Linux AmpLayer Design Spec

版本信息

日期	版本	修改者	说明
2010-3-3	V1.0	Zhouzhi： Zhi.zhou@amlogic.com	创建
2010-4-16	V1.1	Xuhui： hui.xu@amlogic.com	完善 player
2010-4-20	V1.2	Xuhui: hui.xu@amlogic.com	更新文档格式布局，补充文档内容
2010-4-27	V1.3	Gongping： ping.gong@amlogoc.com	添加 d-bus 接口说明

目　　录

LINUX AMPLAYER DESIGN SPEC 0

版本信息 1

1　播放器基本结构图 1

1.1 本地播放器 1
1.2　DVB TS 流播放的应用 2
1.3　嵌入 UI 进程模式 2

2 AMCODEC 3

2.1 概述 3
2.2 特征 3
2.3 数据结构 3
2.3.1 宏 3
2.3.2 枚举 6
2.3.3 结构体 8
2.4　函数接口 12
2.4.1 通用 codec 接口 12
2.4.2 音频 codec 接口 18

3　PLAYER 21

3.1 概述 21
3.2 特征 21
3.2.1 支持文件格式 21
3.2.2 支持的音频解码格式 21
3.2.3 支持的视频解码格式 22
3.3 数据结构 22
3.3.1 宏 22
3.3.2 枚举 24
3.3.3 结构体 25
3.4 函数接口 30
3.4.1 player 接口 30
3.4.2 音频控制接口 37
3.4.3 消息接口 41
3.5 播放器控制模式 43
3.5.1 Socket 控制接口 43
3.5.2 D-BUS 控制接口 45

