

amplayerMy Project

Main Page	Classes	Files
Class List	Class Index	

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

audio_tag_info	
callback_t	
drm_info	
hwbufstats_t	
maudio_info_t	
media_info_t	
mstream_info_t	
msub_info_t	
mvideo_info_t	
pid_info	
play_control_t	
player_file_type	
player_info	

Generated on Tue Dec 2 2014 21:55:21 for amplayerMy Project by
[doxygen](#) 1.7.6.1

amplayerMy Project

[Main Page](#)[Classes](#)[Files](#)[Class List](#)[Class Index](#)

Public Attributes

audio_tag_info Struct Reference

List of all members.

Public Attributes

```
char title [512]
char author [512]
char album [512]
char comment [512]
char year [4]
int track
char genre [32]
char copyright [512]
audio_cover_type pic
```

Detailed Description

Definition at line [116](#) of file [player_type.h](#).

The documentation for this struct was generated from the following file:

- [player_type.h](#)
-

Generated on Tue Dec 2 2014 21:55:18 for amplayerMy Project by
[doxygen](#) 1.7.6.1

amplayerMy Project

Main Page

Classes

Files

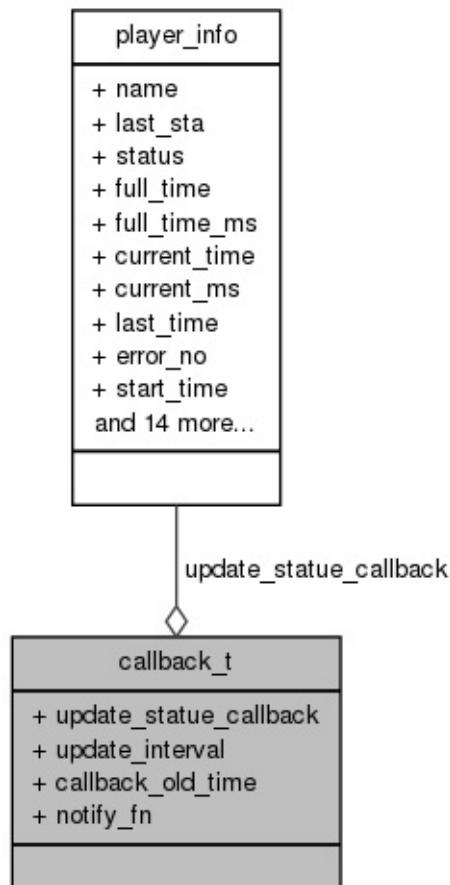
Class List

Class Index

Public Attributes

callback_t Struct Reference

Collaboration diagram for callback_t:



[legend]

List of all members.

Public Attributes

update_state_fun_t	update_statue_callback
int	update_interval
long	callback_old_time
notify_callback	notify_fn

Detailed Description

Definition at line [265](#) of file [player_type.h](#).

The documentation for this struct was generated from the following file:

- [player_type.h](#)
-

Generated on Tue Dec 2 2014 21:55:19 for amplayerMy Project by
[doxygen](#) 1.7.6.1

amplayerMy Project

[Main Page](#)

[Classes](#)

[Files](#)

[Class List](#)

[Class Index](#)

[Public Attributes](#)

drm_info Struct Reference

List of all members.

Public Attributes

drm_level_t	drm_level
int	drm_flag
int	drm_hasesdata
int	drm_priv
unsigned int	drm_pktsize
unsigned int	drm_pktptr
unsigned int	drm_phy
unsigned int	drm_vir
unsigned int	drm_remap
int	data_offset
int	extpad [8]

Detailed Description

Definition at line [77](#) of file [player_type.h](#).

The documentation for this struct was generated from the following file:

- [player_type.h](#)
-

Generated on Tue Dec 2 2014 21:55:19 for amplayerMy Project by
[doxygen](#) 1.7.6.1

amplayerMy Project

[Main Page](#)[Classes](#)[Files](#)[Class List](#)[Class Index](#)

Public Attributes

hwbufstats_t Struct Reference

List of all members.

Public Attributes

int **vbufused**

int **vbufsize**

int **vdatasize**

int **abufused**

int **abufsize**

int **adatasize**

int **sbufused**

int **sbufsize**

int **sdatasize**

Detailed Description

Definition at line [251](#) of file [player_type.h](#).

The documentation for this struct was generated from the following file:

- [player_type.h](#)
-

Generated on Tue Dec 2 2014 21:55:19 for amplayerMy Project by
[doxygen](#) 1.7.6.1

amplayerMy Project

Main Page

Classes

Files

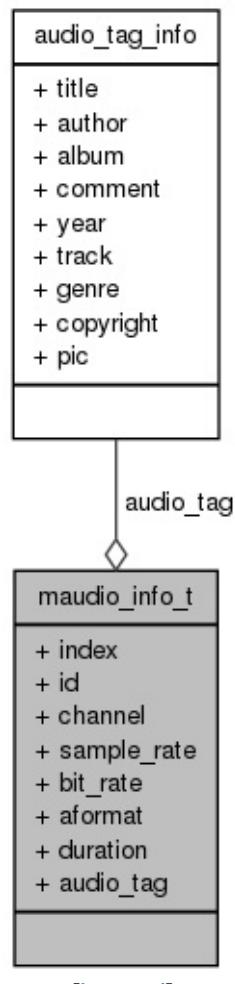
Class List

Class Index

Public Attributes

maudio_info_t Struct Reference

Collaboration diagram for maudio_info_t:



[legend]

List of all members.

Public Attributes

int	index
int	id
int	channel
int	sample_rate
int	bit_rate
aformat_t	aformat
int	duration
audio_tag_info * audio_tag	

Detailed Description

Definition at line [129](#) of file [player_type.h](#).

The documentation for this struct was generated from the following file:

- [player_type.h](#)
-

Generated on Tue Dec 2 2014 21:55:19 for amplayerMy Project by
[doxygen](#) 1.7.6.1

amplayerMy Project

[Main Page](#)

[Classes](#)

[Files](#)

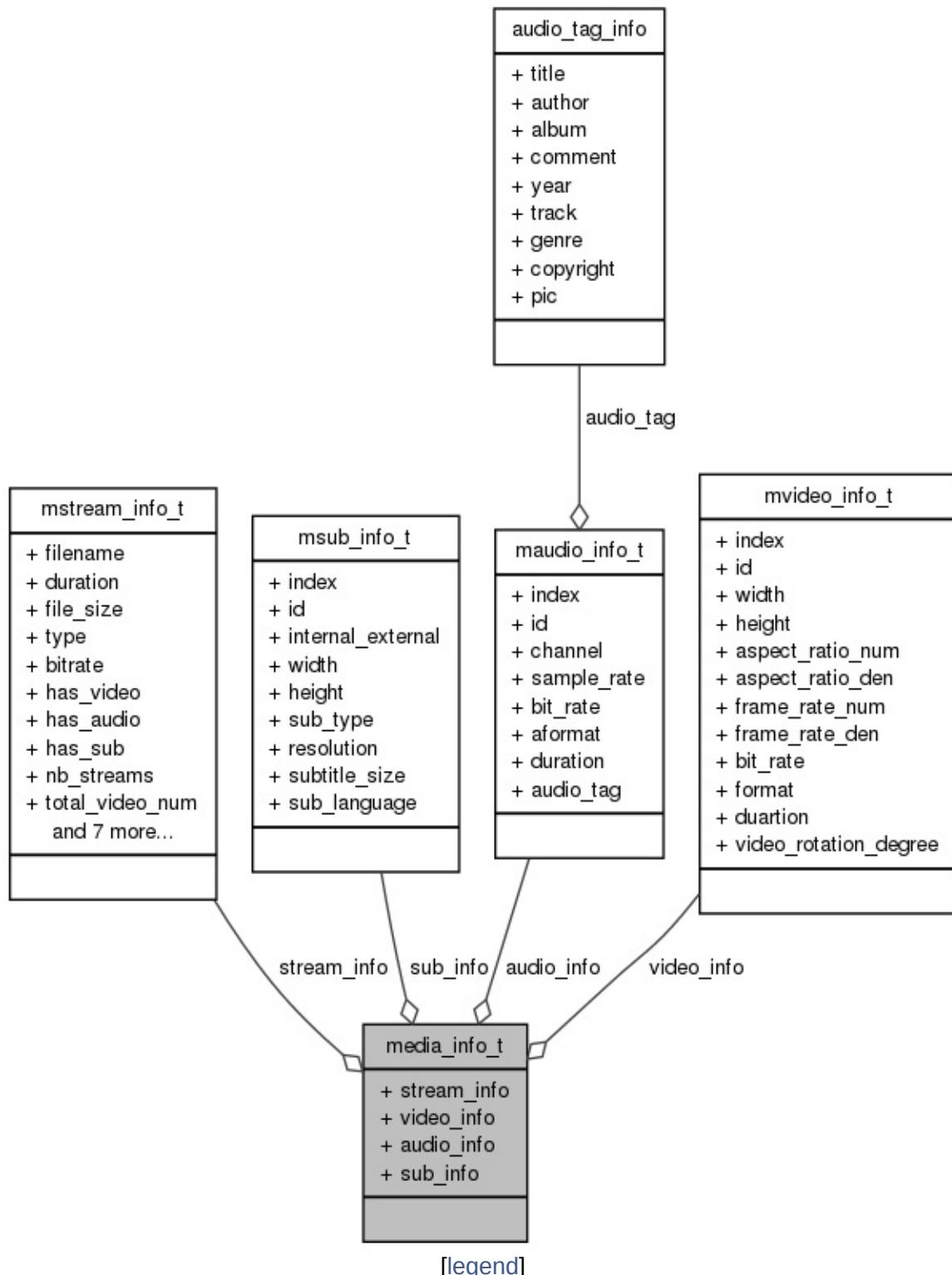
[Class List](#)

[Class Index](#)

[Public Attributes](#)

media_info_t Struct Reference

Collaboration diagram for media_info_t:



[legend]

List of all members.

Public Attributes

<code>mstream_info_t stream_info</code>
<code>mvideo_info_t * video_info [MAX_VIDEO_STREAMS]</code>
<code>maudio_info_t * audio_info [MAX_AUDIO_STREAMS]</code>
<code>msub_info_t * sub_info [MAX_SUB_STREAMS]</code>

Detailed Description

Definition at line [176](#) of file [player_type.h](#).

The documentation for this struct was generated from the following file:

- [player_type.h](#)
-

Generated on Tue Dec 2 2014 21:55:20 for amplayerMy Project by
[doxygen](#) 1.7.6.1

amplayerMy Project

[Main Page](#)

[Classes](#)

[Files](#)

[Class List](#)

[Class Index](#)

[Public Attributes](#)

mstream_info_t Struct Reference

List of all members.

Public Attributes

char *	filename
int	duration
long long	file_size
pfile_type	type
int	bitrate
int	has_video
int	has_audio
int	has_sub
int	nb_streams
int	total_video_num
int	cur_video_index
int	total_audio_num
int	cur_audio_index
int	total_sub_num
int	cur_sub_index
int	seekable
int	drm_check
int	adif_file_flag

Detailed Description

Definition at line [154](#) of file [player_type.h](#).

The documentation for this struct was generated from the following file:

- [player_type.h](#)
-

Generated on Tue Dec 2 2014 21:55:20 for amplayerMy Project by
[doxygen](#) 1.7.6.1

amplayerMy Project

[Main Page](#)

[Classes](#)

[Files](#)

[Class List](#)

[Class Index](#)

[Public Attributes](#)

msub_info_t Struct Reference

List of all members.

Public Attributes

int	index
char	id
char	internal_external
unsigned short	width
unsigned short	height
unsigned int	sub_type
char	resolution
long long	subtitle_size
char *	sub_language

Detailed Description

Definition at line [141](#) of file [player_type.h](#).

The documentation for this struct was generated from the following file:

- [player_type.h](#)
-

Generated on Tue Dec 2 2014 21:55:20 for amplayerMy Project by
[doxygen](#) 1.7.6.1

amplayerMy Project

[Main Page](#)[Classes](#)[Files](#)[Class List](#)[Class Index](#)

Public Attributes

mvideo_info_t Struct Reference

List of all members.

Public Attributes

```
int index
int id
int width
int height
int aspect_ratio_num
int aspect_ratio_den
int frame_rate_num
int frame_rate_den
int bit_rate
vformat_t format
int duartion
unsigned int video_rotation_degree
```

Detailed Description

Definition at line [93](#) of file [player_type.h](#).

The documentation for this struct was generated from the following file:

- [player_type.h](#)
-

Generated on Tue Dec 2 2014 21:55:20 for amplayerMy Project by
[doxygen](#) 1.7.6.1

amplayerMy Project

[Main Page](#)[Classes](#)[Files](#)[Class List](#)[Class Index](#)

Public Attributes

pid_info Struct Reference

List of all members.

Public Attributes

```
int num
int pid [MAX_PLAYER_THREADS]
```

Detailed Description

Definition at line [214](#) of file [player_type.h](#).

The documentation for this struct was generated from the following file:

- [player_type.h](#)
-

Generated on Tue Dec 2 2014 21:55:20 for amplayerMy Project by
[doxygen](#) 1.7.6.1

amplayerMy Project

[Main Page](#)

[Classes](#)

[Files](#)

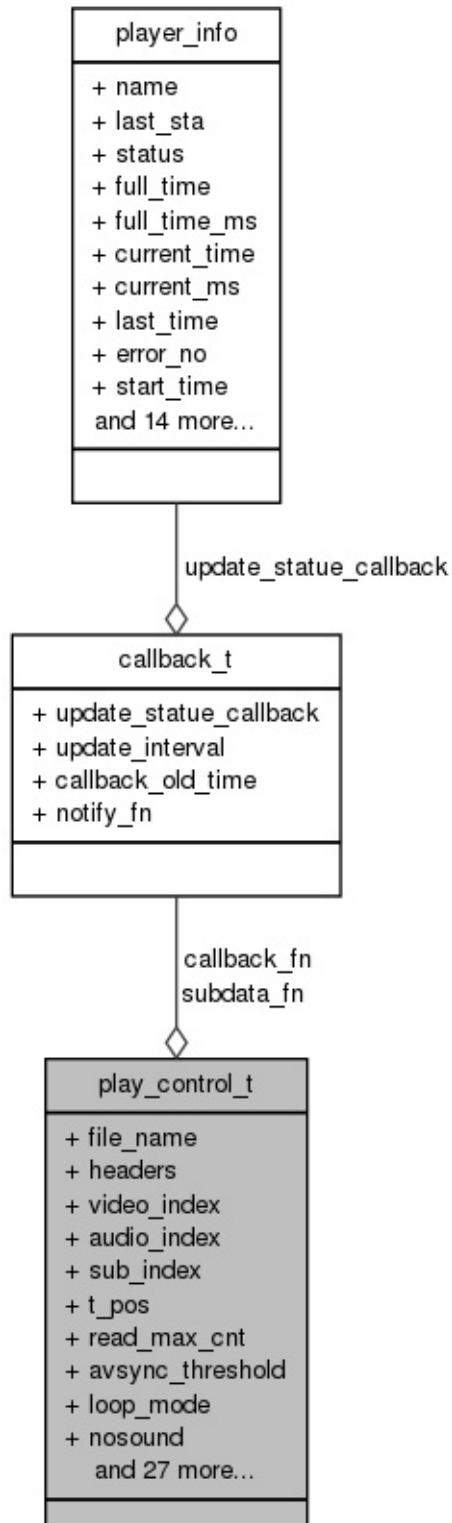
[Class List](#)

[Class Index](#)

Public Attributes

play_control_t Struct Reference

Collaboration diagram for play_control_t:



[legend]

List of all members.

