顺序

valid:

1: 有效可播放设备中查找,

0: 所有设备,包括有可播放设备及无可播放设备中查找

@return 成功返回设备设备节点,失败返回 NULL

@note

*/

/*-----*/

struct __dev *dev_manager_find_by_index(u32 index, u8 valid)

/*-----*/

/**@brief 设备扫描释放

@param

fsn: 扫描句柄

@return 无

@note

*/

/*-----*/

void dev_manager_scan_disk_release(struct vfscan *fsn)

/*-----*/

/**@brief 设备扫描

@param

dev: 设备节点

path: 指定扫描目录

parm: 扫描参数, 包括文件后缀等

cycle_mode: 播放循环模式

callback: 扫描打断回调

@return 成功返回扫描控制句柄, 失败返回 NULL

@note

*/

/*-----*/

struct vfscan *dev_manager_scan_disk(struct __dev *dev, const char *path, const char *parm,
u8 cycle_mode, int (*callback)(void))

/*-----*/

/**@brief 通过设备节点标记指定设备是否有效

版权所有, 侵权必究

@param

dev: 设备节点

flag:

1: 设备有效,

0: 设备无效

@return

@note 这里有无效是指是否有可播放文件

*/

/*-----*/

void dev_manager_set_valid(struct __dev *dev, u8 flag)

/*-----*/

/**@brief 通过逻辑盘符标记指定设备是否有效

@param

logo: 逻辑盘符, 如: sd0/sd1/udisk0

flag:

1: 设备有效,

0: 设备无效

@return

@note 这里有无效是指是否有可播放文件

*/

/*-----*/

void dev_manager_set_valid_by_logo(char *logo, u8 flag)

/*-----*/

/**@brief 激活指定设备节点的设备

@param

dev: 设备节点

@return

@note 该接口可以将设备变为最新活动设备

*/

/*-----*/

void dev_manager_set_active(struct __dev *dev)

/*-----*/

/**@brief 激活指定逻辑盘符的设备

@param

logo: 逻辑盘符, 如: sd0/sd1/udisk0

@return

@note 该接口可以将设备变为最新活动设备

*/

/*-----*/

void dev_manager_set_active_by_logo(char *logo)

/*-----*/

/**@brief 获取设备节点的逻辑盘符

@param

dev: 设备节点

@return 成功返回逻辑盘符, 如: sd0/sd1/udisk0, 失败返回 NULL

@note

*/

/*-----*/

char *dev_manager_get_logo(struct __dev *dev)

/*-----*/

/**@brief 获取物理设备节点的逻辑盘符(去掉_rec 后缀)

@param

dev: 设备节点

@return 成功返回逻辑盘符, 如: sd0/sd1/udisk0, 失败返回 NULL

@note 物理逻辑盘符是指非录音文件夹设备盘符(录音文件夹设备如: sd0_rec)

*/

/*-----*/

char *dev_manager_get_phy_logo(struct __dev *dev)

/*-----*/

/**@brief 获取录音文件夹设备节点的逻辑盘符(追加_rec 后缀)

@param

dev: 设备节点

@return 成功返回逻辑盘符, 如: sd0_rec/sd1_rec/udisk0_rec, 失败返回 NULL

@note

*/

/*-----*/

char *dev_manager_get_rec_logo(struct __dev *dev)

```
/*-----*/  
/**@brief 通过设备节点获取设备文件系统根目录  
    @param  
        dev: 设备节点  
    @return 成功返回根目录，失败返回 NULL  
    @note  
*/  
/*-----*/
```

char *dev_manager_get_root_path(struct __dev *dev)

```
/*-----*/  
/**@brief 通过逻辑盘符获取设备文件系统根目录  
    @param  
        logo: 逻辑盘符，如：sd0/sd1/udisk0  
    @return 成功返回根目录，失败返回 NULL  
    @note  
*/  
/*-----*/
```

char *dev_manager_get_root_path_by_logo(char *logo)

```
/*-----*/  
/**@brief 通过设备节点获取设备文件系统文件浏览根目录  
    @param  
        dev: 设备节点  
    @return 成功返回根目录，失败返回 NULL  
    @note  
*/  
/*-----*/
```

char *dev_manager_get_bs_root_path(struct __dev *dev)

```
/*-----*/  
/**@brief 通过逻辑盘符获取设备文件系统文件浏览根目录  
    @param  
        logo: 逻辑盘符，如：sd0/sd1/udisk0  
    @return 成功返回根目录，失败返回 NULL  
    @note  
*/  
/*-----*/
```

char *dev_manager_get_bs_root_path_by_logo(char *logo)

```
/**-----*/
/**@brief 通过逻辑盘符判断设备是否在线
    @param
        logo: 逻辑盘符, 如: sd0/sd1/udisk0
        valid:
            1: 检查有效可播放设备
            0: 检查所有设备
    @return 1: 在线 0: 不在线
    @note
*/
/**-----*/
```

int dev_manager_online_check_by_logo(char *logo, u8 valid)

```
/**-----*/
/**@brief 通过设备节点判断设备是否在线
    @param
        dev: 设备节点
        valid:
            1: 检查有效可播放设备
            0: 检查所有设备
    @return 1: 在线 0: 不在线
    @note
*/
/**-----*/
```

int dev_manager_online_check(struct __dev *dev, u8 valid)

```
/**-----*/
/**@brief 设备检测线程处理
    @param
    @return
    @note
*/
/**-----*/
```

static void dev_manager_task(void *p)

```
/**-----*/  
/**@brief 设备管理器初始化  
    @param  
    @return  
    @note  
*/  
/*-----*/
```

```
void dev_manager_init(void)
```