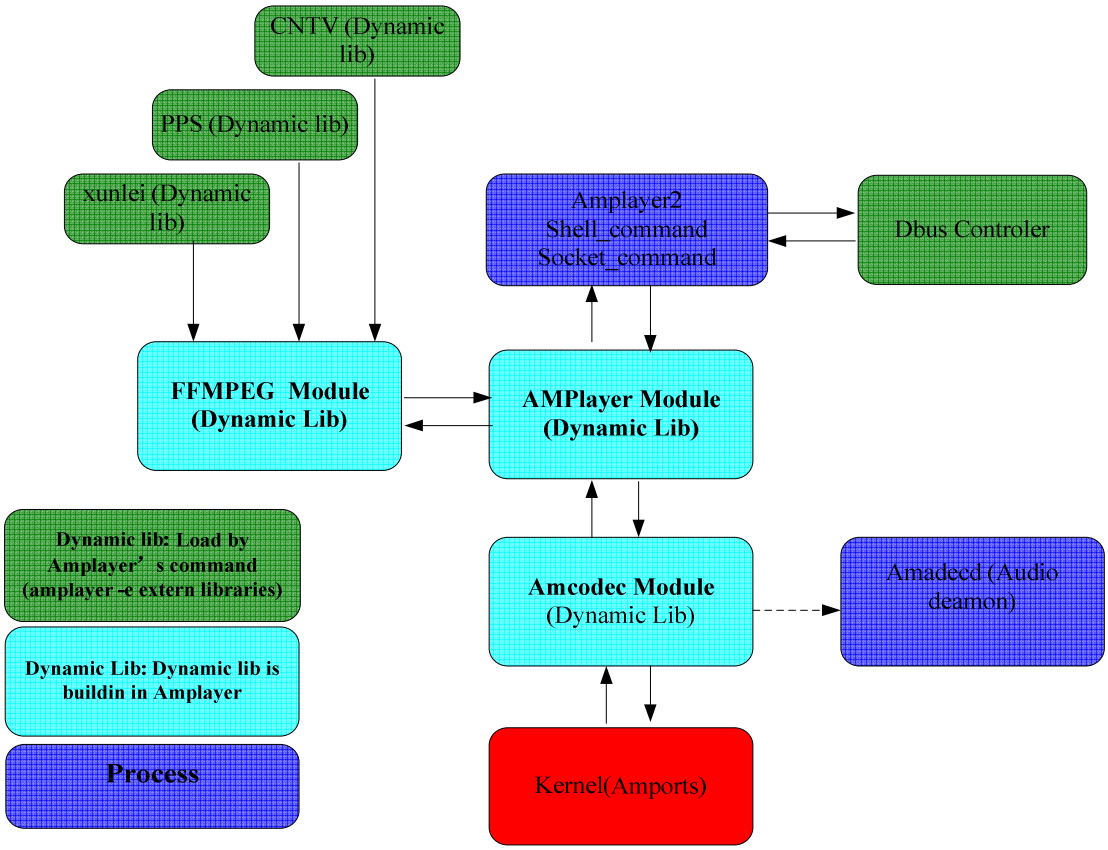
4　编写外接输入流库 49

4.1 头文件 49
4.2 输入库的结构体 49
4.3 库的注册 49

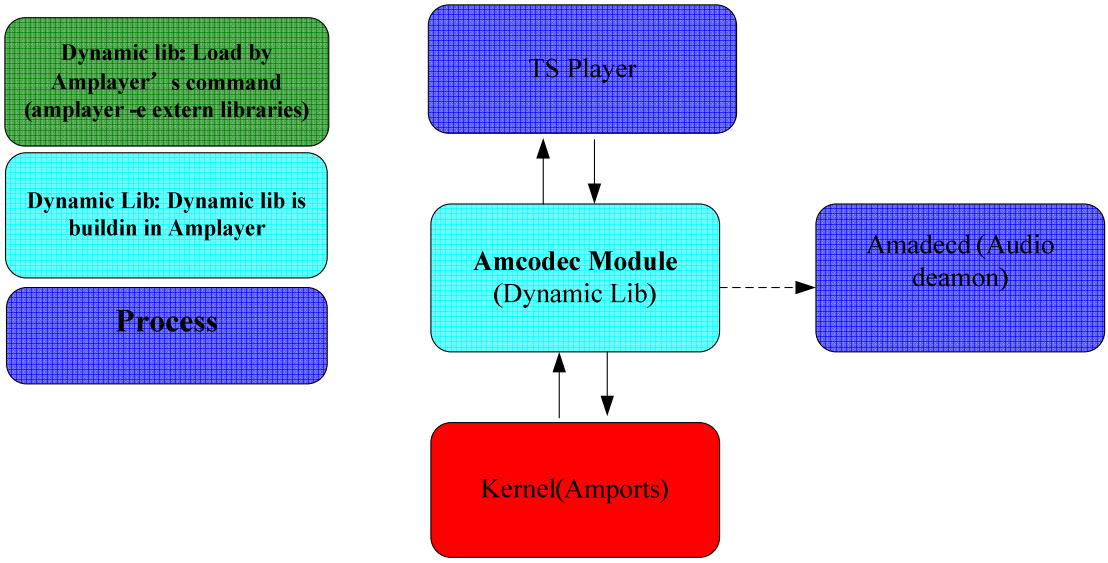
5 AMPLAYER 的外接控制器接口 51

1 播放器基本结构图

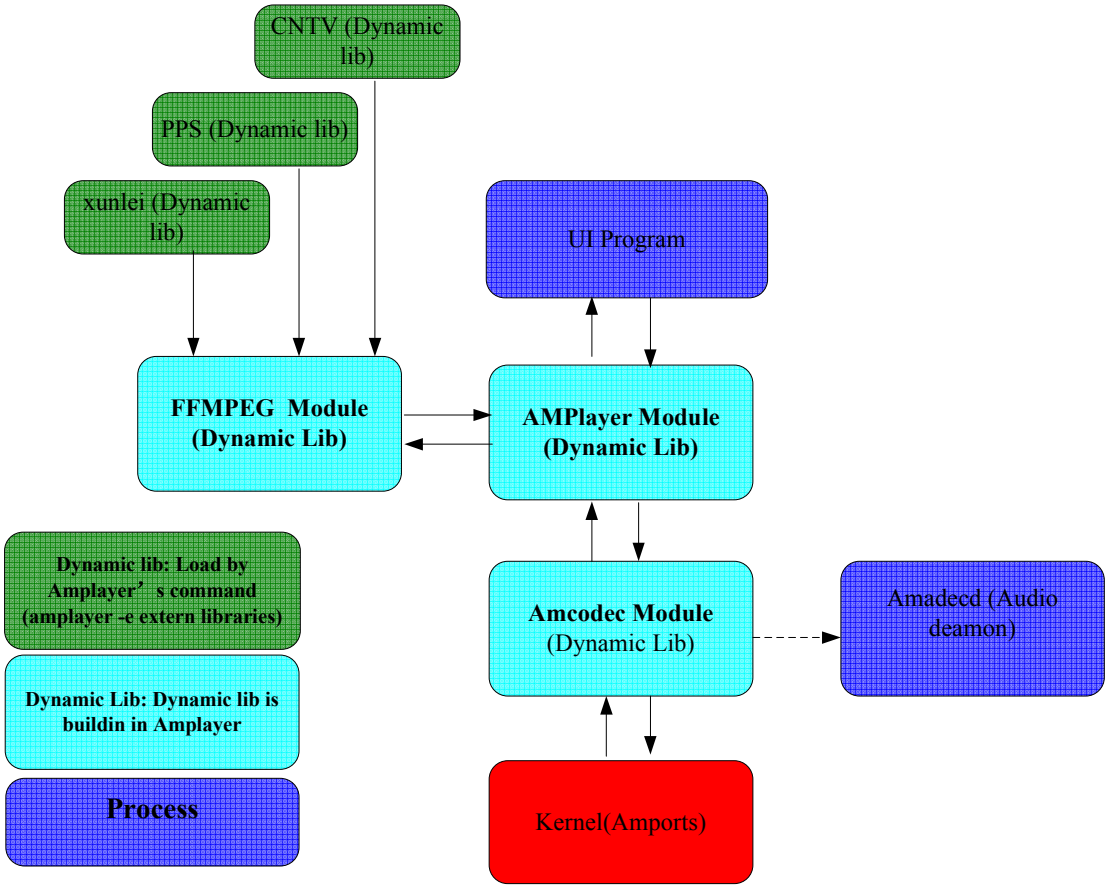
1.1 本地播放器



1.2 DVB TS 流播放的应用



1.3 嵌入 UI 进程模式



2 Amcodec

2.1 概述

Amcodec 是播放器与解码核的接口，主要负责解码核的设置与控制。通过调用 amcodec 接口函数，为解码核配置不同格式的解码信息，从而实现解码核控制。

2.2 特征

2.3 数据结构

2.3.1 宏

（1） 有效性判断

- **IS_VALID_PID**

  【定义】

  ```
  #define IS_VALID_PID(t)        (t>=0 && t<=0x1fff)
  ```

  【描述】

  判断流的 PID 是否有效

- **IS_VALID_STREAM**

  【定义】

  ```
  #define IS_VALID_STREAM(t) (t>0 && t<=0x1fff)
  ```

  【描述】

  判断流是否有效

- **IS_VALID_ATYPE**

  【定义】

  ```
  #define IS_VALID_ATYPE(t)     (t>=0 && t<AFORMAT_MAX)
  ```

  【描述】

  判断音频编码类型是否支持

- **IS_VALID_VTYPE**

  【定义】

  ```
  #define IS_VALID_VTYPE(t)     (t>=0 && t<VFORMAT_MAX)
  ```

  【描述】

  判断视频编码类型是否支持

（2） 视频编解码类型标记

- **MJPEG 标记**

【定义】

```
#define CODEC_TAG_MJPEG      (0x47504a4d)
#define CODEC_TAG_mjpeg      (0x47504a4c)
#define CODEC_TAG_jpeg       (0x6765706a)
```