Public Attributes

```
char * file_name
char * headers
int video_index
int audio_index
int sub_index
float t_pos
int read_max_cnt
int avsync_threshold

union {
    struct {
        unsigned int loop_mode:1
        unsigned int nosound:1
        unsigned int novideo:1
        unsigned int hassub:1
        unsigned int need_start:1
        unsigned int displast_frame: 1
    }
    int mode
};

callback_t callback_fn
callback_t subdata_fn
void * subhd
int subdatasource
int byteiobufsize
int loopbufsize
int enable_rw_on_pause
int auto_buffing_enable
float buffering_min
float buffering_middle
float buffering_max
int is_playlist
```

```
int is_type_parser
int is_livemode
int buffинг_starttime_s
int buffинг_force_delay_s
int lowbuffermode_flag
int lowbuffermode_limited_ms
int is_ts_soft_demux
int reserved [56]
int SessionID
int t_duration_ms
```

Detailed Description

Definition at line [273](#) of file [player_type.h](#).

The documentation for this struct was generated from the following file:

- [player_type.h](#)
-

Generated on Tue Dec 2 2014 21:55:20 for amplayerMy Project by
[doxygen](#) 1.7.6.1

amplayerMy Project

[Main Page](#)

[Classes](#)

[Files](#)

[Class List](#)

[Class Index](#)

[Public Attributes](#)

player_file_type Struct Reference

List of all members.

Public Attributes

```
const char * fmt_string  
    int video_tracks  
    int audio_tracks  
    int subtitle_tracks
```

Detailed Description

Definition at line [220](#) of file [player_type.h](#).

The documentation for this struct was generated from the following file:

- [player_type.h](#)
-

Generated on Tue Dec 2 2014 21:55:20 for amplayerMy Project by
[doxygen](#) 1.7.6.1

amplayerMy Project

[Main Page](#)

[Classes](#)

[Files](#)

[Class List](#)

[Class Index](#)

[Public Attributes](#)

player_info Struct Reference

List of all members.

Public Attributes

```
char * name
player_status last_sta
player_status status
    int full_time
    int full_time_ms
    int current_time
    int current_ms
    int last_time
    int error_no
int64_t start_time
int64_t first_time
    int pts_video
unsigned int current_pts
    long curtime_old_time
unsigned int video_error_cnt
unsigned int audio_error_cnt
    float audio_bufferlevel
    float video_bufferlevel
int64_t bufed_pos
    int bufed_time
unsigned int drm_rental
    int64_t download_speed
unsigned int last_pts
    int seek_point
    int seek_delay
```

Detailed Description

Definition at line [184](#) of file [player_type.h](#).

The documentation for this struct was generated from the following file:

- [player_type.h](#)
-

Generated on Tue Dec 2 2014 21:55:20 for amplayerMy Project by
[doxygen](#) 1.7.6.1

amplayerMy Project

Main Page

Classes

Files

Class List

Class Index

Class Index

A | C | D | H | M | P

A

[audio_tag_info](#)

C

[callback_t](#)

D

[drm_info](#)

H

[hwbufstats_t](#)

M

[maudio_info_t](#)

[media_info_t](#)

[mstream_info_t](#)

[msub_info_t](#)
[mvideo_info_t](#)

P

[pid_info](#)

A | C | D | H | M | P

Generated on Tue Dec 2 2014 21:55:21 for amplayerMy Project by

[doxygen](#) 1.7.6.1

amplayerMy Project

Main Page

Classes

Files

File List

File Members

File List

Here is a list of all documented files with brief descriptions:

[player.h](#) [code]

[player_ctrl.c](#) [code]

[player_id.h](#) [code]

[player_type.h](#) [code]

Generated on Tue Dec 2 2014 21:55:21 for amplayerMy Project by

[doxygen](#) 1.7.6.1

amplayer My Project

Main Page

Classes

Files

File List

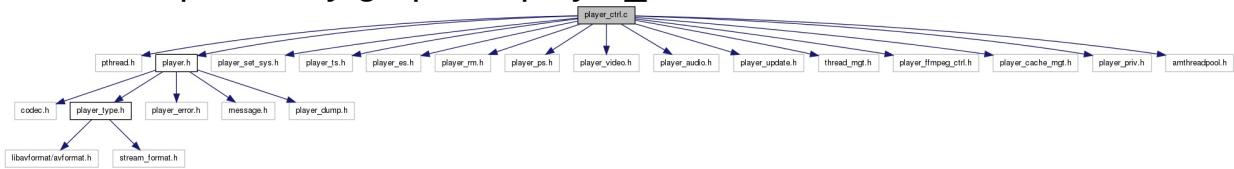
File Members

Defines | Functions

player_ctrl.c File Reference

```
#include <pthread.h> #include <player.h>
#include <player_set_sys.h>
#include "player_ts.h"
#include "player_es.h"
#include "player_rm.h"
#include "player_ps.h"
#include "player_video.h"
#include "player_audio.h"
#include "player_update.h"
#include "thread_mgt.h"
#include "player_ffmpeg_ctrl.h"
#include "player_cache_mgt.h"
#include "player_priv.h"
#include <amthreadpool.h>
```

Include dependency graph for player_ctrl.c:



Go to the source code of this file.

Defines

```
#define FBIOPUT OSD SRCCOLORKEY 0x46fb
#define FBIOPUT OSD SRCKEY_ENABLE 0x46fa
```

Functions

void	print_version_info ()
int	player_init (void) Amlogic player initialization. Make sure call it once when setup amlogic player every time.
int	player_start (play_control_t *ctrl_p, unsigned long priv) Amlogic player start to play a specified path streaming file.
int	player_start_play (int pid) if need_start set 1, call player_start_play to start playback
int	player_stop (int pid) send stop command to player (synchronous)
int	player_stop_async (int pid) send stop command to player (asynchronous)
int	player_exit (int pid) release player resource
int	player_pause (int pid) send pause command to player
int	player_resume (int pid) send resume command to player
int	player_loop (int pid) send loop command to set loop play current file
int	player_noloop (int pid) send noloop command to cancel loop play
int	player_timeSearch (int pid, float s_time) seek to designated time point to play.
int	player_forward (int pid, int speed) send fastforward command to player
int	player_backward (int pid, int speed) send fast backward command to player.

	int	player_aid (int pid, int audio_id) switch audio stream to designed id audio stream.
	int	player_sid (int pid, int sub_id) send switch subtitle id command to player
	int	player_enable_autobuffer (int pid, int enable) enable/disable auto buffering
	int	player_set_autobuffer_level (int pid, float min, float middle, float max) player_set_autobuffer_level
	int	player_send_message (int pid, player_cmd_t *cmd) send message to player thread
	int	player_register_update_callback (callback_t *cb, update_state_fun_t up_fn, int interval_s) App can register a update callback function into player.
player_status		player_get_state (int pid) get player current state
unsigned int		player_get_extern_priv (int pid) get current player's unique identification
	int	player_get_play_info (int pid, player_info_t *info) get player's information
int64_t		player_get_lpbuffbuffedsize (int pid) get player current lpbuffbuffedsize
int64_t		player_get_streambufbuffedsize (int pid) get player current streambufbuffedsize
	int	player_get_media_info (int pid, media_info_t *minfo) get file media information
	int	player_video_overlay_en (unsigned enable) enable osd colorkey
	int	audio_set_mute (int pid, int mute_on) volume mute switch
	int	audio_get_volume_range (int pid, float *min, float *max)
		get volume range

int	audio_set_volume (int pid, float val)	
	set val to volume	
int	audio_get_volume (int pid, float *vol)	
	get volume	
int	audio_set_lrvolume (int pid, float lvol, float rvol)	
	set left and right volume	
int	audio_get_lrvolume (int pid, float *lvol, float *rvol)	
	get left/right volume	
int	audio_set_volume_balance (int pid, int balance)	
	switch balance	
int	audio_swap_left_right (int pid)	
	swap left and right channel	
int	audio_left_mono (int pid)	
int	audio_right_mono (int pid)	
	audio_right_mono	
int	audio_stereo (int pid)	
int	audio_lr_mix_set (int pid, int enable)	
	+ * + *	
int	audio_cur_pcmpara_Applied_get (int pid, int *pfs, int *pch)	
int	audio_set_spectrum_switch (int pid, int isStart, int interval)	
int	player_progress_exit (void)	
	used for all exit,please only call at this process fatal error.	
int	player_list_allpid (pid_info_t *pid)	
	list all alived player pid	
int	player_cache_system_init (int enable, const char *dir, int max_size, int block_size)	
	player_cache_system_init	
char *	player_status2str (player_status status)	
	convert player state value to string	
char *	player_value2str (char *key, int value)	

convert player state value to string

int **audio_get_decoder_enable** (int pid)

Detailed Description

Author:

Xu Hui <hui.xu@amlogic.com>

Version:

1.0.1

Date:

2012-01-19

Definition in file [player_ctrl.c](#).

Function Documentation

```
int audio_get_lrvolume( int pid,  
                        float * lvol,  
                        float * rvol  
)
```

get left/right volume

audio_get_lrvolume

Parameters:

[in] **pid** player tag which get from player_start return value

Returns:

r = 0 for success

lvol,rvol range : 0~1

Definition at line [1205](#) of file [player_ctrl.c](#).

```
int audio_get_volume( int pid,  
                      float * vol  
)
```

get volume

audio_get_volume

Parameters:

[in] **pid** player tag which get from player_start return value

Returns:

r = 0 success

vol range:0~1

Definition at line [1150](#) of file [player_ctrl.c](#).

```
int audio_get_volume_range( int pid,  
                            float * min,  
                            float * max  
                        )
```

get volume range

audio_get_volume_range

Parameters:

- [in] **pid** player tag which get from player_start return value
- [out] **min** volume minimum
- [out] **max** volume maximum

Returns:

PLAYER_SUCCESS success PLAYER_FAILED failed

0~1

Definition at line [1112](#) of file [player_ctrl.c](#).

```
int audio_left_mono( int pid )
```

audio_left_mono

Parameters:

- [in] **pid** player tag which get from player_start return value

Returns:

PLAYER_SUCCESS success PLAYER_FAILED failed

Definition at line [1270](#) of file `player_ctrl.c`.

```
int audio_lr_mix_set( int pid,  
                      int enable  
)
```

+ * + *

+ * audio_lr_mix + * + *

Parameters:

[in] **pid** player tag which get from player_start return value +
* + *

Returns:

PLAYER_SUCCESS success + * PLAYER_FAILED failed+ *

+

Definition at line [1368](#) of file `player_ctrl.c`.

```
int audio_right_mono( int pid )
```

audio_right_mono

audio_right_mono

Parameters:

[in] **pid** player tag which get from player_start return value

Returns:

PLAYER_SUCCESS success PLAYER_FAILED failed

Definition at line [1303](#) of file `player_ctrl.c`.

```
int audio_set_lrvolume( int pid,  
                        float lvol,  
                        float rvol  
)
```

set left and right volume

audio_set_lrvolume

Parameters:

- [in] **pid** player tag which get from player_start return value
- [in] **lval**,: left volume value
- [in] **rval**,: right volume value

Returns:

PLAYER_SUCCESS success PLAYER_FAILED failed

lvol,rvol range: 0~1

Definition at line [1176](#) of file [player_ctrl.c](#).

```
int audio_set_mute( int pid,  
                     int mute_on  
)
```

volume mute switch

audio_set_mute

Parameters:

- [in] **pid** player tag which get from player_start return value
- [in] **mute_on** volume mute flag 1:mute 0:inmute

Returns:

PLAYER_SUCCESS success PLAYER_FAILED failed

Definition at line [1072](#) of file [player_ctrl.c](#).

```
int audio_set_spectrum_switch( int pid,  
                               int isStart,  
                               int interval  
                             )
```

audio_set_spectrum_switch

Parameters:

- [in] **pid** player tag which get from player_start return value
- [in] **isStart** open/close spectrum switch function
- [in] **interval** swtich interval

Returns:

PLAYER_SUCCESS success PLAYER_FAILED failed

Definition at line [1424](#) of file [player_ctrl.c](#).

```
int audio_set_volume( int pid,  
                      float val  
                    )
```

set val to volume

audio_set_volume

Parameters:

- [in] **pid** player tag which get from player_start return value
- [in] **val** volume value

Returns:

PLAYER_SUCCESS success PLAYER_FAILED failed

val range: 0~1

Definition at line [1132](#) of file [player_ctrl.c](#).

```
int audio_set_volume_balance( int pid,  
                                int balance  
                            )
```

switch balance

audio_set_volume_balance

Parameters:

[in] **pid** player tag which get from player_start return value
[in] **balance** balance flag 1:set balance 0:cancel balance

Returns:

PLAYER_SUCCESS success PLAYER_FAILED failed

Definition at line [1232](#) of file [player_ctrl.c](#).

```
int audio_stereo( int pid )
```

audio_stereo

Parameters:

[in] **pid** player tag which get from player_start return value

Returns:

PLAYER_SUCCESS success PLAYER_FAILED failed

Definition at line [1336](#) of file `player_ctrl.c`.

`int audio_swap_left_right(int pid)`

swap left and right channel

`audio_swap_left_right`

Parameters:

[in] `pid` player tag which get from `player_start` return value

Returns:

`PLAYER_SUCCESS` success `PLAYER_FAILED` failed

Definition at line [1251](#) of file `player_ctrl.c`.

`int player_aid(int pid,
 int audio_id
)`

switch audio stream to designed id audio stream.

`player_aid`

Parameters:

[in] `pid` player tag which get from `player_start` return value

[in] `audio_id` target audio stream id, can find through `media_info` command

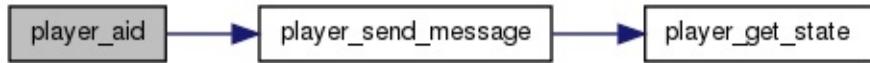
Returns:

`PLAYER_NOT_VALID_PID` playet tag invalid `PLAYER_NOMEM` alloc memory failed `PLAYER_SUCCESS` success

`audio_id` is audio stream index

Definition at line **604** of file [player_ctrl.c](#).

Here is the call graph for this function:



```
int player_backward ( int pid,  
                      int speed  
)
```

send fast backward command to player.

player_backward

Parameters:

- [in] **pid** player tag which get from player_start return value
- [in] **speed** fast backward step

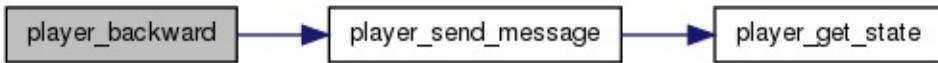
Returns:

PLAYER_NOT_VALID_PID playet tag invalid
PLAYER_NOMEM alloc memory failed
PLAYER_SUCCESS success

After fb, player playback from a key frame

Definition at line **569** of file [player_ctrl.c](#).