- **MPEG4_3 标记**

【定义】

```
#define CODEC_TAG_MP43       (0x3334504d)
#define CODEC_TAG_DIV3       (0x33564944)
#define CODEC_TAG_COL1       (0x314c4f43)
```

- **MPEG4_4 标记**

【定义】

```
#define CODEC_TAG_DIV4       (0x34564944)
#define CODEC_TAG_DIVX       (0x58564944)
```

- **MPEG4_5 标记**

【定义】

```
#define CODEC_TAG_XVID       (0x44495658)
#define CODEC_TAG_xvid       (0x64697678)
#define CODEC_TAG_XVIX       (0x58495658)
#define CODEC_TAG_xvix       (0x78697678)
#define CODEC_TAG_DIV5       (0x35564944)
#define CODEC_TAG_DX50       (0x30355844)
#define CODEC_TAG_M4S2       (0x3253344d)
#define CODEC_TAG_DIV6       (0x36564944)
#define CODEC_TAG_RMP4       (0x34504d52)
#define CODEC_TAG_MP42       (0x3234504d)
#define CODEC_TAG_MPG4       (0x3447504d)
#define CODEC_TAG_MP4V       (0x5634504d)
#define CODEC_TAG_mp4v       (0x7634706d)
#define CODEC_TAG_MP4        (0x8e22ada)
```

- **H264 标记**

【定义】

```
#define CODEC_TAG_AVC1       (0x31435641)
#define CODEC_TAG_avc1       (0x31637661)
#define CODEC_TAG_H264       (0x34363248)
#define CODEC_TAG_h264       (0x34363268)
```

（3）codec 错误信息定义

- Codec 错误前缀

【定义】

```
#define C_PAE                    (0x01000000)
```

- 无错误

【定义】

```
#define CODEC_ERROR_NONE     ( 0)
```

- 无效指针

【定义】

```
#define CODEC_ERROR_INVAL    (C_PAE | 1)
```

- 无足够空间

【定义】

```
#define CODEC_ERROR_NOMEM  (C_PAE | 2)
```

- 设备已经打开或设备忙

【定义】

```
#define CODEC_ERROR_BUSY     (C_PAE | 3)
```

- 端口错误

【定义】

```
#define CODEC_ERROR_IO       (C_PAE | 4)
```

- 参数错误

【定义】

```
#define CODEC_ERROR_PARAMETER                (C_PAE | 5)
```

- 未知音频类型

【定义】

```
#define CODEC_ERROR_AUDIO_TYPE_UNKNOW        (C_PAE | 6)
```

- 未知视频类型

【定义】

```
#define CODEC_ERROR_VIDEO_TYPE_UNKNOW        (C_PAE | 7)
```

- 未知流类型

【定义】

```
#define CODEC_ERROR_STREAM_TYPE_UNKNOW       (C_PAE | 8)
```

- 错误视频编码类型

【定义】

```
#define CODEC_ERROR_VDEC_TYPE_UNKNOW         (C_PAE | 9)
```

- 初始化失败

【定义】

#define CODEC_ERROR_INIT_FAILED (C_PAE | 10)

- 设置 Buffer 大小失败

【定义】

#define CODEC_ERROR_SET_BUFSIZE_FAILED (C_PAE | 11)

4、设备定义

- ES 流视频解码 Buffer

【定义】

#define CODEC_VIDEO_ES_DEVICE "/dev/amstream_vbuf"

- ES 流音频解码 Buffer

【定义】

#define CODEC_AUDIO_ES_DEVICE "/dev/amstream_abuf"

- TS 流解码 Buffer

【定义】

#define CODEC_TS_DEVICE "/dev/amstream_mpts"

- PS 流解码 Buffer

【定义】

#define CODEC_PS_DEVICE "/dev/amstream_mpps"

- Real 流解码 Buffer

【定义】

#define CODEC_RM_DEVICE "/dev/amstream_rm"

- 控制设备

【定义】

#define CODEC_CNTL_DEVICE "/dev/amvideo"

2.3.2 枚举

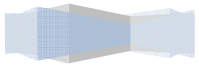
（1） 流的编码格式

- **stream_type**

【描述】

流类型定义

【定义】



```
typedef enum
{
    STREAM_TYPE_UNKNOW,
    STREAM_TYPE_ES_VIDEO,
    STREAM_TYPE_ES_AUDIO,
    STREAM_TYPE_PS,
    STREAM_TYPE_TS,
    STREAM_TYPE_RM,
}stream_type;
```

【参数】

参数	描述
STREAM_TYPE_UNKNOW	未知流类型
STREAM_TYPE_ES_VIDEO	ES 视频流
STREAM_TYPE_ES_AUDIO,	ES 音频流
STREAM_TYPE_PS	PS 流
STREAM_TYPE_TS	TS 流
STREAM_TYPE_RM	RM 流

- **vdec_type_t**

【描述】

播放器视频格式定义

【定义】

```
typedef enum
{
        VIDEO_DEC_FORMAT_UNKNOW,
        VIDEO_DEC_FORMAT_MPEG4_3,
        VIDEO_DEC_FORMAT_MPEG4_4,
        VIDEO_DEC_FORMAT_MPEG4_5,
        VIDEO_DEC_FORMAT_H264,
        VIDEO_DEC_FORMAT_MJPEG,
        VIDEO_DEC_FORMAT_MP4,
        VIDEO_DEC_FORMAT_H263,
        VIDEO_DEC_FORMAT_REAL_8,
        VIDEO_DEC_FORMAT_REAL_9,
}vdec_type_t;
```

【参数】

参数	描述
VIDEO_DEC_FORMAT_UNKNOW	未知视频解码类型
VIDEO_DEC_FORMAT_MPEG4_3	DIVX3 及兼容类型

VIDEO_DEC_FORMAT_MPEG4_4	DIVX4 及兼容类型
VIDEO_DEC_FORMAT_MPEG4_5,	DIVX5 及兼容类型
VIDEO_DEC_FORMAT_H264	H264 类型
VIDEO_DEC_FORMAT_MJPEG	MJPEG 类型
VIDEO_DEC_FORMAT_MP4	MP4 类型
VIDEO_DEC_FORMAT_H263	H263 类型
VIDEO_DEC_FORMAT_REAL_8	REAL8 类型
VIDEO_DEC_FORMAT_REAL_9	REAL9 类型

- **vformat_t**

【描述】

视频解码格式定义

【定义】

```
typedef enum {
    VFORMAT_MPEG12 = 0,
    VFORMAT_MPEG4,
    VFORMAT_H264,
    VFORMAT_MJPEG,
    VFORMAT_REAL,
    VFORMAT_MAX
} vformat_t;
```

【参数】

参数	描述
VFORMAT_MPEG12	MPEG1,MPEG2 编码格式
VFORMAT_MPEG4,	MPEG4 编码格式
VFORMAT_H264	H264 编码格式
VFORMAT_MJPEG	MJPEG 编码格式
VFORMAT_REAL	REAL 编码格式
VFORMAT_MAX	未知编码格式

2.3.3 结构体

- **dec_sysinfo_t**

【描述】

编码信息结构体

【定义】

typedef struct

```
{
    unsigned int    format;
    unsigned int    width;
    unsigned int    height;
    unsigned int    rate;
    unsigned int    extra;
    unsigned int    status;
    unsigned int    ratio;
    void *          param;
} dec_sysinfo_t;
```

【参数】

参数	类型	描述
format	unsigned int	视频编码格式
width	unsigned int	视频帧宽度（像素点）
height	unsigned int	视频帧高度（像素点）
rate	unsigned int	视频帧率
extra	unsigned int	视频编码附加信息
status	unsigned int	视频状态
ratio	unsigned int	视频显示比例
param	void *	其他参数

- **real_cookinfo_t**

【描述】

音频 real cook 解码头信息

【定义】

```
typedef struct
{
    int valid;
    char extradata[2044];
}real_cookinfo_t;
```

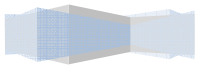
【参数】

参数	类型	描述
valid	int	是否有效标记
extradata	char[2044]	头信息数组

- **codec_para_t**

【描述】

Codec 控制器结构体



【定义】

```
typedef struct
{
    CODEC_HANDLE handle;
    CODEC_HANDLE cntl_handle;
    stream_type stream_type;
    unsigned int has_video:1;
    unsigned int    has_audio:1;
    int video_type;
    int audio_type;
    int video_pid;
    int audio_pid;
    int audio_channels;
    int audio_samplerate;
    int vbuf_size;
    int abuf_size;
    dec_sysinfo_t am_sysinfo;
    real_cookinfo_t audio_info;
}codec_para_t;
```

【参数】

参数	类型	描述
handle	CODEC_HANDLE	
cntl_handle	CODEC_HANDLE	
stream_type	stream_type	stream 的编码格式
has_video	unsigned int	是否存在 video 的标志
has_audio	unsigned int	是否存在 audio 的标志
video_type	int	video 的类型，对应于 vformat_t
audio_type	int	audio 的类型，对应于 aformat_t
video_pid	int	ps，ts 等需要设置 video pid
audio_pid	int	ps，ts 等需要设置 audio pid
audio_channels	int	Pcm 解码时需要设置
audio_samplerate	int	Pcm 解码时需要设置
vbuf_size	int	修改 video buffer 的大小，0 表示使用默认 size
abuf_size	int	修改 video buffer 的大小，0 表示使用默认 size
am_sysinfo	dec_sysinfo_t	系统解码辅助信息

- **buf_status**

【描述】

Buffer 状态结构体；

【定义】

```
struct buf_status
{
        int size;       /* buffer size*/
        int data_len; /* data in buffers*/
        int free_len; /*free buffer len*/
};
```

【参数】

成员变量	类型定义	含义
size	init	Buffer 大小
data_len	int	Buffer 内数据长度
Free_len	int	Buffer 空余长度

- **vdec_status**

【描述】

Video 状态结构体：

【定义】

```
struct vdec_status {
        unsigned int width;
        unsigned int height;
        float fps;
        unsigned int error_count;
        unsigned int status;
};
```

【参数】

成员变量	类型定义	含义
width	unsigned int	视频宽度（像素数）
height	unsigned int	视频高度（像素数）
fps	float	视频帧率
Error_count	unsigned int	视频解码错误数
status	unsinged int	视频解码状态

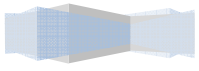
- **adec_status**

【描述】

Audio 状态结构体

【定义】

```
struct adec_status {
```

```
    unsigned int channels;
    unsigned int sample_rate;     /* audio sample rate*/
    unsigned int resolution;      /* bits per sample*/
    unsigned int error_count;
    unsigned int status;
};
```

【参数】

成员变量	类型定义	含义
channels	unsigned int	通道数
sample_rate	unsigned int	采样率
resolution	unsigned int	分辨率，即每个采样点位数
error_count	unsigned int	音频解码错误帧数
status	unsigned int	音频解码状态