Here is the call graph for this function:



```
int player_cache_system_init ( int enable,  
                               const char * dir,  
                               int max_size,  
                               int block_size
```

)

player_cache_system_init

player_cache_system_init

Parameters:

- [in] **enable**
- [in] **dir**
- [in] **max_size**
- [in] **block_size**

Returns:

0;

Definition at line [1511](#) of file [player_ctrl.c](#).

```
int player_enable_autobuffer( int pid,  
                                int enable  
                            )
```

enable/disable auto buffering

player_enable_autobuffer

Parameters:

- [in] **pid** player tag which get from player_start return value
- [in] **enable** enable/disable auto buffer function

Returns:

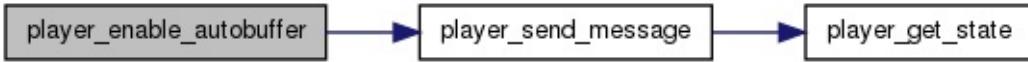
PLAYER_NOT_VALID_PID playet tag invalid
PLAYER_NOMEM alloc memory failed
PLAYER_SUCCESS success

if enable auto buffering, need set limit use

`player_set_autobuffer_level.`

Definition at line **675** of file **player_ctrl.c**.

Here is the call graph for this function:



int player_exit (int pid)

release player resource

`player_exit`

Parameters:

[**in**] **pid** player tag which get from `player_start` return value

Returns:

PLAYER_NOT_VALID_PID playet tag invalid
PLAYER_NOMEM alloc memory failed
PLAYER_SUCCESS success

`player_exit` must with pairs of `player_play`

Definition at line **330** of file **player_ctrl.c**.

Here is the call graph for this function:



int player_forward (int pid, int speed)

send fastforward command to player

`player_forward`

Parameters:

- [in] **pid** player tag which get from player_start return value
- [in] **speed** fast forward step

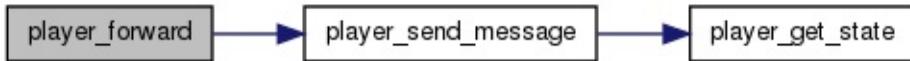
Returns:

PLAYER_NOT_VALID_PID playet tag invalid
PLAYER_NOMEM alloc memory failed
PLAYER_SUCCESS success

After ff, player playback from a key frame

Definition at line [535](#) of file **player_ctrl.c**.

Here is the call graph for this function:



unsigned int player_get_extern_priv (int pid)

get current player's unique identification

player_get_extern_priv

Parameters:

- [in] **pid** player tag which get from player_start return value

Returns:

externed player's unique identification
PLAYER_NOT_VALID_PID error,invalid pid

Definition at line [861](#) of file **player_ctrl.c**.

int64_t player_get_lpbuffbuffedszie (int pid)

get player current lpbuffbuffedszie

`player_get_lpbuffbuffedsize`

Parameters:

[in] **pid** player tag which get from `player_start` return value

Returns:

`plbuffbuffedsize;`

state defined in `player_type.h`

Definition at line 921 of file `player_ctrl.c`.

```
int player_get_media_info ( int pid,
                           media_info_t * minfo
                         )
```

get file media information

`player_get_media_info`

Parameters:

[in] **pid** player tag which get from `player_start` return value

[out] **minfo** media info structure pointer

Returns:

`PLAYER_SUCCESS` success
`PLAYER_NOT_VALID_PID`
error,invalid pid

get file media information, such as audio format, video format, etc.

Definition at line 980 of file `player_ctrl.c`.

Here is the call graph for this function:



```
int player_get_play_info ( int pid,  
                           player_info_t * info  
                         )
```

get player's information

player_get_play_info

Parameters:

- [in] **pid** player tag which get from player_start return value
- [out] **info** play info structure pointer

Returns:

PLAYER_SUCCESS success
PLAYER_NOT_VALID_PID
error,invalid pid

get playing information,status, current_time, buferlevel etc.

Definition at line [893](#) of file [player_ctrl.c](#).

player_status **player_get_state** (int **pid**)

get player current state

player_get_state

Parameters:

- [in] **pid** player tag which get from player_start return value

Returns:

status player current status
PLAYER_NOT_VALID_PID
error,invalid pid

state defined in [player_type.h](#)

Definition at line [831](#) of file [player_ctrl.c](#).

Referenced by [player_get_media_info\(\)](#), and [player_send_message\(\)](#).

int64_t player_get_streambufbuffedsize (int pid)

get player current streambufbuffedsize

player_get_streambufbuffedsize

Parameters:

[in] **pid** player tag which get from player_start return value

Returns:

streambufbuffedsize;

state defined in [player_type.h](#)

Definition at line [949](#) of file [player_ctrl.c](#).

int player_init (void)

Amlogic player initilization. Make sure call it once when setup amlogic player every time.

player_init

Parameters:

void

Returns:

PLAYER_SUCCESS success

register all formats and codecs; player id pool initilization; audio basic initilization; register support decoder(ts,es,rm,pure audio, pure video); keep last frame displaying for default; enable demux and set demux channel;

Definition at line [61](#) of file `player_ctrl.c`.

`int player_list_allpid (pid_info_t * pid)`

list all alive player pid

player_list_allpid

Parameters:

[out] `pid` pid list structure pointer

Returns:

`PLAYER_SUCCESS` success `PLAYER_FAILED` failed

support multiple player threads, but only one threads use hardware decoder

Definition at line [1477](#) of file `player_ctrl.c`.

`int player_loop (int pid)`

send loop command to set loop play current file

player_loop

Parameters:

[in] `pid` player tag which get from `player_start` return value

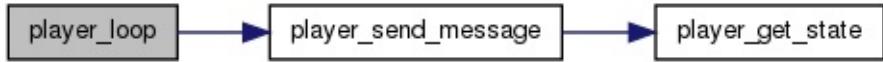
Returns:

`PLAYER_NOT_VALID_PID` playet tag invalid `PLAYER_NOMEM` alloc memory failed `PLAYER_SUCCESS` success

need set loop before stream play end

Definition at line [436](#) of file `player_ctrl.c`.

Here is the call graph for this function:



int **player_noloop** (int **pid**)

send noloop command to cancel loop play

player_noloop

Parameters:

[in] **pid** player tag which get from player_start return value

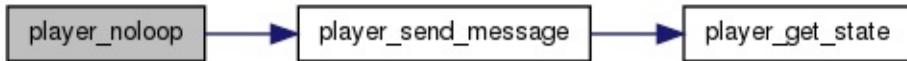
Returns:

PLAYER_NOT_VALID_PID playet tag invalid
PLAYER_NOMEM alloc memory failed
PLAYER_SUCCESS success

need cancel loop before stream play end

Definition at line **469** of file **player_ctrl.c**.

Here is the call graph for this function:



int **player_pause** (int **pid**)

send pause command to player

player_pause

Parameters:

[in] **pid** player tag which get from player_start return value

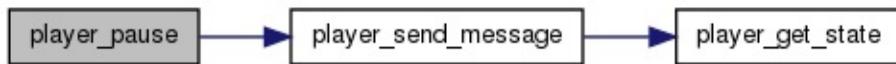
Returns:

PLAYER_NOT_VALID_PID playet tag invalid
PLAYER_NOMEM alloc memory failed
PLAYER_SUCCESS success

null

Definition at line [372](#) of file `player_ctrl.c`.

Here is the call graph for this function:



`int player_progress_exit(void)`

used for all exit,please only call at this process fatal error.

`player_progress_exit`

Returns:

`PLAYER_SUCCESS` success

Do not wait any things in this function

Definition at line [1455](#) of file `player_ctrl.c`.

`int player_register_update_callback(callback_t * cb,
 update_state_fun_t up_fn,
 int interval_)`

App can register a update callback function into player.

`player_register_update_callback`

Parameters:

- [in] `cb` callback structure point
- [in] `up_fn` update function
- [in] `interval_s` update interval (milliseconds)

Returns:

PLAYER_EMPTY_P invalid pointer
PLAYER_ERROR_CALLBACK up_fn invalid
PLAYER_SUCCESS success

used to update player status

Definition at line [804](#) of file `player_ctrl.c`.

int player_resume (int pid)

send resume command to player

`player_resume`

Parameters:

[in] **pid** player tag which get from `player_start` return value

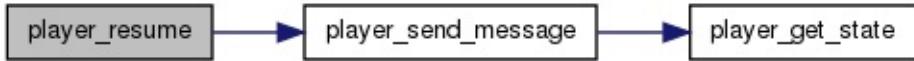
Returns:

PLAYER_NOT_VALID_PID playet tag invalid PLAYER_NOMEM
alloc memory failed PLAYER_SUCCESS success

`null`

Definition at line [404](#) of file `player_ctrl.c`.

Here is the call graph for this function:



**int player_send_message (int pid,
 player_cmd_t * cmd
)**

send message to player thread

`player_send_message`

Parameters:

- [in] **pid** player tag which get from `player_start` return value
- [in] **cmd** player control command

Returns:

PLAYER_NOT_VALID_PID playet tag invalid
PLAYER_NOMEM alloc memory failed
PLAYER_SUCCESS success

if player has exited, send message invalid

Definition at line [756](#) of file `player_ctrl.c`.

Referenced by `player_aid()`, `player_backward()`,
`player_enable_autobuffer()`, `player_forward()`, `player_loop()`,
`player_noloop()`, `player_pause()`, `player_resume()`,
`player_set_autobuffer_level()`, `player_sid()`, and
`player_timesearch()`.

Here is the call graph for this function:



```
int player_set_autobuffer_level ( int pid,  
                                    float min,  
                                    float middle,  
                                    float max  
)
```

`player_set_autobuffer_level`

`player_set_autobuffer_level`

Parameters:

- [in] **pid** player tag which get from `player_start` return

		value
[in]	min	buffer min percent (less than min, enter buffering, av pause)
[in]	middle	buffer middle percent(more than middle, exit buffering, av resume)
[in]	max	buffer max percent(more than max, do not feed data)

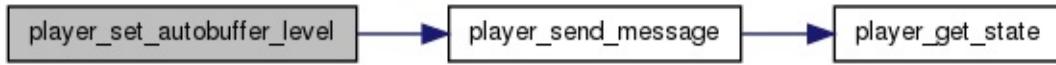
Returns:

PLAYER_NOT_VALID_PID playet tag invalid
 PLAYER_NOMEM alloc memory failed
 PLAYER_SUCCESS success

if buffer level low than min, player auto pause to buffer data, if buffer level high than middle, player auto resume playback

Definition at line [713](#) of file [player_ctrl.c](#).

Here is the call graph for this function:



```
int player_sid( int pid,
                int sub_id
            )
```

send switch subtitle id command to player

player_sid

Parameters:

[in]	pid	player tag which get from player_start return value
[in]	sub_id	target subtitle stream id, can find through media_info command

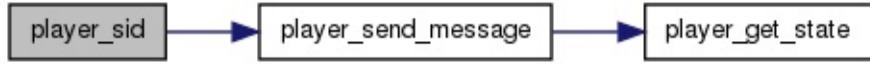
Returns:

PLAYER_NOT_VALID_PID playet tag invalid
PLAYER_NOMEM alloc memory failed
PLAYER_SUCCESS success

sub_id is subtitle stream index

Definition at line [640](#) of file [player_ctrl.c](#).

Here is the call graph for this function:



```
int player_start ( play_control_t * ctrl_p,  
                    unsigned long    priv  
                )
```

Amlogic player start to play a specified path streaming file.

player_start

Parameters:

[in] **ctrl_p** player control parameters structure pointer
[in] **priv** Player unique identification

Returns:

pid current player tag

request id for current player; if not set displast_frame, or change file ,set black out; creat player thread for playback;

Definition at line [98](#) of file [player_ctrl.c](#).

int player_start_play (int pid)

if need_start set 1, call player_start_play to start playback

player_start_play

Parameters:

[in] **pid** player tag which get from player_start return value

Returns:

PLAYER_NOT_VALID_PID playet tag invalid
PLAYER_NOMEM alloc memory failed
PLAYER_SUCCESS success

if need_start set 0, no need call player_start_play

Definition at line [173](#) of file **player_ctrl.c**.

char* player_status2str (player_status **status)**

convert player state value to string

player_status2str

Parameters:

[in] **status** player status

Returns:

player status details strings

Definition at line [1529](#) of file **player_ctrl.c**.

Referenced by **player_value2str()**.

int player_stop (int **pid)**

send stop command to player (synchronous)

player_stop

Parameters:

[in] **pid** player tag which get from player_start return value

Returns:

PLAYER_NOT_VALID_PID playet tag invalid
PLAYER_NOMEM alloc memory failed
PLAYER_SUCCESS success

if player already stop, return directly wait thread exit after send stop command

Definition at line [216](#) of file [player_ctrl.c](#).

Referenced by [player_exit\(\)](#).

`int player_stop_async (int pid)`

send stop command to player (asynchronous)

player_stop_async

Parameters:

[in] **pid** player tag which get from player_start return value

Returns:

PLAYER_NOT_VALID_PID playet tag invalid
PLAYER_NOMEM alloc memory failed
PLAYER_SUCCESS success

if player already stop, return directly needn't wait thread exit

Definition at line [277](#) of file [player_ctrl.c](#).

`int player_timeSearch (int pid, float s_time)`

seek to designated time point to play.

player_timeSearch

Parameters:

[in] **pid** player tag which get from player_start return value

[in] **s_time** target time, unit is second

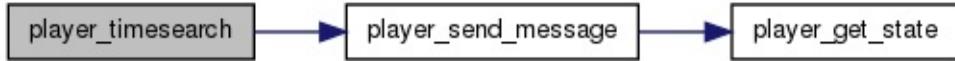
Returns:

PLAYER_NOT_VALID_PID playet tag invalid
PLAYER_NOMEM alloc memory failed
PLAYER_SUCCESS success

After time search, player playback from a key frame

Definition at line [502](#) of file **player_ctrl.c**.

Here is the call graph for this function:



```
char* player_value2str( char * key,  
                        int     value  
                      )
```

convert player state value to string

player_value2str

Parameters:

[in] **char** *key valuetype key: status player status; vformat video format; aformat aduio format

[in] **int** value which need convert to string

Returns:

player status details strings

Definition at line [1734](#) of file **player_ctrl.c**.