2.4 函数接口

2.4.1 通用 codec 接口

- **codec_init**

【描述】

初始化解码控制器

【原型】

```
int    codec_init(codec_para_t *);
```

【参数】

参数名称	描述	输入/输出
codec_para_t *p	Codec 控制器结构体指针	输入

【返回值】

返回值	描述
CODEC_ERROR_NONE	初始化成功
-CODEC_ERROR_STREAM_TYPE_UNKNOW	初始化失败，未知流类型
-CODEC_ERROR_SET_BUFSIZE_FAILED	初始化失败，buffer 设置失败
-CODEC_ERROR_PARAMETER	初始化失败，参数错误
-CODEC_ERROR_INIT_FAILED	初始化失败，端口初始化失败
-CODEC_COMMON_ERROR	初始化失败，Codec 通用错误

其中，CODEC_COMMON_ERROR 包含：

（1）CODEC_ERROR_NONE

（2）CODEC_ERROR_BUSY
（3）CODEC_ERROR_NOMEM
（4）CODEC_ERROR_IO
（5）CODEC_ERROR_INVAL

【注意】
如果设备已经打开，则会初始化失败

- **codec_close**

【描述】
关闭 codec

【原型】
int codec_close(codec_para_t *);

【参数】

参数名称	描述	输入/输出
codec_para_t *p	Codec 控制器结构体指针	输入

【返回值】

返回值	描述
CODEC_ERROR_NONE	关闭成功

【注意】
无

- **codec_reset**

【描述】
codec 复位，即关闭并重新初始化

【原型】
int codec_reset(codec_para_t *p);

【参数】

参数名称	描述	输入/输出
codec_para_t *p	Codec 控制器结构体指针	输入

【返回值】

返回值	描述
CODEC_ERROR_NONE	复位成功
-CODEC_ERROR_STREAM_TYPE_UNKNOW	复位失败，未知流类型

-CODEC_ERROR_SET_BUFSIZE_FAILED	复位失败，buffer 设置失败
-CODEC_ERROR_PARAMETER	复位失败，参数错误
-CODEC_ERROR_INIT_FAILED	复位失败，端口初始化失败
-CODEC_COMMON_ERROR	复位失败，Codec 通用错误

【注意】

Codec 复位需要在设备无读写等操作的前提下进行，否则可能失败

- **codec_write**

【描述】

数据流写入 codec

【原型】

int codec_write(codec_para_t *pcodec,char *buffer,int len);

【参数】

参数名称	描述	输入/输出
codec_para_t *p	Codec 控制器结构体指针	输入
Char *buffer	数据流要写入的 buffer 指针	输入
Int len	要写入的数据流长度	输入

【返回值】

返回值	描述
-CODEC_COMMON_ERROR	写失败，Codec 通用错误

【注意】

无

- **codec_checkin_pts**

【描述】

写入该当前流对应的 pts

【原型】

int codec_checkin_pts(codec_para_t *pcodec, unsigned long pts);

【参数】

参数名称	描述	输入/输出
codec_para_t * pcodec	Codec 控制器结构体指针	输入
long pts	要写入的 pts	输入

【返回值】

返回值	描述
-1	Handle 无效
-CODEC_COMMON_ERROR	写入 pts 失败，Codec 通用错误

【注意】

无

- **codec_get_vbuf_state**

【描述】

获取 video buffer 的状态

【原型】

int codec_get_vbuf_state(codec_para_t *,struct buf_status *);

【参数】

参数名称	描述	输入/输出
codec_para_t *p	Codec 控制器结构体指针	输入
struct buf_status *buf	Buffer 状态	输出

【返回值】

返回值	描述
-1	Handle 无效
-CODEC_COMMON_ERROR	获取视频 Buffer 状态失败，Codec 通用错误

【注意】

无

- **codec_get_abuf_state**

【描述】

获取 audio buffer 的状态

【原型】

int codec_get_abuf_state(codec_para_t *,struct buf_status *);

【参数】

参数名称	描述	输入/输出
codec_para_t *p	Codec 控制器结构体指针	输入
struct buf_status *buf	Buffer 状态	输出

【返回值】

返回值	描述

-1	Handle 无效
-CODEC_COMMON_ERROR	获取音频 Buffer 状态失败，Codec 通用错误

【注意】

无

- **codec_get_vdec_state**

【描述】

获取 video codec 的状态

【原型】

int codec_get_vdec_state(codec_para_t *,struct vdec_status *);

【参数】

参数名称	描述	输入/输出
codec_para_t *p	Codec 控制器结构体指针	输入
struct vdec_status *sta	解码核状态	输出

【返回值】

返回值	描述
-1	Handle 无效
-CODEC_COMMON_ERROR	获取视频解码状态失败，Codec 通用错误

【注意】

只要解码核开始解码，读取到的状态信息才有意义

- **codec_get_adec_state**

【描述】

获取 audio codec 的状态

【原型】

int codec_get_adec_state(codec_para_t *,struct vdec_status *);

【参数】

参数名称	描述	输入/输出
codec_para_t *p	Codec 控制器结构体指针	输入
struct vdec_status *sta	解码核状态	输出

【返回值】

返回值	描述
-1	Handle 无效
-CODEC_COMMON_ERROR	获取音频解码状态失败，Codec 通用错误

【注意】

只要解码核开始解码，读取到的状态信息才有意义

- **codec_pause**

【描述】

暂停解码

【原型】

int codec_pause(codec_para_t *);

【参数】

参数名称	描述	输入/输出
codec_para_t *p	Codec 控制器结构体指针	输入

【返回值】

无

【注意】

无

- **codec_resume**

【描述】

恢复解码

【原型】

int codec_resume(codec_para_t *);