Here is the call graph for this function:



int player_video_overlay_en (unsigned enable)

enable osd colorkey

player_video_overlay_en

Parameters:

[in] **enable** osd colorkey enable flag

Returns:

PLAYER_SUCCESS success PLAYER_FAILED failed

Definition at line **1034** of file **player_ctrl.c**.

amplayerMy Project

Main Page	Classes	Files
File List	File Members	
All	Functions	
a	p	

Here is a list of all documented file members with links to the documentation:

- a -

- `audio_get_lrvolume()` : `player_ctrl.c`
- `audio_get_volume()` : `player_ctrl.c`
- `audio_get_volume_range()` : `player_ctrl.c`
- `audio_left_mono()` : `player_ctrl.c`
- `audio_lr_mix_set()` : `player_ctrl.c`
- `audio_right_mono()` : `player_ctrl.c`
- `audio_set_lrvolume()` : `player_ctrl.c`
- `audio_set_mute()` : `player_ctrl.c`
- `audio_set_spectrum_switch()` : `player_ctrl.c`
- `audio_set_volume()` : `player_ctrl.c`
- `audio_set_volume_balance()` : `player_ctrl.c`
- `audio_stereo()` : `player_ctrl.c`
- `audio_swap_left_right()` : `player_ctrl.c`

- p -

- `player_aid()` : `player_ctrl.c`
- `player_backward()` : `player_ctrl.c`
- `player_cache_system_init()` : `player_ctrl.c`
- `player_enable_autobuffer()` : `player_ctrl.c`
- `player_exit()` : `player_ctrl.c`
- `player_forward()` : `player_ctrl.c`
- `player_get_extern_priv()` : `player_ctrl.c`
- `player_get_lpbufbuffedsize()` : `player_ctrl.c`

- player_get_media_info() : [player_ctrl.c](#)
- player_get_play_info() : [player_ctrl.c](#)
- player_get_state() : [player_ctrl.c](#)
- player_get_streambufbuffedsize() : [player_ctrl.c](#)
- player_init() : [player_ctrl.c](#)
- player_list_allpid() : [player_ctrl.c](#)
- player_loop() : [player_ctrl.c](#)
- player_noloop() : [player_ctrl.c](#)
- player_pause() : [player_ctrl.c](#)
- player_progress_exit() : [player_ctrl.c](#)
- player_register_update_callback() : [player_ctrl.c](#)
- player_resume() : [player_ctrl.c](#)
- player_send_message() : [player_ctrl.c](#)
- player_set_autobuffer_level() : [player_ctrl.c](#)
- player_sid() : [player_ctrl.c](#)
- player_start() : [player_ctrl.c](#)
- player_start_play() : [player_ctrl.c](#)
- player_status2str() : [player_ctrl.c](#)
- player_stop() : [player_ctrl.c](#)
- player_stop_async() : [player_ctrl.c](#)
- player_timesearch() : [player_ctrl.c](#)
- player_value2str() : [player_ctrl.c](#)
- player_video_overlay_en() : [player_ctrl.c](#)

Generated on Tue Dec 2 2014 21:55:21 for amplayerMy Project by
[doxygen](#) 1.7.6.1

amplayer My Project

Main Page	Classes	Files
File List	File Members	
All	Functions	
a	p	

- a -

- audio_get_lrvolume() : [player_ctrl.c](#)
- audio_get_volume() : [player_ctrl.c](#)
- audio_get_volume_range() : [player_ctrl.c](#)
- audio_left_mono() : [player_ctrl.c](#)
- audio_lr_mix_set() : [player_ctrl.c](#)
- audio_right_mono() : [player_ctrl.c](#)
- audio_set_lrvolume() : [player_ctrl.c](#)
- audio_set_mute() : [player_ctrl.c](#)
- audio_set_spectrum_switch() : [player_ctrl.c](#)
- audio_set_volume() : [player_ctrl.c](#)
- audio_set_volume_balance() : [player_ctrl.c](#)
- audio_stereo() : [player_ctrl.c](#)
- audio_swap_left_right() : [player_ctrl.c](#)

- p -

- player_aid() : [player_ctrl.c](#)
- player_backward() : [player_ctrl.c](#)
- player_cache_system_init() : [player_ctrl.c](#)
- player_enable_autobuffer() : [player_ctrl.c](#)
- player_exit() : [player_ctrl.c](#)
- player_forward() : [player_ctrl.c](#)
- player_get_extern_priv() : [player_ctrl.c](#)
- player_get_lpbuffbuffedsize() : [player_ctrl.c](#)
- player_get_media_info() : [player_ctrl.c](#)

- player_get_play_info() : `player_ctrl.c`
- player_get_state() : `player_ctrl.c`
- player_get_streambufbuffedsize() : `player_ctrl.c`
- player_init() : `player_ctrl.c`
- player_list_allpid() : `player_ctrl.c`
- player_loop() : `player_ctrl.c`
- player_noloop() : `player_ctrl.c`
- player_pause() : `player_ctrl.c`
- player_progress_exit() : `player_ctrl.c`
- player_register_update_callback() : `player_ctrl.c`
- player_resume() : `player_ctrl.c`
- player_send_message() : `player_ctrl.c`
- player_set_autobuffer_level() : `player_ctrl.c`
- player_sid() : `player_ctrl.c`
- player_start() : `player_ctrl.c`
- player_start_play() : `player_ctrl.c`
- player_status2str() : `player_ctrl.c`
- player_stop() : `player_ctrl.c`
- player_stop_async() : `player_ctrl.c`
- player_timesearch() : `player_ctrl.c`
- player_value2str() : `player_ctrl.c`
- player_video_overlay_en() : `player_ctrl.c`

Generated on Tue Dec 2 2014 21:55:22 for amplayerMy Project by
[doxygen](#) 1.7.6.1

amplayerMy Project

Main Page	Classes	Files
Class List	Class Index	

audio_tag_info Member List

This is the complete list of members for [audio_tag_info](#), including all inherited members.

album (defined in audio_tag_info)	audio_tag_info
author (defined in audio_tag_info)	audio_tag_info
comment (defined in audio_tag_info)	audio_tag_info
copyright (defined in audio_tag_info)	audio_tag_info
genre (defined in audio_tag_info)	audio_tag_info
pic (defined in audio_tag_info)	audio_tag_info
title (defined in audio_tag_info)	audio_tag_info
track (defined in audio_tag_info)	audio_tag_info
year (defined in audio_tag_info)	audio_tag_info

Generated on Tue Dec 2 2014 21:55:18 for amplayerMy Project by
[doxygen](#) 1.7.6.1

amplayerMy Project

[Main Page](#)[Classes](#)[Files](#)[File List](#)[File Members](#)

player_type.h

```
00001 #ifndef _PLAYER_TYPE_H_
00002 #define _PLAYER_TYPE_H_
00003
00004 #include <libavformat/avformat.h>
00005 #include <stream_format.h>
00006
00007 #define MSG_SIZE 64
00008 #define MAX_VIDEO_STREAMS 10
00009 #define MAX_AUDIO_STREAMS 16
00010 #define MAX_SUB_INTERNAL 64
00011 #define MAX_SUB_EXTERNAL 24
00012 #define MAX_SUB_STREAMS (MAX_SUB
_INTERNAL + MAX_SUB_EXTERNAL)
00013 #define MAX_PLAYER_THREADS 32
00014
00015 #define CALLBACK_INTERVAL
(300)
00016
00017 //#define DEBUG_VARIABLE_DUR
00018
00019 typedef enum
00020 {
00021     ****
00022     * 0x1000x:
00023     * player do parse file
00024     * decoder not running
00025     ****
00026     PLAYER_INITING = 0x10001,
```

```
00027     PLAYER_TYPE_READY = 0x10002,
00028     PLAYER_INITOK      = 0x10003,
00029
00030     /*****
00031     * 0x2000x:
00032     * playback status
00033     * decoder is running
00034     *****/
00035     PLAYER_RUNNING        = 0x20001,
00036     PLAYER_BUFFERING       = 0x20002,
00037     PLAYER_PAUSE           = 0x20003,
00038     PLAYER_SEARCHING       = 0x20004,
00039
00040     PLAYER_SEARCHOK         = 0x20005,
00041     PLAYER_START            = 0x20006,
00042
00043     PLAYER_FF_END           = 0x20007,
00044     PLAYER_FB_END           = 0x20008,
00045
00046     PLAYER_PLAY_NEXT         = 0x20009,
00047
00048     PLAYER_BUFFER_OK          = 0x2000a,
00049
00050     PLAYER_FOUND_SUB          = 0x2000b,
00051
00052     /*****
00053     * 0x3000x:
00054     * player will exit
00055     *****/
00056     PLAYER_ERROR             = 0x30001,
00057     PLAYER_PLAYEND            = 0x30002,
00058
00059     PLAYER_STOPED             = 0x30003,
00060     PLAYER_EXIT                = 0x30004,
```

```
00058     ****
00059     * 0x4000x:
00060     * divx drm
00061     * decoder will exit or give
00062     * a message dialog
00063     * ****
00064     PLAYER_DIVX_AUTHERR      = 0x40001,
00065     PLAYER_DIVX_RENTAL_EXPIRED = 0x40002,
00066     PLAYER_DIVX_RENTAL_VIEW  = 0x40003,
00067 }player_status;
00068
00069
00070 typedef enum {
00071     DRM_LEVEL1      = 1,
00072     DRM_LEVEL2      = 2,
00073     DRM_LEVEL3      = 3,
00074     DRM_NONE        = 4,
00075 } drm_level_t;
00076
00077 typedef struct drminfo {
00078     drm_level_t drm_level;
00079     int drm_flag;
00080     int drm_hasesdata;
00081     int drm_priv;
00082     unsigned int drm_pktsize;
00083     unsigned int drm_pktpts;
00084     unsigned int drm_phy;
00085     unsigned int drm_vir;
00086     unsigned int drm_remap;
00087     int data_offset;
00088     int extpad[8];
00089 } drminfo_t;
00090
00091
00092
00093 typedef struct
00094 {
```

```
00095         int index;
00096         int id;
00097         int width;
00098         int height;
00099         int aspect_ratio_num;
00100         int aspect_ratio_den;
00101         int frame_rate_num;
00102         int frame_rate_den;
00103             int bit_rate;
00104         vformat_t format;
00105         int duartion;
00106         unsigned int video_rotation_degree;
00107 }mvideo_info_t;
00108
00109 typedef enum
00110 {
00111     ACOVER_NONE    = 0,
00112     ACOVER_JPG     ,
00113     ACOVER_PNG     ,
00114 }audio_cover_type;
00115
00116 typedef struct
00117 {
00118     char title[512];
00119     char author[512];
00120     char album[512];
00121     char comment[512];
00122     char year[4];
00123     int track;
00124     char genre[32];
00125     char copyright[512];
00126     audio_cover_type pic;
00127 }audio_tag_info;
00128
00129 typedef struct
00130 {
00131     int index;
```

```
00132     int id;
00133     int channel;
00134     int sample_rate;
00135     int bit_rate;
00136     aformat_t aformat;
00137     int duration;
00138         audio_tag_info *audio_tag;
00139 }maudio_info_t;
00140
00141 typedef struct
00142 {
00143     int index;
00144     char id;
00145     char internal_external; //0:internal_sub
00146     1:external_sub
00147     unsigned short width;
00148     unsigned short height;
00149     unsigned int sub_type;
00150     char resolution;
00151     long long subtitle_size;
00152     char *sub_language;
00153 }msub_info_t;
00154
00155 typedef struct
00156 {
00157     char *filename;
00158     int duration;
00159     long long file_size;
00160     pfile_type type;
00161     int bitrate;
00162     int has_video;
00163     int has_audio;
00164     int has_sub;
00165     int nb_streams;
00166     int total_video_num;
00167     int cur_video_index;
00168     int total_audio_num;
```

```
00168     int cur_audio_index;
00169     int total_sub_num;
00170     int cur_sub_index;
00171     int seekable;
00172     int drm_check;
00173         int adif_file_flag;
00174 }mstream_info_t;
00175
00176 typedef struct
00177 {
00178     mstream_info_t stream_info;
00179     mvideo_info_t *video_info[MAX_VIDEO_
STREAMS];
00180     maudio_info_t *audio_info[MAX_AUDIO_
STREAMS];
00181     msub_info_t *sub_info[MAX_SUB_STREAMS];
00182 }media_info_t;
00183
00184 typedef struct player_info
00185 {
00186     char *name;
00187     player_status last_sta;
00188     player_status status;           /
*stop,pause */
00189     int full_time;      /*Seconds      */
00190     int full_time_ms;   /* mSeconds */
00191     int current_time;   /*Seconds      */
00192     int current_ms;    /*ms*/
00193     int last_time;
00194     int error_no;
00195     int64_t start_time;
00196     int64_t first_time;
00197     int pts_video;
00198     //int pts_pcrscr;
00199     unsigned int current_pts;
00200     long curtime_old_time;
00201     unsigned int video_error_cnt;
```

```
00202         unsigned int audio_error_cnt;
00203         float audio_bufferlevel; // relative
00204             value
00205             float video_bufferlevel; // relative
00206                 value
00207             int64_t bufed_pos;
00208             int     bufed_time; /* Second*/
00209             unsigned int drm_rental;
00210             int64_t download_speed; //download s
00211             peed
00212 }player_info_t;
00213
00214 typedef struct pid_info
00215 {
00216     int num;
00217     int pid[MAX_PLAYER_THREADS];
00218 }pid_info_t;
00219
00220 typedef struct player_file_type
00221 {
00222     const char *fmt_string;
00223     int video_tracks;
00224     int audio_tracks;
00225     int subtitle_tracks;
00226
00227 }player_file_type_t;
00228
00229
00230 #define STATE_PRE(sta) (sta>>16)
00231 #define PLAYER_THREAD_IS_INITING(sta)    (STA
00232 TE_PRE(sta)==0x1)
00233 #define PLAYER_THREAD_IS_RUNNING(sta)   (STA
00234 TE_PRE(sta)==0x2)
00235 #define PLAYER_THREAD_IS_STOPPED(sta)  (sta
```

```
==PLAYER_EXIT)
00234
00235 typedef int (*update_state_fun_t)(int pid,player_info_t *);
00236 typedef int (*notify_callback)(int pid,int msg,unsigned long ext1,unsigned long ext2);
00237 typedef enum
00238 {
00239     PLAYER_EVENTS_PLAYER_INFO=1,
00240     PLAYER_EVENTS_STATE_CHANGED,
00241     PLAYER_EVENTS_ERROR,
00242     PLAYER_EVENTS_BUFFERING,
00243     PLAYER_EVENTS_FILE_TYPE,
00244     PLAYER_EVENTS_HTTP_WV,
00245     PLAYER_EVENTS_HWBUF_DATA_SIZE_CHANGE
D,
00246     PLAYER_EVENTS_NOT_SUPPORT_SEEKABLE,
    //not support seek;
00247     PLAYER_EVENTS_VIDEO_SIZE_CHANGED,
00248     PLAYER_EVENTS_SUBTITLE_DATA,
    // sub data ext1 refers to subtitledata struct
00249 }player_events;
00250
00251 typedef struct
00252 {
00253     int vbufused;
00254     int vbufsize;
00255     int vdatasize;
00256     int abufused;
00257     int abufsize;
```

```
00258     int adatasize;
00259     int sbufused;
00260     int sbufsize;
00261     int sdatasize;
00262 }hwbufstats_t;
00263
00264
00265 typedef struct
00266 {
00267     update_state_fun_t update_statue_callback;
00268     int update_interval;
00269     long callback_old_time;
00270     notify_callback    notify_fn;
00271 }callback_t;
00272
00273 typedef struct
00274 {
00275     char *file_name;
00276             //file url
00277     char *headers;
00278             //file name's authentication information, maybe used in network streaming
00279             //List *play_list;
00280     int video_index;
00281             //video track, no assigned, please set to -1
00282     int audio_index;
00283             //audio track, no assigned, please set to -1
00284     int sub_index;
00285             //subtitle track, no assigned, please set to -1
00286     float t_pos;
00287             //start position, use second as unit
00288     int read_max_cnt;
```

```
//read retry maximum counts, if exceed it, return error
00283     int avsync_threshold;
//for adec av sync threshold in ms
00284     union
00285     {
00286         struct{
00287             unsigned int loop_mode;
//file loop mode 0:loop 1:not
00288             unsigned int nosound;
//0:play with audio 1:play without audio
00289             unsigned int novideo;
//0:play with video 1:play without video
00290             unsigned int hassubtitle;
//0:ignore subtitle 1:extract subtitle if have
00291             unsigned int need_start;
art:1; /*If set need_start, we need call player_start_play to playback*/
00292             #ifdef DEBUG_VARIABLE_DUR
00293             unsigned int is_variable;
//0:extrack duration from header 1:update duration during playback
00294             #endif
00295             unsigned int display_frame : 1;
//0:black out when player exit 1:keep last frame when player exit
00296         };
00297             int mode;
//no use
00298     };
00299     callback_t callback_fn;
//callback function
```

```
00300         callback_t subdata_fn;
    // subtitle data notify function
00301         void *subhd;
    // sub handle
00302         int subdatasource;
    // sub data source
00303         int byteiobufsize;
                                //byteio buffer size u
sed in ffmpeg
00304         int loopbufsize;
                                //loop buffer size use
d in ffmpeg
00305         int enable_rw_on_pause;
                                //no use
00306         /*
00307             data%<min && data% <max  enter buffe
ring;
00308             data% >middle exit buffering;
00309         */
00310         int auto_buffering_enable;
                                //auto buffering switch
00311         float buffering_min;
                                //auto buffering low
limit
00312         float buffering_middle;
                                //auto buffering middle limit
00313         float buffering_max;
                                //auto buffering high
limit
00314         int is_playlist;
                                //no use
00315         int is_type_parser;
                                //is try to get file
type
00316         int is_livemode;
// support timeshift for chinamobile
00317         int buffering_starttime_s;
```

```
        //for rest buffering_middle,buffering se
conds data to start.
00318        int buffering_force_delay_s;
00319        int lowbuffermode_flag;
00320        int lowbuffermode_limited_ms;
00321        int is_ts_soft_demux;
00322        int reserved [56];
                    //reserved for furthur used,s
ome one add more ,can del reserved num
00323        int SessionID;
00324        int t_duration_ms;           //du
ration parsed from url
00325 }play_control_t;
00326
00327 #endif
```

Generated on Tue Dec 2 2014 21:55:13 for amplayerMy Project by
[doxygen](#) 1.7.6.1

My Project

Main Page

Classes

Files

Graph Legend

This page explains how to interpret the graphs that are generated by doxygen.

Consider the following example:

```
/*! Invisible class because of truncation */
class Invisible { };

/*! Truncated class, inheritance relation is hidden */
class Truncated : public Invisible { };

/* Class not documented with doxygen comments */
class Undocumented { };

/*! Class that is inherited using public inheritance */
class PublicBase : public Truncated { };

/*! A template class */
template<class T> class Templ { };

/*! Class that is inherited using protected inheritance */
class ProtectedBase { };

/*! Class that is inherited using private inheritance */
class PrivateBase { };
```

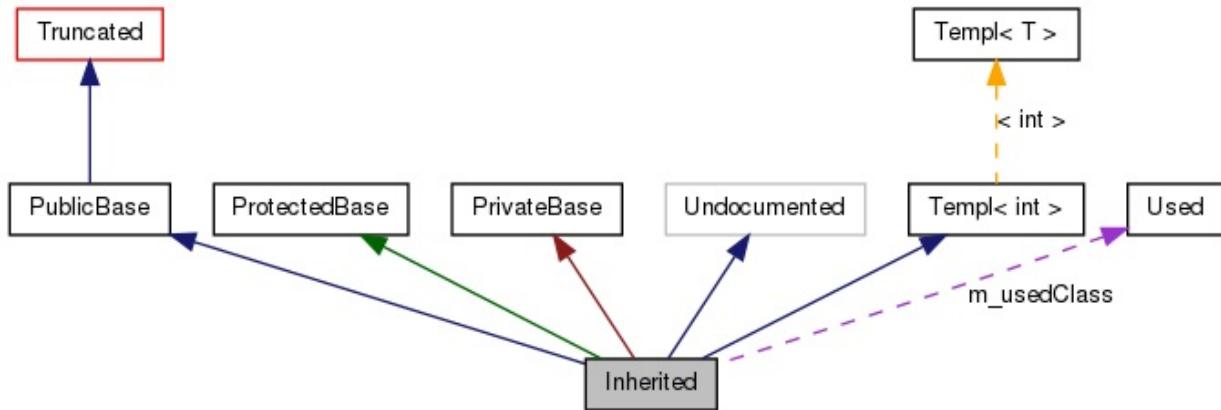
```

/*! Class that is used by the Inherited class */
class Used { };

/*! Super class that inherits a number of other classes */
class Inherited : public PublicBase,
                   protected ProtectedBase,
                   private PrivateBase,
                   public Undocumented,
                   public Templ<int>
{
    private:
        Used *m_usedClass;
};

```

This will result in the following graph:



The boxes in the above graph have the following meaning:

- A filled gray box represents the struct or class for which the graph is generated.
- A box with a black border denotes a documented struct or class.
- A box with a grey border denotes an undocumented struct or class.
- A box with a red border denotes a documented struct or class for which not all inheritance/containment relations are shown. A graph is truncated if it does not fit within the specified boundaries.

The arrows have the following meaning:

- A dark blue arrow is used to visualize a public inheritance relation between two classes.
- A dark green arrow is used for protected inheritance.
- A dark red arrow is used for private inheritance.
- A purple dashed arrow is used if a class is contained or used by another class. The arrow is labeled with the variable(s) through which the pointed class or struct is accessible.
- A yellow dashed arrow denotes a relation between a template instance and the template class it was instantiated from. The arrow is labeled with the template parameters of the instance.

Generated on Tue Dec 2 2014 21:55:21 for amplayerMy Project by
[doxygen](#) 1.7.6.1

amplayerMy Project

[Main Page](#)[Classes](#)[Files](#)[Class List](#)[Class Index](#)

callback_t Member List

This is the complete list of members for [callback_t](#), including all inherited members.

callback_old_time (defined in callback_t)	callback_t
notify_fn (defined in callback_t)	callback_t
update_interval (defined in callback_t)	callback_t
update_statue_callback (defined in callback_t)	callback_t

Generated on Tue Dec 2 2014 21:55:19 for amplayerMy Project by
[doxygen](#) 1.7.6.1

amplayerMy Project

Main Page	Classes	Files
Class List	Class Index	

drm_info Member List

This is the complete list of members for [drm_info](#), including all inherited members.

data_offset (defined in drm_info)	drm_info
drm_flag (defined in drm_info)	drm_info
drm_hasesdata (defined in drm_info)	drm_info
drm_level (defined in drm_info)	drm_info
drm_phy (defined in drm_info)	drm_info
drm_pktpts (defined in drm_info)	drm_info
drm_pktsize (defined in drm_info)	drm_info
drm_priv (defined in drm_info)	drm_info
drm_remap (defined in drm_info)	drm_info
drm_vir (defined in drm_info)	drm_info
extpad (defined in drm_info)	drm_info

Generated on Tue Dec 2 2014 21:55:19 for amplayerMy Project by
[doxygen](#) 1.7.6.1

amplayerMy Project

Main Page	Classes	Files
Class List	Class Index	

hwbufstats_t Member List

This is the complete list of members for `hwbufstats_t`, including all inherited members.

abufsize (defined in <code>hwbufstats_t</code>)	hwbufstats_t
abufused (defined in <code>hwbufstats_t</code>)	hwbufstats_t
adatasize (defined in <code>hwbufstats_t</code>)	hwbufstats_t
sbufsize (defined in <code>hwbufstats_t</code>)	hwbufstats_t
sbufused (defined in <code>hwbufstats_t</code>)	hwbufstats_t
sdatasize (defined in <code>hwbufstats_t</code>)	hwbufstats_t
vbufsize (defined in <code>hwbufstats_t</code>)	hwbufstats_t
vbufused (defined in <code>hwbufstats_t</code>)	hwbufstats_t
vdatasize (defined in <code>hwbufstats_t</code>)	hwbufstats_t

Generated on Tue Dec 2 2014 21:55:19 for amplayerMy Project by
[doxygen](#) 1.7.6.1

amplayerMy Project

Main Page	Classes	Files
Class List	Class Index	

maudio_info_t Member List

This is the complete list of members for [maudio_info_t](#), including all inherited members.

aformat (defined in maudio_info_t)	maudio_info_t
audio_tag (defined in maudio_info_t)	maudio_info_t
bit_rate (defined in maudio_info_t)	maudio_info_t
channel (defined in maudio_info_t)	maudio_info_t
duration (defined in maudio_info_t)	maudio_info_t
id (defined in maudio_info_t)	maudio_info_t
index (defined in maudio_info_t)	maudio_info_t
sample_rate (defined in maudio_info_t)	maudio_info_t

Generated on Tue Dec 2 2014 21:55:20 for amplayerMy Project by
[doxygen](#) 1.7.6.1

amplayerMy Project

Main Page	Classes	Files
Class List	Class Index	

[media_info_t](#) Member List

This is the complete list of members for **[media_info_t](#)**, including all inherited members.

[audio_info](#) (defined in **[media_info_t](#)**) **[media_info_t](#)**

[stream_info](#) (defined in **[media_info_t](#)**) **[media_info_t](#)**

[sub_info](#) (defined in **[media_info_t](#)**) **[media_info_t](#)**

[video_info](#) (defined in **[media_info_t](#)**) **[media_info_t](#)**

Generated on Tue Dec 2 2014 21:55:20 for amplayerMy Project by

[doxygen](#) 1.7.6.1

amplayerMy Project

Main Page	Classes	Files
Class List	Class Index	

mstream_info_t Member List

This is the complete list of members for [mstream_info_t](#), including all inherited members.

adif_file_flag (defined in mstream_info_t)	mstream_info_t
bitrate (defined in mstream_info_t)	mstream_info_t
cur_audio_index (defined in mstream_info_t)	mstream_info_t
cur_sub_index (defined in mstream_info_t)	mstream_info_t
cur_video_index (defined in mstream_info_t)	mstream_info_t
drm_check (defined in mstream_info_t)	mstream_info_t
duration (defined in mstream_info_t)	mstream_info_t
file_size (defined in mstream_info_t)	mstream_info_t
filename (defined in mstream_info_t)	mstream_info_t
has_audio (defined in mstream_info_t)	mstream_info_t
has_sub (defined in mstream_info_t)	mstream_info_t
has_video (defined in mstream_info_t)	mstream_info_t
nb_streams (defined in mstream_info_t)	mstream_info_t
seekable (defined in mstream_info_t)	mstream_info_t
total_audio_num (defined in mstream_info_t)	mstream_info_t
total_sub_num (defined in mstream_info_t)	mstream_info_t
total_video_num (defined in mstream_info_t)	mstream_info_t
type (defined in mstream_info_t)	mstream_info_t

amplayerMy Project

Main Page	Classes	Files
Class List	Class Index	

msub_info_t Member List

This is the complete list of members for **msub_info_t**, including all inherited members.

height (defined in msub_info_t)	msub_info_t
id (defined in msub_info_t)	msub_info_t
index (defined in msub_info_t)	msub_info_t
internal_external (defined in msub_info_t)	msub_info_t
resolution (defined in msub_info_t)	msub_info_t
sub_language (defined in msub_info_t)	msub_info_t
sub_type (defined in msub_info_t)	msub_info_t
subtitle_size (defined in msub_info_t)	msub_info_t
width (defined in msub_info_t)	msub_info_t

Generated on Tue Dec 2 2014 21:55:20 for amplayerMy Project by
[doxygen](#) 1.7.6.1

amplayerMy Project

Main Page	Classes	Files
Class List	Class Index	

mvideo_info_t Member List

This is the complete list of members for [mvideo_info_t](#), including all inherited members.

aspect_ratio_den (defined in mvideo_info_t)	mvideo_info_t
aspect_ratio_num (defined in mvideo_info_t)	mvideo_info_t
bit_rate (defined in mvideo_info_t)	mvideo_info_t
duartion (defined in mvideo_info_t)	mvideo_info_t
format (defined in mvideo_info_t)	mvideo_info_t
frame_rate_den (defined in mvideo_info_t)	mvideo_info_t
frame_rate_num (defined in mvideo_info_t)	mvideo_info_t
height (defined in mvideo_info_t)	mvideo_info_t
id (defined in mvideo_info_t)	mvideo_info_t
index (defined in mvideo_info_t)	mvideo_info_t
video_rotation_degree (defined in mvideo_info_t)	mvideo_info_t
width (defined in mvideo_info_t)	mvideo_info_t

Generated on Tue Dec 2 2014 21:55:20 for amplayerMy Project by

[doxygen](#) 1.7.6.1

amplayerMy Project

Main Page	Classes	Files
Class List	Class Index	

pid_info Member List

This is the complete list of members for [pid_info](#), including all inherited members.

[num](#) (defined in [pid_info](#)) [pid_info](#)

[pid](#) (defined in [pid_info](#)) [pid_info](#)

Generated on Tue Dec 2 2014 21:55:20 for amplayerMy Project by

[doxygen](#) 1.7.6.1

amplayerMy Project

Main Page	Classes	Files
Class List	Class Index	

play_control_t Member List

This is the complete list of members for [play_control_t](#), including all inherited members.

audio_index (defined in play_control_t)	play_control_t
auto_buffering_enable (defined in play_control_t)	play_control_t
avsync_threshold (defined in play_control_t)	play_control_t
buffering_force_delay_s (defined in play_control_t)	play_control_t
buffering_max (defined in play_control_t)	play_control_t
buffering_middle (defined in play_control_t)	play_control_t
buffering_min (defined in play_control_t)	play_control_t
buffering_starttime_s (defined in play_control_t)	play_control_t
byteiobufsize (defined in play_control_t)	play_control_t
callback_fn (defined in play_control_t)	play_control_t
displast_frame (defined in play_control_t)	play_control_t
enable_rw_on_pause (defined in play_control_t)	play_control_t
file_name (defined in play_control_t)	play_control_t
hassub (defined in play_control_t)	play_control_t
headers (defined in play_control_t)	play_control_t
is_livemode (defined in play_control_t)	play_control_t
is_playlist (defined in play_control_t)	play_control_t
is_ts_soft_demux (defined in play_control_t)	play_control_t
is_type_parser (defined in play_control_t)	play_control_t
loop_mode (defined in play_control_t)	play_control_t
loopbufsize (defined in play_control_t)	play_control_t
lowbuffermode_flag (defined in play_control_t)	play_control_t
lowbuffermode_limited_ms (defined in	

play_control_t	play_control_t
mode (defined in play_control_t)	play_control_t
need_start (defined in play_control_t)	play_control_t
nosound (defined in play_control_t)	play_control_t
novideo (defined in play_control_t)	play_control_t
read_max_cnt (defined in play_control_t)	play_control_t
reserved (defined in play_control_t)	play_control_t
SessionID (defined in play_control_t)	play_control_t
sub_index (defined in play_control_t)	play_control_t
subdata_fn (defined in play_control_t)	play_control_t
subdatasource (defined in play_control_t)	play_control_t
subhd (defined in play_control_t)	play_control_t
t_duration_ms (defined in play_control_t)	play_control_t
t_pos (defined in play_control_t)	play_control_t
video_index (defined in play_control_t)	play_control_t

Generated on Tue Dec 2 2014 21:55:20 for amplayerMy Project by
[doxygen](#) 1.7.6.1

amplayerMy Project

Main Page	Classes	Files
Class List	Class Index	

player_file_type Member List

This is the complete list of members for [player_file_type](#), including all inherited members.

audio_tracks (defined in player_file_type)	player_file_type
fmt_string (defined in player_file_type)	player_file_type
subtitle_tracks (defined in player_file_type)	player_file_type
video_tracks (defined in player_file_type)	player_file_type

Generated on Tue Dec 2 2014 21:55:20 for amplayerMy Project by
[doxygen](#) 1.7.6.1

amplayerMy Project

Main Page	Classes	Files
Class List	Class Index	

player_info Member List

This is the complete list of members for [player_info](#), including all inherited members.