【参数】

参数名称	描述	输入/输出
codec_para_t *p	Codec 控制器结构体指针	输入

【返回值】

无

【注意】

无

2.4.2 音频 codec 接口

- **codec_set_mute**

【描述】

静音（mute=1）

【原型】

int codec_set_mute(codec_para_t*，int mute)

【参数】

参数名称	描述	输入/输出
codec_para_t *p	Codec 控制器结构体指针	输入
Int mute	静音标志，设为 1 时静音	输入

【返回值】

返回值	描述
0	成功
-1	创建 socket 失败
-2	连接 socket 失败

【注意】

无

- **codec_get_volume_range**

【描述】

获取音量的范围

【原型】

int codec_get_volume_range(codec_para_t *,int *min,int *max);

【参数】

参数名称	描述	输入/输出
codec_para_t *p	Codec 控制器结构体指针	输入
int *min	音量最小值指针	输出
Int *max	音量最大值指针	输出

【返回值】

返回值	描述
-CODEC_ERROR_IO	

【注意】

此函数未实现

- **codec_set_volume**

【描述】

设置音量

【原型】

int codec_set_volume(codec_para_t *,int val);

【参数】

参数名称	描述	输入/输出
codec_para_t *p	Codec 控制器结构体指针	输入
int val	需设置的音量值	输入

【返回值】

返回值	描述
0	成功
-1	创建 socket 失败
-2	连接 socket 失败

【注意】

无

- **codec_get_volume**

【描述】

获取音量

【原型】

int codec_get_volume(codec_para_t*)

【参数】

参数名称	描述	输入/输出
codec_para_t *p	Codec 控制器结构体指针	输入

【返回值】

返回值	描述
0	成功
-1	创建 socket 失败
-2	连接 socket 失败

【注意】

无

- **codec_set_volume_balance**

【描述】

设置左右声道平衡（0-100）

【原型】

int codec_set_volume_balance(codec_para_t *,int)

【参数】

参数名称	描述	输入/输出
codec_para_t *p	Codec 控制器结构体指针	输入
Int balance	需设置的平衡值	输入

【返回值】

返回值	描述
-CODEC_ERROR_IO	

【注意】

此函数未实现

- **codec_swap_left_right**

【描述】

左右声道互换

【原型】

int codec_swap_left_right(codec_para_t*)

【参数】

参数名称	描述	输入/输出
codec_para_t *p	Codec 控制器结构体指针	输入

【返回值】

返回值	描述
-CODEC_ERROR_IO	

【注意】

此函数未实现

3 Player

3.1 概述

Player 是 Linux 系统上基于 amcodec 和 ffmpeg 开发的播放器。具有播放、停止、暂停、恢复、拖动、快进、快退、音频切换等功能。Player 采用消息机制控制播放器的各种操作。

3.2 特征

3.2.1 支持文件格式

- ts
- mpg
- avi
- mkv
- mov
- mp4
- ps
- rmvb
- rm
- flv
- vob
- dat
- h264
- m2v
- mp3
- flac
- ac3
- aac
- pcm
- dts

3.2.2 支持的音频解码格式

- PCM_MULAW
- PCM_ALAW
- MP2
- MP3
- AAC

- AC3
- DTS
- PCM_S16BE
- PCM_S16LE
- PCM_U8
- COOK

3.2.3 支持的视频解码格式

- MJPEG
- jpeg
- XVID
- XVIX
- MP4
- COL1
- Mpeg1
- Mpeg2
- MP43
- M4S2
- DIV4
- DIVX
- DIV5
- DX50
- DIV6
- RMP4
- MP42
- MPG4
- MP4V
- AVC1
- H264
- RV30
- RV40

3.3 数据结构

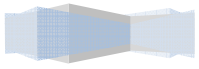
3.3.1 宏

- MESSAGE_MAX

  【定义】

  #define MESSAGE_MAX 4

  【描述】



Message 队列可容纳的最多未处理消息数目

- 错误码

（1） 前缀标记

- PLAYER 错误码前缀：

  #define P_PRE (0x02000000)

- FFMPEG 错误码前缀：

  #define F_PRE (0x03000000)

（2） PLAYER 错误码

- PLAYER 成功：

  #define PLAYER_SUCCESS (0)

- PLAYER 失败

  #define PLAYER_FAILED (-(P_PRE|0x1))

- PLAYER 内存不足

  #define PLAYER_NOMEM (-(P_PRE|0x2))

- PLAYER 空指针

  #define PLAYER_EMPTY_P (-(P_PRE|0x3))

- PLAYER 读数据失败

  #define PLAYER_RD_FAILED (-(P_PRE|0x4))

- PLAYER 读数据时遇到空指针

  #define PLAYER_RD_EMPTYP (-(P_PRE|0x5))

- PLAYER 读数据超时退出

  #define PLAYER_RD_TIMEOUT (-(P_PRE|0x6))

- PLAYER 写数据失败

  #define PLAYER_WR_FAILED (-(P_PRE|0x7))

- PLAYER 写数据时遇到空指针

  #define PLAYER_WR_EMPTYP (-(P_PRE|0x8))

- PLAYER 写数据完成

  #define PLAYER_WR_FINISH (P_PRE|0x1)

- PLAYER 写入 PTS 错误

  #define PLAYER_PTS_ERROR (-(P_PRE|0x9))

- PLAYER 找不到对应的 decoder
  #define PLAYER_NO_DECODER (-(P_PRE|0xa))

- Decoder 复位失败
  #define DECODER_RESET_FAILED (-(P_PRE|0xb))

- Decoder 初始化失败
  #define DECODER_INIT_FAILED (-(P_PRE|0xc))