audio_bufferlevel (defined in player_info)	player_info
audio_error_cnt (defined in player_info)	player_info
bufed_pos (defined in player_info)	player_info
bufed_time (defined in player_info)	player_info
current_ms (defined in player_info)	player_info
current_pts (defined in player_info)	player_info
current_time (defined in player_info)	player_info
curtime_old_time (defined in player_info)	player_info
download_speed (defined in player_info)	player_info
drm_rental (defined in player_info)	player_info
error_no (defined in player_info)	player_info
first_time (defined in player_info)	player_info
full_time (defined in player_info)	player_info
full_time_ms (defined in player_info)	player_info
last_pts (defined in player_info)	player_info
last_sta (defined in player_info)	player_info
last_time (defined in player_info)	player_info
name (defined in player_info)	player_info
pts_video (defined in player_info)	player_info
seek_delay (defined in player_info)	player_info
seek_point (defined in player_info)	player_info
start_time (defined in player_info)	player_info
status (defined in player_info)	player_info

video_bufferlevel (defined in [player_info](#)) [player_info](#)
video_error_cnt (defined in [player_info](#)) [player_info](#)

Generated on Tue Dec 2 2014 21:55:21 for amplayerMy Project by
[doxygen](#) 1.7.6.1

amplayerMy Project

Main Page

Classes

Files

File List

File Members

player.h

```
00001 #ifndef _PLAYER_H_
00002 #define _PLAYER_H_
00003
00004
00005 #include <codec.h>
00006 #include <player_type.h>
00007 #include <player_error.h>
00008 #include <message.h>
00009 #include <player_dump.h>
00010
00011 #ifdef __cplusplus
00012 extern "C" {
00013 #endif
00014
00015 int      player_init();
00016 int      player_start(play_control_t *p, unsig
ned long  priv);
00017 int      player_stop(int pid);
00018 int      player_stop_async(int pid);
00019 int      player_exit(int pid);
00020 int      player_pause(int pid);
00021 int          player_resume(int pid);
00022 int      player_timeSearch(int pid, float s_ti
me);
00023 int      player_forward(int pid, int speed);
00024 int      player_backward(int pid, int speed);
00025 int      player_aid(int pid, int audio_id);
00026 int      player_sid(int pid, int sub_id);
```

```
00027 int      player_progress_exit(void);
00028 int      player_list_allpid(pid_info_t *pid);
00029 int      check_pid_valid(int pid);
00030 int      player_get_play_info(int pid,player_
info_t *info);
00031 int      player_get_media_info(int pid,media_
info_t *minfo);
00032 int      player_video_overlay_en(unsigned ena
ble);
00033 int      player_start_play(int pid);
00034 int      player_send_message(int pid, player_
cmd_t *cmd);
00035 player_status  player_get_state(int pid);
00036 unsigned int    player_get_extern_priv(int p
id);
00037 int      player_enable_autobuffer(int pid, int
enable);
00038 int      player_set_autobuffer_level(int pid,
float min, float middle, float max);
00039
00040 int      audio_set_mute(int pid,int mute);
00041 int      audio_get_volume_range(int pid,float
*min,float *max);
00042 int      audio_set_volume(int pid,float val);
00043 int      audio_get_volume(int pid, float *val
);
00044
00045 int      audio_set_lrvolume(int pid,float lvo
l,float rvol);
00046 int      audio_get_lrvolume(int pid, float* l
vol,float* rvol);
00047
00048 int      audio_set_volume_balance(int pid,int
balance);
00049 int      audio_swap_left_right(int pid);
00050 int      audio_left_mono(int pid);
00051 int      audio_right_mono(int pid);
```

```
00052 int      audio_stereo(int pid);
00053 int      audio_lr_mix_set(int pid,int enable)
;
00054 int      audio_cur_pcmpara_Applied_get(int pi
d,int *pfs,int *pch);
00055
00056 int      audio_set_spectrum_switch(int pid,int
isStart,int interval);
00057 int      player_register_update_callback(call
back_t *cb,update_state_fun_t up_fn,int interval_s
);
00058 char *player_status2str(player_status status
);
00059 char *player_value2str(char *key, int value)
;
00060 int      player_cache_system_init(int enable,
const char*dir,int max_size,int block_size);
00061
00062 //control interface
00063 int      player_loop(int pid);
00064 int      player_noloop(int pid);
00065
00066 int      check_url_type(char *filename);
00067 int      play_list_player(play_control_t *pct
rl,unsigned long priv);
00068
00069 //freescale
00070 int      enable_freescale(int cfg);
00071 int      disable_freescale(int cfg);
00072 int      disable_freescale_MBX();
00073 int      enable_2Xscale();
00074 int      enable_2XYscale();
00075 int      enable_freescale_MBX();
00076 int      disable_2X_2XYscale();
00077 int      GL_2X_scale(int mSwitch);
00078 int      wait_play_end();
00079 int      wait_video_unreg();
```

```
00080 int    clear_video_buf();
00081 int    freescale_is_enable();
00082 int64_t player_get_lpbuffbuffedsize(int pid);
00083 int64_t player_get_streambufbuffedsize(int p
id);
00084 int audio_get_decoder_enable(int pid);
00085
00086 #ifdef __cplusplus
00087 }
00088 #endif
00089
00090 #endif
00091
```

Generated on Tue Dec 2 2014 21:55:12 for amplayerMy Project by
[doxygen](#) 1.7.6.1

amplayerMy Project

Main Page

Classes

Files

File List

File Members

player_ctrl.c

Go to the documentation of this file.

```
00001
00009 /* Copyright (c) 2007-2011, Amlogic Inc.
00010 * All right reserved
00011 *
00012 */
00013
00014 #include <pthread.h>
00015 #include <player.h>
00016 #include <player_set_sys.h>
00017
00018 #include "player_ts.h"
00019 #include "player_es.h"
00020 #include "player_rm.h"
00021 #include "player_ps.h"
00022 #include "player_video.h"
00023 #include "player_audio.h"
00024
00025 #include "player_update.h"
00026 #include "thread_mgt.h"
00027 #include "player_ffmpeg_ctrl.h"
00028 #include "player_cache_mgt.h"
00029 #include "player_priv.h"
00030 #include <amthreadpool.h>
00031
00032 #ifndef FBIOPUT OSD_SRCCOLORKEY
00033 #define FBIOPUT OSD_SRCCOLORKEY 0x46fb
00034 #endif
```

```
00035
00036 #ifndef FBIOPUT OSD_SRCKEY_ENABLE
00037 #define FBIOPUT OSD_SRCKEY_ENABLE 0x46fa
00038 #endif
00039
00040 extern void print_version_info();
00041
00042 /* -----
00043 ----- */
00059 /* -----
00060 ----- */
00061 int player_init(void)
00062 {
00063     print_version_info();
00064     update_loglevel_setting();
00065     /*register all formats and codecs*/
00066     ffmpeg_init();
00067
00068     player_id_pool_init();
00069
00070     codec_audio_basic_init();
00071
00072     /*register all support decoder */
00073     ts_register_stream_decoder();
00074     es_register_stream_decoder();
00075     ps_register_stream_decoder();
00076     rm_register_stream_decoder();
00077     audio_register_stream_decoder();
00078     video_register_stream_decoder();
00079     return PLAYER_SUCCESS;
00080 }
00081
00082 /* -----
00083 ----- */
00097 /* -----
00098 ----- */
```

```
00098 int player_start(play_control_t *ctrl_p, uns  
00099 igned long  priv)  
00100 {  
00101     int ret;  
00102     int pid = -1;  
00103     play_para_t *p_para;  
00104     //char stb_source[32];  
00105     update_loglevel_setting();  
00106     update_dump_dir_path();  
00107     print_version_info();  
00108     log_print("[player_start:enter]p=%p blac  
k=%d\n", ctrl_p, get_black_policy());  
00109  
00110     if (ctrl_p == NULL) {  
00111         return PLAYER_EMPTY_P;  
00112     }  
00113  
00114     /*keep last frame displaying --default*/  
00115     set_black_policy(0);  
00116     /* if not set keep last frame, or change  
file playback, clear display last frame */  
00117     if (!ctrl_p->displast_frame) {  
00118         set_black_policy(1);  
00119     } else if (!check_file_same(ctrl_p->file  
_name)) {  
00120         set_black_policy(1);  
00121     }  
00122  
00123     pid = player_request_pid();  
00124     if (pid < 0) {  
00125         return PLAYER_NOT_VALID_PID;  
00126     }  
00127  
00128     p_para = MALLOC(sizeof(play_para_t));  
00129     if (p_para == NULL) {  
00130         return PLAYER_NOMEM;
```

```
00131     }
00132
00133     MEMSET(p_para, 0, sizeof(play_para_t));
00134
00135     /* init time_point to a invalid value */
00136     p_para->playctrl_info.time_point = -1;
00137
00138     player_init_pid_data(pid, p_para);
00139
00140     message_pool_init(p_para);
00141
00142     p_para->start_param = ctrl_p;
00143     p_para->player_id = pid;
00144     p_para->extern_priv = priv;
00145     log_debug1("[player_start]player_para=%p\n",
00146 , start_param=%p pid=%d\n", p_para, p_para->start_param,
00147 pid);
00148
00149     ret = player_thread_create(p_para) ;
00150     if (ret != PLAYER_SUCCESS) {
00151         FREE(p_para);
00152         player_release_pid(pid);
00153         return PLAYER_CAN_NOT_CREATE_THREADS;
00154     }
00155     log_print("[player_start:exit]pid = %d \n", pid);
00156
00157     return pid;
00158 }
00159
00160 /* -----
00161 -----*/
00162 /* -----
00163 -----*/
00164
00165 int player_start_play(int pid)
00166 {
00167     player_cmd_t *cmd;
```

```
00176     int r = PLAYER_SUCCESS;
00177     play_para_t *player_para;
00178
00179     log_print("[player_start_play:enter]pid=%d\n", pid);
00180
00181     player_para = player_open_pid_data(pid);
00182     if (player_para == NULL) {
00183         return PLAYER_NOT_VALID_PID;
00184     }
00185
00186     cmd = message_alloc();
00187     if (cmd) {
00188         cmd->ctrl_cmd = CMD_START;
00189         r = send_message(player_para, cmd);
00190     } else {
00191         r = PLAYER_NOMEM;
00192     }
00193
00194     player_close_pid_data(pid);
00195     log_print("[player_start_play:exit]pid = %d\n", pid);
00196
00197     return r;
00198 }
00199
00200 /* -----
00215 /* -----
00216 int player_stop(int pid)
00217 {
00218     player_cmd_t *cmd;
00219     int r = PLAYER_SUCCESS;
00220     play_para_t *player_para;
00221     player_status sta;
00222
```

```
00223     log_print("[player_stop:enter]pid=%d\n",
00224     pid);
00225     player_para = player_open_pid_data(pid);
00226     if (player_para == NULL) {
00227         return PLAYER_NOT_VALID_PID;
00228     }
00229
00230     sta = get_player_state(player_para);
00231     log_print("[player_stop]player_status=%x
00232 \n", sta);
00233     if (PLAYER_THREAD_IS_STOPPED(sta)) {
00234         player_close_pid_data(pid);
00235         log_print("[player_stop]pid=%d thread
00236 is already stopped\n", pid);
00237         return PLAYER_SUCCESS;
00238     }
00239 /*if (player_para->pFormatCtx) {
00240     av_ioctl(player_para->pFormatCtx, A
00241 VIOCTL_STOP, 0, 0);
00242 }*/
00243     clear_all_message(player_para);/*clear o
00244 ld message to make sure fast exit.*/
00245     cmd = message_alloc();
00246     if (cmd) {
00247         cmd->ctrl_cmd = CMD_STOP;
00248         ffmpeg_interrupt(player_para->thread
00249 _mgt.pthread_id);
00250         r = send_message(player_para, cmd);
00251         r = player_thread_wait_exit(player_p
00252 ara);
00253         log_print("[player_stop:%d]wait play
00254 er_theadpid[%d] r = %d\n", __LINE__, player_para->
00255 player_id, r);
00256         clear_all_message(player_para);
00257         ffmpeg_uninterrupt(player_para->thre
00258 ad_mgt.pthread_id);
```

```
00250     } else {
00251         r = PLAYER_NOMEM;
00252     }
00253
00254     player_close_pid_data(pid);
00255     log_print("[player_stop:exit]pid=%d\n",
00256     pid);
00256     tcppool_refresh_link_and_check();
00257     log_print("[tcppool_refresh_link_and
00258 _check]pid=%d\n", pid);
00258     return r;
00259 }
00260
00261 /* -----
00262 -----*/
00276 /* -----
00277 -----*/
00277 int player_stop_async(int pid)
00278 {
00279     player_cmd_t *cmd;
00280     int r = PLAYER_SUCCESS;
00281     play_para_t *player_para;
00282     player_status sta;
00283
00284     player_para = player_open_pid_data(pid);
00285
00286     if (player_para == NULL) {
00287         return PLAYER_NOT_VALID_PID;
00288     }
00289
00290     sta = get_player_state(player_para);
00291     log_print("[player_stop]player_status=%x
00292 \n", sta);
00292     if (PLAYER_THREAD_IS_STOPPED(sta)) {
00293         player_close_pid_data(pid);
00294         log_print("[player_stop]pid=%d threa
00295 d is already stopped\n", pid);
```

```
00295         return PLAYER_SUCCESS;
00296     }
00297     clear_all_message(player_para); /*clear o
ld message to make sure fast exit.*/
00298     cmd = message_alloc();
00299     if (cmd) {
00300         cmd->ctrl_cmd = CMD_STOP;
00301         ffmpeg_interrupt(player_para->thread
_mgt.pthread_id);
00302         r = send_message(player_para, cmd);
00303     } else {
00304         r = PLAYER_NOMEM;
00305     }
00306
00307     player_close_pid_data(pid);
00308
00309     return r;
00310 }
00311
00312
00313
00314
00315 /* -----
00316 -----*/
00329 /* -----
00330 -----*/
00330 int player_exit(int pid)
00331 {
00332     int ret = PLAYER_SUCCESS;
00333     play_para_t *para;
00334
00335     log_print("[player_exit:enter]pid=%d\n",
00336     pid);
00337     para = player_open_pid_data(pid);
00338     if (para != NULL) {
00339         log_print("[player_exit]player_state
```

```
=0x%x\n", get_player_state(para));
00340         if (get_player_state(para) != PLAYER
00341             _EXIT) {
00341             player_stop(pid);
00342         }
00343
00344         ret = player_thread_wait_exit(para);
00345         log_print("[player_exit]player threa
d already exit: %d\n", ret);
00346         ffmpeg_uninterrupt(para->thread_mgt.
pthread_id);
00347         FREE(para);
00348         para = NULL;
00349     }
00350     player_close_pid_data(pid);
00351     player_release_pid(pid);
00352     log_print("[player_exit:exit]pid=%d\n",
00353 pid);
00354
00354     return ret;
00355 }
00356
00357 /* -----
00357 -----*/
00371 /* -----
00371 -----*/
00372 int player_pause(int pid)
00373 {
00374     player_cmd_t cmd;
00375     int ret = PLAYER_SUCCESS;
00376
00377     log_print("[player_pause:enter]pid=%d\n"
00377 , pid);
00378
00379     MEMSET(&cmd, 0, sizeof(player_cmd_t));
00380
00381     cmd.ctrl_cmd = CMD_PAUSE;
```

```
00382
00383     ret = player_send_message(pid, &cmd);
00384     log_print("[player_pause:exit]pid=%d ret
00385     =%d\n", pid, ret);
00386     return ret;
00387 }
00388
00389 /* -----
00403 /* -----
00404 int player_resume(int pid)
00405 {
00406     player_cmd_t cmd;
00407     int ret;
00408
00409     log_print("[player_resume:enter]pid=%d\n"
00410     , pid);
00411     MEMSET(&cmd, 0, sizeof(player_cmd_t));
00412
00413     cmd.ctrl_cmd = CMD_RESUME;
00414
00415     ret = player_send_message(pid, &cmd);
00416     log_print("[player_resume:exit]pid=%d re
00417     t=%d\n", pid, ret);
00418     return ret;
00419 }
00420
00421 /* -----
00435 /* -----
00436 int player_loop(int pid)
00437 {
```

```
00438     player_cmd_t cmd;
00439     int ret;
00440
00441     log_print("[player_loop:enter]pid=%d\n",
00442             pid);
00443     MEMSET(&cmd, 0, sizeof(player_cmd_t));
00444
00445     cmd.set_mode = CMD_LOOP;
00446
00447     ret = player_send_message(pid, &cmd);
00448     log_print("[player_loop:exit]pid=%d ret=%d\n", pid, ret);
00449
00450     return ret;
00451 }
00452
00453 /* -----
00454 * -----
00455 */
00467 /* -----
00468 * -----
00469 */
00469 int player_noloop(int pid)
00470 {
00471     player_cmd_t cmd;
00472     int ret;
00473
00474     log_print("[player_loop:enter]pid=%d\n",
00475             pid);
00476     MEMSET(&cmd, 0, sizeof(player_cmd_t));
00477
00478     cmd.set_mode = CMD_NOLOOP;
00479
00480     ret = player_send_message(pid, &cmd);
00481     log_print("[player_loop:exit]pid=%d ret=%d\n", pid, ret);
```

```
00482
00483     return ret;
00484 }
00485
00486 /* -----
00487 -----*/
00501 /* -----
00502     */
00502 int player_timesearch(int pid, float s_time)
00503 {
00504     player_cmd_t cmd;
00505     int ret;
00506     log_print("[player_timesearch:enter]pid=%d s_time=%f\n", pid, s_time);
00507
00508     MEMSET(&cmd, 0, sizeof(player_cmd_t));
00509
00510     cmd.ctrl_cmd = CMD_SEARCH;
00511     cmd.f_param = s_time;
00512
00513     ret = player_send_message(pid, &cmd);
00514     log_print("[player_timesearch:exit]pid=%d ret=%d\n", pid, ret);
00515
00516     return ret;
00517 }
00518
00519 /* -----
00520 -----*/
00534 /* -----
00535 -----*/
00535 int player_forward(int pid, int speed)
00536 {
00537     player_cmd_t cmd;
00538     int ret;
00539
00540     log_print("[player_forward:enter]pid=%d
```

```
speed=%d\n", pid, speed);
00541
00542     MEMSET(&cmd, 0, sizeof(player_cmd_t));
00543
00544     cmd.ctrl_cmd = CMD_FF;
00545     cmd.param = speed;
00546
00547     ret = player_send_message(pid, &cmd);
00548     log_print("[player_forward:exit]pid=%d r
et=%d\n", pid, ret);
00549
00550     return ret;
00551 }
00552
00553 /* -----
00568 * -----
00569 int player_backward(int pid, int speed)
00570 {
00571     player_cmd_t cmd;
00572     int ret;
00573
00574     log_print("[player_backward:enter]pid=%d
speed=%d\n", pid, speed);
00575
00576     MEMSET(&cmd, 0, sizeof(player_cmd_t));
00577
00578     cmd.ctrl_cmd = CMD_FB;
00579     cmd.param = speed;
00580
00581     ret = player_send_message(pid, &cmd);
00582     log_print("[player_backward]cmd=%x param
=%d ret=%d\n", cmd.ctrl_cmd, cmd.param, ret);
00583
00584     return ret;
00585 }
```

```
00586
00587 /* -----
00588 * -----
00589 */
00603 /* -----
00604 * -----
00605 */
00606     player_cmd_t cmd;
00607     int ret;
00608
00609     log_print("[player_aid:enter]pid=%d aid=%d\n", pid, audio_id);
00610
00611     MEMSET(&cmd, 0, sizeof(player_cmd_t));
00612
00613     cmd.ctrl_cmd = CMD_SWITCH_AID;
00614     cmd.param = audio_id;
00615
00616     ret = player_send_message(pid, &cmd);
00617     log_print("[player_aid:exit]pid=%d ret=%d\n", pid, ret);
00618
00619     return ret;
00620
00621 }
00622
00623 /* -----
00624 * -----
00625 */
00639 /* -----
00640 * -----
00641 */
00642     player_cmd_t cmd;
00643     int ret;
00644
00645     log_print("[player_sid:enter]pid=%d sub_id=%d\n", pid, sub_id);
```

```
00646
00647     MEMSET(&cmd, 0, sizeof(player_cmd_t));
00648
00649     cmd.ctrl_cmd = CMD_SWITCH_SID;
00650     cmd.param = sub_id;
00651
00652     ret = player_send_message(pid, &cmd);
00653     log_print("[player_sid:exit]pid=%d sub_id=%d\n", pid, sub_id);
00654
00655     return ret;
00656
00657 }
00658
00659 /* -----
00660 -----*/
00674 /* -----
00675 -----*/
00675 int player_enable_autobuffer(int pid, int enable)
00676 {
00677     player_cmd_t cmd;
00678     int ret;
00679
00680     log_print("[%s:enter]pid=%d enable=%d\n"
00681             , __FUNCTION__, pid, enable);
00682
00683     MEMSET(&cmd, 0, sizeof(player_cmd_t));
00684
00685     cmd.set_mode = CMD_EN_AUTOBUF;
00686     cmd.param = enable;
00687
00688     ret = player_send_message(pid, &cmd);
00689     log_print("[%s:exit]pid=%d enable=%d\n",
00690             __FUNCTION__, pid, enable);
00691
00692     return ret;
```

```
00691
00692 }
00693
00694 /* -----
00695 ----- */
00712 /* -----
00713 ----- */
00713 int player_set_autobuffer_level(int pid, float min, float middle, float max)
00714 {
00715     player_cmd_t cmd;
00716     int ret;
00717
00718     log_print("[%s:enter]pid=%d min=%.3f middle=%.3f max=%.3f\n", __FUNCTION__, pid, min, middle, max);
00719
00720     if (min < middle && middle < max && max < 1) {
00721         MEMSET(&cmd, 0, sizeof(player_cmd_t));
00722
00723         cmd.set_mode = CMD_SET_AUTOBUF_LEV;
00724         cmd.f_param = min;
00725         cmd.f_param1 = middle;
00726         cmd.f_param2 = max;
00727
00728         ret = player_send_message(pid, &cmd);
00729     } else {
00730         ret = -1;
00731         log_error("[%s]invalid param, please check!\n", __FUNCTION__);
00732     }
00733     log_print("[%s:exit]pid=%d min=%.3f middle=%.3f max=%.3f\n", __FUNCTION__, pid, min, middle, max);
```

```
00734
00735     return ret;
00736
00737 }
00738
00739
00740 /* -----
00741 -----*/
00755 /* -----
00756     */
00756 int player_send_message(int pid, player_cmd_
t *cmd)
00757 {
00758     player_cmd_t *mycmd;
00759     int r = -1;
00760     play_para_t *player_para;
00761     char buf[512];
00762
00763     player_para = player_open_pid_data(pid);
00764     if (player_para == NULL) {
00765         return PLAYER_NOT_VALID_PID;
00766     }
00767
00768     if (player_get_state(pid) == PLAYER_EXIT
00769 ) {
00770         player_close_pid_data(pid);
00771         return PLAYER_SUCCESS;
00772     }
00773
00773     mycmd = message_alloc();
00774     if (mycmd) {
00775         memcpy(mycmd, cmd, sizeof(*cmd));
00776         r = send_message_by_pid(pid, mycmd);
00777         if (cmd2str(cmd, buf) != -1) {
00778             log_print("[%s]cmd = %s\n", __FU
00779             NCTION__, buf);
00779         }
```

```
00780     } else {
00781         r = PLAYER_NOMEM;
00782     }
00783     player_close_pid_data(pid);
00784     return r;
00785 }
00786
00787 /* -----
00788 * -----
00789 * -----
00800 */
00803 /* -----
00804     -----
00805 */
00804 int player_register_update_callback(callback
00805 _t *cb, update_state_fun_t up_fn, int interval_s)
00805 {
00806     int ret;
00807     if (!cb) {
00808         log_error("[player_register_update_c
00809 allback]empty callback pointer!\n");
00810         return PLAYER_EMPTY_P;
00811     }
00812     ret = register_update_callback(cb, up_fn
00813 , interval_s);
00814     return ret;
00815 }
00816
00817 /* -----
00818 * -----
00819 * -----
00820 */
00830 /* -----
00831 */
00831 player_status player_get_state(int pid)
00832 {
00833     player_status status;
00834     play_para_t *player_para;
00835
00836     player_para = player_open_pid_data(pid);
```

```
00837     if (player_para == NULL) {
00838         return PLAYER_NOT_VALID_PID;
00839     }
00840
00841     status = get_player_state(player_para);
00842     player_close_pid_data(pid);
00843
00844     return status;
00845 }
00846
00847 /* -----
00848 -----*/
00860 /* -----
00861 -----*/
00861 unsigned int player_get_extern_priv(int pid)
00862 {
00863     unsigned long extersed;
00864     play_para_t *player_para;
00865
00866     player_para = player_open_pid_data(pid);
00867     if (player_para == NULL) {
00868         return PLAYER_NOT_VALID_PID; /*th
00869         is data is 0 for default!*/
00870     }
00871
00872     extersed = player_para->extern_priv;
00873     player_close_pid_data(pid);
00874
00875     return extersed;
00876
00877
00878 /* -----
00879 -----*/
00892 /* -----
00903 -----*/
00893 int player_get_play_info(int pid, player_inf
```

```
o_t *info)
00894 {
00895     play_para_t *player_para;
00896
00897     player_para = player_open_pid_data(pid);
00898     if (player_para == NULL) {
00899         return PLAYER_NOT_VALID_PID; /*th
is data is 0 for default!*/
00900     }
00901
00902     MEMSET(info, 0, sizeof(player_info_t));
00903     MEMCPY(info, &player_para->state, sizeof(
player_info_t));
00904     player_close_pid_data(pid);
00905
00906     return PLAYER_SUCCESS;
00907 }
00908 /* -----
00920 /* -----
00921 int64_t player_get_lpbuffbuffedsize(int pid)
00922 {
00923     int64_t buffedsize = -1;
00924     play_para_t *player_para;
00925
00926     player_para = player_open_pid_data(pid);
00927     if (player_para == NULL) {
00928         return PLAYER_NOT_VALID_PID;