- 不支持播放类型
  #define PLAYER_UNSUPPORT (-(P_PRE|0xd))

- seek 越位
  #define PLAYER_SEEK_OVERSPILL (-(P_PRE|0xe))

（3） FFMPEG 错误码

- FFMPEG 成功
  #define FFMPEG_SUCCESS (0)

- FFMPEG 打开文件失败
  #define FFMPEG_OPEN_FAILED (-(F_PRE|0x1))

- FFMPEG 解析文件失败
  #define FFMPEG_PARSE_FAILED (-(F_PRE|0x2))

- FFMPEG 遇到空指针
  #define FFMPEG_EMP_POINTER (-(F_PRE|0x3))

- 未指定文件
  #define FFMPEG_NO_FILE (-(F_PRE|0x4))

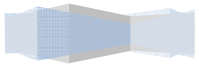
3.3.2 枚举

- **player_status**

【描述】
播放器状态

【定义】

```
typedef enum
{
    PLAYER_STOPED = 0,
    PLAYER_RUNNING,
```

```
    PLAYER_PAUSE,
    PLAYER_WAITING,
    PLAYER_SEARCHING,
    PLAYER_INITING,
    PLAYER_ERROR,
    PLAYER_PLAYEND,
    PLAYER_START,
}player_status;
```

【参数】

参数	含义
PLAYER_STOPED	播放器停止
PLAYER_RUNNING	播放器正在播放
PLAYER_PAUSE	播放器暂停
PLAYER_WAITING	播放器等待
PLAYER_SEARCHING	播放器正在拖动
PLAYER_INITING	播放器正在初始化
PLAYER_ERROR	播放器出错
PLAYER_PLAYEND	播放器播放结束
PLAYER_START	播放器开始播放

3.3.3 结构体

- **play_control_t**

【描述】

播放器控制参数

【定义】

```
typedef struct
{
    char    *file_name;
    //List    *play_list;
    int    video_index;
    int    audio_index;
    int    loop_mode;
    int    nosound;
    int    novideo;
}play_control_t;
```

【参数】

参数	类型	描述

file_name	char *	指定播放文件名
video_index	int	指定播放的视频索引（除 Real 外默认播放第一个视频，real 播放码率最大的视频）
audio_index	int	指定播放的音频索引（除 Real 外默认播放第一个音频，real 播放码率最大的音频）
loop_mode	int	设置循环播放模式
nosound	int	不播放声音
novideo	int	不播放视频

- **mvideo_info_t**

【描述】

视频媒体信息

【定义】

```
typedef struct
{
    int frame_rate;
    int video_width;
    int video_height;
    int video_format;
    int aspect_ratio;
    int64_t total_frame_num;
    int64_t first_dts;
    int64_t start_time;        //first pts
}mvideo_info_t;
```

【参数】

参数	类型	描述
frame_rate	Int	视频帧率
video_width	Int	视频帧宽
video_height	Int	视频帧高
video_format	Int	视频格式
aspect_ratio	int	视频显示比例
total_frame_num	int64_t	视频总帧数
first_dts	int64_t	视频第一个 PTS
start_time	int64_t	视频播放起始时间

- **maudio_info_t**

【描述】

音频媒体信息

【定义】

```
typedef struct
{
    int audio_format;
    int audio_channel;
    int audio_samplerate;
    int64_t total_frame_num;
    int64_t first_dts;
    int64_t start_time;
}maudio_info_t;
```

【参数】

参数	类型	描述
audio_format	int	音频格式
audio_channel	int	音频通道数
audio_samplerate	int	音频采样率
total_frame_num	int64_t	音频总帧数
first_dts	int64_t	音频第一个 pts
start_time	int64_t	音频播放起始时间

- **mstream_info_t**

【描述】

流媒体信息

【定义】

```
typedef struct
{
    int duration;
    int file_size;
    int bitrate;
}mstream_info_t;
```

【参数】

参数	类型	描述
duration	Int	流媒体播放总时间
file_size	Int	流媒体文件大小
bitrate	int	流媒体比特率

- **media_info_t**

【描述】

媒体信息

【定义】

```
typedef struct
{
    mstream_info_t stream_info;
    mvideo_info_t video_info;
    maudio_info_t audio_info;
}media_info_t;
```

【参数】

参数	类型	描述
stream_info	mstream_info_t	流媒体信息
video_info	mvideo_info_t	视频媒体信息
audio_info	maudio_info_t	音频媒体信息

- **player_state_t**

【描述】

播放器当前状态

【定义】

```
typedef struct player_state
{
    char *name;
    int status;          /*stop,pause    */
    int full_time;       /*Seconds */
    int current_time;    /*Seconds       */
    int video_error_cnt;
    int audio_error_cnt;
    int error_no;
}player_state_t;
```

【参数】

参数	类型	描述
name	char *	文件名
status	int	播放器状态
full_time	int	文件播放总时间
current_time	int	当前播放时间
video_error_cnt	int	视频解码错误帧数
audio_error_cnt	int	音频解码错误帧数
error_no	int	播放器错误码

- **player_cmd_t**

【描述】

播放器控制命令

【定义】

```
typedef struct
{
    union{
        struct{
        unsigned int exit:1;
        unsigned int play:1;
        unsigned int stop:1;
        unsigned int start:1;
        unsigned int next:1;
        unsigned int prev:1;
        unsigned int pause:1;
        unsigned int resume:1;
        unsigned int position_set:1;
        unsigned int get_status:1;
        unsigned int loop:1;
        unsigned int noloop:1;
        unsigned int noblack:1;
        unsigned int black_out:1;
        unsigned int fast_forward:1;
        unsigned int fast_backward:1;
        unsigned int aid:1;
    };
    unsigned int cmd;
    };
    union{
        char *filename;
        char *file_list;
        int    param;
        unsigned long data[4];
    };
}player_cmd_t;
```