00929     }
00930
00931     buffedsize = getl pBuffer_buffedsiz
er_para);
00932     player_close_pid_data(pid);
00933
00934     return buffedsize;
00935 }
```

```
00936 /* -----
00948 /* -----
00949 int64_t player_get_streambufbuffedszie(int pid)
00950 {
00951     int64_t buffedszie = -1;
00952     play_para_t *player_para;
00953
00954     player_para = player_open_pid_data(pid);
00955     if (player_para == NULL) {
00956         return PLAYER_NOT_VALID_PID;
00957     }
00958
00959     buffedszie = getstreambuffer_buffedszie(
00960         player_para);
00961     player_close_pid_data(pid);
00962
00963 }
00964
00965 /* -----
00979 /* -----
00980 int player_get_media_info(int pid, media_info_t *minfo)
00981 {
00982     play_para_t *player_para;
00983     player_status sta;
00984
00985     while (player_get_state(pid) < PLAYE
R_INITOK) {
00986         sta = player_get_state(pid);
00987         if (sta == NULL){
00988             log_error("player_ge
```

```
t_media_info failed pid [%d]\n",pid);
00989                                     return PLAYER_FAILED
;
00990                                     }
00991                                     if (sta >= PLAYER_ERROR && s
ta <= PLAYER_EXIT) {
00992                                         player_close_pid_dat
a(pid);
00993                                         log_error("player_ge
t_media_info status err [0x%x]\n",sta);
00994                                         return PLAYER_INVALID
D_CMD;
00995                                     }
00996                                     if ((player_get_state(pid))
== PLAYER_ERROR ||
00997                                         player_get_state(pid
) == PLAYER_STOPED ||
00998                                         player_get_state(pid
) == PLAYER_PLAYEND ||
00999                                         player_get_state(pid
) == PLAYER_EXIT) {
01000                                         log_error("player_ge
t_media_info failed status [0x%x]\n",sta);
01001                                         return PLAYER_FAILED
;
01002                                     }
01003                                         usleep(1000 * 10);
01004                                     }
01005
01006     player_para = player_open_pid_data(pid);
01007     if (player_para == NULL) {
01008         return PLAYER_NOT_VALID_PID; /*th
is data is 0 for default!*/
01009     }
01010
01011     MEMSET(minfo, 0, sizeof(media_info_t));
01012     MEMCPY(minfo, &player_para->media_info,
```

```
    sizeof(media_info_t));
01013
01014     log_print("[player_get_media_info]video_
num=%d vidx=%d\n", minfo->stream_info.total_video_
num, minfo->stream_info.cur_video_index);
01015     player_close_pid_data(pid);
01016
01017     return PLAYER_SUCCESS;
01018 }
01019
01020 /* -----
01021 ----- */
01033 /* -----
01034 int player_video_overlay_en(unsigned enable)
01035 {
01036     int fd = open("/dev/graphics/fb0", O_RDWR);
01037     if (fd >= 0) {
01038         unsigned myKeyColor = 0;
01039         unsigned myKeyColor_en = enable;
01040
01041         if (myKeyColor_en) {
01042             myKeyColor = 0xff; /*set another
value to solved the bug in kernel..remove later*/
01043             ioctl(fd, FBIOPUT OSD_SRCCOLORKE
Y, &myKeyColor);
01044             myKeyColor = 0;
01045             ioctl(fd, FBIOPUT OSD_SRCCOLORKE
Y, &myKeyColor);
01046             ioctl(fd, FBIOPUT OSD_SRCKEY_ENA
BLE, &myKeyColor_en);
01047         } else {
01048             ioctl(fd, FBIOPUT OSD_SRCKEY_ENA
BLE, &myKeyColor_en);
01049         }
01050         close(fd);
```

```
01051         return PLAYER_SUCCESS;
01052     }
01053     return PLAYER_FAILED;
01054 }
01055
01056 /* -----
01057 * -----
01058 */
01059
01060 /* -----
01061 * -----
01062 */
01063
01064
01065     int ret = PLAYER_FAILED;
01066     play_para_t *player_para;
01067     codec_para_t *p;
01068
01069     player_para = player_open_pid_data(pid);
01070     if (player_para != NULL) {
01071         player_para->playctrl_info.audio_mute = mute_on & 0x1;
01072         log_print("[audio_set_mute:%d]muteon=%d audio_mute=%d\n", __LINE__, mute_on, player_para->playctrl_info.audio_mute);
01073
01074         p = get_audio_codec(player_para);
01075         if (p != NULL) {
01076             ret = codec_set_mute(p, mute_on)
01077         }
01078         player_close_pid_data(pid);
01079     } else {
01080         ret = codec_set_mute(NULL, mute_on);
01081     }
01082
01083     return ret;
01084 }
```

```
01095
01096 /* -----
01097 ----- */
01111 /* -----
01112 ----- */
01112 int audio_get_volume_range(int pid, float *min,
01113     in, float *max)
01113 {
01114     return codec_get_volume_range(NULL, min,
01115         max);
01115 }
01116
01117 /* -----
01118 ----- */
01131 /* -----
01132 ----- */
01132 int audio_set_volume(int pid, float val)
01133 {
01134     return codec_set_volume(NULL, val);
01135 }
01136
01137 /* -----
01138 ----- */
01149 /* -----
01150 ----- */
01150 int audio_get_volume(int pid, float *vol)
01151 {
01152     int r;
01153
01154     r = codec_get_volume(NULL, vol);
01155     log_print("[audio_get_volume:%d]r=%d\n",
01156     __LINE__, r);
01156
01157     return r;//codec_get_volume(NULL);
01158 }
01159
01160 /* -----
```

```
----- */
01175 /* -----
----- */
01176 int audio_set_lrvolume(int pid, float lvol,
float rvol)
01177 {
01178     play_para_t *player_para;
01179     log_print("[audio_set_lrvolume:enter]pid
=%d\n", pid);
01180     player_para = player_open_pid_data(pid);
01181     if(player_para == NULL){
01182         log_print("player ID is NULL!\n");
01183         return -1;
01184     }
01185     if(player_para->acodec == NULL){
01186         log_print("codec is not ready!\n");
01187         return -1;
01188     }
01189     return codec_set_lrvolume(player_para->a
codec, lvol, rvol);
01190 }
01191
01192 /* -----
----- */
01204 /* -----
----- */
01205 int audio_get_lrvolume(int pid, float *lvol,
float* rvol)
01206 {
01207     int r;
01208
01209     r = codec_get_lrvolume(NULL, lvol, rvol)
;
01210     log_print("[audio_get_volume:%d]r=%d\n",
__LINE__, r);
01211
01212     return r;//codec_get_volume(NULL);
```

```
01213 }
01214
01215
01216
01217 /* -----
01218 -----*/
01231 /* -----
01232 -----*/
01232 int audio_set_volume_balance(int pid, int ba
lance)
01233 {
01234     return codec_set_volume_balance(NULL, ba
lance);
01235 }
01236
01237 /* -----
01238 -----*/
01250 /* -----
01251 -----*/
01251 int audio_swap_left_right(int pid)
01252 {
01253     return codec_swap_left_right(NULL);
01254 }
01255
01256 /* -----
01257 -----*/
01268 /* -----
01269 -----*/
01269
01270 int audio_left_mono(int pid)
01271 {
01272     int ret = -1;
01273     play_para_t *player_para;
01274     codec_para_t *p;
01275
01276     player_para = player_open_pid_data(pid);
01277     if (player_para == NULL) {
```

```
01278         return 0;      /*this data is 0 for de
fault!*/
01279     }
01280
01281     p = get_audio_codec(player_para);
01282     if (p != NULL) {
01283         ret = codec_left_mono(p);
01284     }
01285     player_close_pid_data(pid);
01286
01287     return ret;
01288 }
01289
01290 /* -----
01291 -----*/
01302 /* -----
01303 -----*/
01304 int audio_right_mono(int pid)
01305 {
01306     int ret = -1;
01307     play_para_t *player_para;
01308     codec_para_t *p;
01309
01310     player_para = player_open_pid_data(pid);
01311     if (player_para == NULL) {
01312         return 0;      /*this data is 0 for de
fault!*/
01313     }
01314
01315     p = get_audio_codec(player_para);
01316     if (p != NULL) {
01317         ret = codec_right_mono(p);
01318     }
01319     player_close_pid_data(pid);
01320
01321     return ret;
01322 }
```

```
01322
01323 /* -----
01324 ----- */
01335 /* -----
01336 ----- */
01336 int audio_stereo(int pid)
01337 {
01338     int ret = -1;
01339     play_para_t *player_para;
01340     codec_para_t *p;
01341
01342     player_para = player_open_pid_data(pid);
01343     if (player_para == NULL) {
01344         return 0;      /*this data is 0 for de
fault!*/
01345     }
01346
01347     p = get_audio_codec(player_para);
01348     if (p != NULL) {
01349         ret = codec_stereo(p);
01350     }
01351     player_close_pid_data(pid);
01352
01353     return ret;
01354 }
01355
01356 /* -----
01357 ----- */
01367 /* -----
01368 ----- */
01368 int audio_lr_mix_set(int pid,int enable)
01369 {
01370     int ret = -1;
01371     play_para_t *player_para;
01372     codec_para_t *p;
01373     player_para = player_open_pid_data(pid);
01374     if (player_para == NULL) {
```

```
01375         log_print("[%s %d] player_para==NULL, s
et fail audio_lr_mix!!",__FUNCTION__,__LINE__);
01376             return -1; /*this data is 0 for d
efault!*/
01377     }
01378     p = get_audio_codec(player_para);
01379     if (p != NULL) {
01380         ret = codec_lr_mix_set(p,enable);
01381     }else{
01382         log_print("[%s %d] p==NULL, set fail
audio_lr_mix!!",__FUNCTION__,__LINE__);
01383     }
01384     player_close_pid_data(pid);
01385     return ret;
01386 }
01387
01388 int audio_cur_pcmpara_Applied_get(int pid,int
*pfs,int *pch)
01389 {
01390     int ret = -1;
01391     play_para_t *player_para;
01392     codec_para_t *p;
01393     player_para = player_open_pid_data(pid);
01394     if (player_para == NULL) {
01395         log_print("[%s %d] player_para==NULL
, set fail audio_FsNch_get!!",__FUNCTION__,__LINE__
);
01396         return -1; /*this data is 0 for d
efault!*/
01397     }
01398     p = get_audio_codec(player_para);
01399     if (p != NULL) {
01400         ret = codec_pcmpara_Applied_get(p,pf
s,pch);
01401     }else{
01402         log_print("[%s %d] p==NULL, set fail
audio_FsNch_get!!",__FUNCTION__,__LINE__);
```

```
01403     }
01404     player_close_pid_data(pid);
01405     return ret;
01406 }
01407
01408
01409 /* -----
01410 -----*/
01423 /* -----
01424 -----*/
01424 int audio_set_spectrum_switch(int pid, int i
sStart, int interval)
01425 {
01426     int ret = -1;
01427     play_para_t *player_para;
01428     codec_para_t *p;
01429
01430     player_para = player_open_pid_data(pid);
01431     if (player_para == NULL) {
01432         return 0;      /*this data is 0 for de
fault!*/
01433     }
01434
01435     p = get_audio_codec(player_para);
01436     if (p != NULL) {
01437         ret = codec_audio_spectrum_switch(p,
01438             isStart, interval);
01439     }
01440     player_close_pid_data(pid);
01441
01442     return ret;
01443 }
01444 /* -----
01445 -----*/
01454 /* -----
01455 -----*/
```

```
01455 int player_progress_exit(void)
01456 {
01457     codec_close_audio(NULL);
01458
01459     return 0;
01460 }
01461
01462 /* -----
01463 -----*/
01475 /* -----
01476 -----*/
01477 int player_list_allpid(pid_info_t *pid)
01478 {
01479     char buf[MAX_PLAYER_THREADS];
01480     int pnum = 0;
01481     int i;
01482
01483     pnum = player_list_pid(buf, MAX_PLAYER_T
HREADS);
01484     pid->num = pnum;
01485
01486     for (i = 0; i < pnum; i++) {
01487         pid->pid[i] = buf[i];
01488     }
01489
01490     return 0;
01491 }
01492
01493 /* -----
01494 -----*/
01508 /* -----
01509 -----*/
01510
01511 int player_cache_system_init(int enable, con
st char*dir, int max_size, int block_size)
```

```
01512 {
01513     return cache_system_init(enable, dir, ma
x_size, block_size);
01514 }
01515
01516 /* -----
01517 ----- */
01528 /* -----
01529 ----- */
01529 char *player_status2str(player_status status
)
01530 {
01531     switch (status) {
01532         case PLAYER_INITING:
01533             return "BEGIN_INIT";
01534
01535         case PLAYER_TYPE_READY:
01536             return "TYPE_READY";
01537
01538         case PLAYER_INITOK:
01539             return "INIT_OK";
01540
01541         case PLAYER_RUNNING:
01542             return "PLAYING";
01543
01544         case PLAYER_BUFFERING:
01545             return "BUFFERING";
01546
01547         case PLAYER_BUFFER_OK:
01548             return "BUFFEROK";
01549
01550         case PLAYER_PAUSE:
01551             return "PAUSE";
01552
01553         case PLAYER_SEARCHING:
01554             return "SEARCH_FFFB";
01555 }
```

```
01556     case PLAYER_SEARCHOK:  
01557         return "SEARCH_OK";  
01558  
01559     case PLAYER_START:  
01560         return "START_PLAY";  
01561  
01562     case PLAYER_FF_END:  
01563         return "FF_END";  
01564  
01565     case PLAYER_FB_END:  
01566         return "FB_END";  
01567  
01568     case PLAYER_ERROR:  
01569         return "ERROR";  
01570  
01571     case PLAYER_PLAYEND:  
01572         return "PLAY_END";  
01573  
01574     case PLAYER_STOPED:  
01575         return "STOPED";  
01576  
01577     case PLAYER_EXIT:  
01578         return "EXIT";  
01579  
01580     case PLAYER_PLAY_NEXT:  
01581         return "PLAY_NEXT";  
01582  
01583     case PLAYER_FOUND_SUB:  
01584         return "NEW_SUB";  
01585  
01586     case PLAYER_DIVX_AUTHERR:  
01587         return "DIVX_AUTHERR";  
01588     case PLAYER_DIVX_RENTAL_VIEW:  
01589         return "DIVX_RENTAL";  
01590     case PLAYER_DIVX_RENTAL_EXPIRED:  
01591         return "DIVX_EXPIRED";  
01592 default:
```

```
01593         return "UNKNOW_STATE";
01594     }
01595 }
01596
01597 static char* player_vformat2str(vformat_t va
lue)
01598 {
01599     switch(value) {
01600         case VFORMAT_MPEG12:
01601             return "VFORMAT_MPEG12";
01602
01603         case VFORMAT_MPEG4:
01604             return "VFORMAT_MPEG4";
01605
01606         case VFORMAT_H264:
01607             return "VFORMAT_H264";
01608
01609         case VFORMAT_HEVC:
01610             return "VFORMAT_HEVC";
01611
01612         case VFORMAT_MJPEG:
01613             return "VFORMAT_MJPEG";
01614
01615         case VFORMAT_REAL:
01616             return "VFORMAT_REAL";
01617
01618         case VFORMAT_JPEG:
01619             return "VFORMAT_JPEG";
01620
01621         case VFORMAT_VC1:
01622             return "VFORMAT_VC1";
01623
01624         case VFORMAT_AVIS:
01625             return "VFORMAT_AVIS";
01626
01627         case VFORMAT_SW:
01628             return "VFORMAT_SW";
```

```
01629
01630     case VFORMAT_H264MVC:
01631         return "VFORMAT_H264MVC";
01632
01633     case VFORMAT_H264_4K2K:
01634         return "VFORMAT_H264_4K2K";
01635
01636     default:
01637         return "NOT_SUPPORT VFORMAT";
01638 }
01639 return NULL;
01640 }
01641
01642 static char* player_aformat2str(aformat_t value)
01643 {
01644     switch(value) {
01645         case AFORMAT_MPEG:
01646             return "AFORMAT_MPEG";
01647
01648         case AFORMAT_PCM_S16LE:
01649             return "AFORMAT_PCM_S16LE";
01650
01651         case AFORMAT_AAC:
01652             return "AFORMAT_AAC";
01653
01654         case AFORMAT_AC3:
01655             return "AFORMAT_AC3";
01656
01657         case AFORMAT_ALAW:
01658             return "AFORMAT_ALAW";
01659
01660         case AFORMAT_MULAW:
01661             return "AFORMAT_MULAW";
01662
01663         case AFORMAT_DTS:
01664             return "AFORMAT_DTS";
```

```
01665
01666     case AFORMAT_PCM_S16BE:
01667         return "AFORMAT_PCM_S16BE";
01668
01669     case AFORMAT_FLAC:
01670         return "AFORMAT_FLAC";
01671
01672     case AFORMAT_COOK:
01673         return "AFORMAT_COOK";
01674
01675     case AFORMAT_PCM_U8:
01676         return "AFORMAT_PCM_U8";
01677
01678     case AFORMAT_ADPCM:
01679         return "AFORMAT_ADPCM";
01680
01681     case AFORMAT_AMR:
01682         return "AFORMAT_AMR";
01683
01684     case AFORMAT_RAAC:
01685         return "AFORMAT_RAAC";
01686
01687     case AFORMAT_WMA:
01688         return "AFORMAT_WMA";
01689
01690     case AFORMAT_WMAPRO:
01691         return "AFORMAT_WMAPRO";
01692
01693     case AFORMAT_PCM_BLURAY:
01694         return "AFORMAT_PCM_BLURAY";
01695
01696     case AFORMAT_ALAC:
01697         return "AFORMAT_ALAC";
01698
01699     case AFORMAT_VORBIS:
01700         return "AFORMAT_VORBIS";
01701
```

```
01702     case AFORMAT_AAC_LATM:  
01703         return "AFORMAT_AAC_LATM";  
01704  
01705     case AFORMAT_APE:  
01706         return "AFORMAT_APE";  
01707  
01708     case AFORMAT_EAC3:  
01709         return "AFORMAT_EAC3";  
01710     case AFORMAT_TRUEHD:  
01711         return "AFORMAT_TRUE  
01712 HD";  
01713     default:  
01714         return "NOT_SUPPORT_AFFORMAT";  
01715     }  
01716 }  
01717 /* ----- */  
01733 /* ----- */  
01734 char *player_value2str(char *key, int value)  
01735 {  
01736     if(strcasecmp(key, "status") == 0)  
  
01737         return player_status2str((player_sta  
01738 tus)value);  
01738     else if(strcasecmp(key, "vformat") == 0)  
  
01739         return player_vformat2str((vformat_t  
01739 )value);  
01740     else if(strcasecmp(key, "aformat") == 0)  
01741         return player_aformat2str((aformat_t  
01741 )value);  
01742     else  
01743         return ("INVALID KEYWORDS");  
01744 }  
01745
```

```
01746 int audio_get_decoder_enable(int pid)
01747 {
01748
01749     int ret = -1;
01750     play_para_t *player_para;
01751     codec_para_t *p;
01752
01753     player_para = player_open_pid_data(pid);
01754     if (player_para == NULL) {
01755         return -1;      /*this data is 0 for d
efault!*/
01756     }
01757     p = get_audio_codec(player_para);
01758     if (p != NULL) {
01759         ret = codec_get_decoder_enable(p);
01760     }
01761     player_close_pid_data(pid);
01762
01763     return ret;
01764 }
```

Generated on Tue Dec 2 2014 21:55:13 for amplayerMy Project by
[doxygen](#) 1.7.6.1

amplayerMy Project

Main Page

Classes

Files

File List

File Members

player_id.h

```
00001 #ifndef PLAYER_ID_MGT_
00002 #define PLAYER_ID_MGT_
00003
00004 int player_request_pid(void);
00005 int player_release_pid(int pid);
00006 int player_init_pid_data(int pid,void * data
);
00007 void * player_open_pid_data(int pid);
00008 int player_close_pid_data(int pid);
00009 int player_id_pool_init(void);
00010 int player_list_pid(char id[],int size);
00011
00012 #endif
00013
00014
```

Generated on Tue Dec 2 2014 21:55:13 for amplayerMy Project by
[doxygen](#) 1.7.6.1