【参数】

参数	类型	描述
exit	unsigned int：1	播放器退出标记
play	unsigned int：1	播放器播放标记
stop	unsigned int：1	播放器停止标记

start	unsigned int：1	播放器开始标记
next	unsigned int：1	播放下一个文件标记（未实现）
prev	unsigned int：1	播放前一个文件标记（未实现）
pause	unsigned int：1	播放器暂停播放标记
resume	unsigned int：1	播放器恢复播放标记
position_set	unsigned int：1	播放器拖动标记
get_status	unsigned int：1	获取播放器状态标记
loop	unsigned int：1	循环播放标记
noloop	unsigned int：1	取消循环播放标记
noblack	unsigned int：1	播放结束后不黑屏标记
black_out	unsigned int：1	播放结束后打开黑屏标记
fast_forward	unsigned int：1	播放器快进标记
fast_backward	unsigned int：1	播放器快退标记
aid	unsigned int：1	播放器切换音频通道标记
cmd	unsigned int	播放器控制命令
filename	char *	待播放文件名
file_list	char *	待播放文件列表（未实现）
param	int	拖动目标时间点（秒为单位，快进快退操作速度，音频切换目标 PID
data[4]	unsigned long	

3.4 函数接口

3.4.1 player 接口

- **player_init**

【描述】

创建播放器并初始化

【原型】

Int player_init();

【参数】

无

【返回值】

返回值	描述
0	初始化成功

【注意】

无

- **player_start**

【描述】

启动播放器

【原型】

Int player_start(play_control_t *para);

【参数】

参数名称	描述	输入/输出
play_control_t *para	播放器控制参数	输入

【返回值】

返回值	描述
PLAYER_SUCCESS	播放器启动成功
PLAYER_FAILED	播放器启动失败

【注意】

无

- **player_stop**

【描述】

播放器停止。

【原型】

Int player_stop();

【参数】

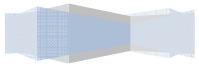
无

【返回值】

返回值	描述
PLAYER_SUCCESS	播放器停止消息发送成功
PLAYER_NOMEM	播放器分配消息空间失败
PLAYER_FAILED	播放器发送消息失败，停止播放无效

【注意】

操作无效时，播放器维持原来的播放状态。



- **player_pause**

【描述】

播放器暂停播放。

【原型】

int player_pause();

【参数】

无

【返回值】

返回值	描述
PLAYER_SUCCESS	播放器暂停消息发送成功
PLAYER_NOMEM	播放器分配消息空间失败
PLAYER_FAILED	播放器发送消息失败，暂停播放无效

【注意】

操作无效时，播放器维持原来的播放状态。

- **player_resume**

【描述】

播放器恢复播放。

【原型】

int player_resume();

【参数】

无

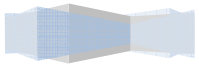
【返回值】

返回值	描述
PLAYER_SUCCESS	播放器恢复消息发送成功
PLAYER_NOMEM	播放器分配消息空间失败
PLAYER_FAILED	播放器发送消息失败，恢复播放无效

【注意】

操作无效时，播放器维持原来的播放状态。

- **player_timesearch**



【描述】

播放器拖动。

【原型】

int player_timesearch(int s_time);

【参数】

参数名称	描述	输入/输出
int s_time	播放器拖动目标时间点（秒）	输入

【返回值】

返回值	描述
PLAYER_SUCCESS	播放器拖动消息发送成功
PLAYER_NOMEM	播放器分配消息空间失败
PLAYER_FAILED	播放器发送消息失败，拖动无效

【注意】

操作无效时，播放器维持原来的播放状态。

- **player_forward**

【描述】

播放器快进。

【原型】

int player_forward(int speed);

【参数】

参数名称	描述	输入/输出
int s_speed	播放器快进速度	输入

【返回值】

返回值	描述
PLAYER_SUCCESS	播放器快进消息发送成功
PLAYER_NOMEM	播放器分配消息空间失败
PLAYER_FAILED	播放器发送消息失败，快进无效

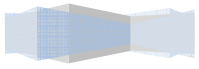
【注意】

操作无效时，播放器维持原来的播放状态。

- **player_backward**

【描述】

播放器快退



【原型】

Int player_backward(int speed);

【参数】

参数名称	描述	输入/输出
int s_speed	播放器快退速度	输入

【返回值】

返回值	描述
PLAYER_SUCCESS	播放器快退消息发送成功
PLAYER_NOMEM	播放器分配消息空间失败
PLAYER_FAILED	播放器发送消息失败，快退无效

【注意】

操作无效时，播放器维持原来的播放状态。

- **player_aid**

【描述】

播放器音频切换

【原型】

int player_aid(int audio_id);

【参数】

参数名称	描述	输入/输出
int audio_id	播放器快进速度	输入

【返回值】

返回值	描述
PLAYER_SUCCESS	播放器快退消息发送成功
PLAYER_NOMEM	播放器分配消息空间失败
PLAYER_FAILED	播放器发送消息失败，快退无效

【注意】

无

- **get_play_info**

【描述】

获取播放器状态

【原型】

player_state_t *get_play_info();

【参数】

无

【返回值】

返回值	描述
player_state_t *	播放器当前状态

【注意】

无

- **get_media_info**

【描述】

获取媒体信息

【定义】

media_info_t *get_media_info();

【参数】

无

【返回值】

返回值	描述
media_info_t *	媒体信息

【注意】

无

- **update_player_state**

【描述】

更新播放器状态。

【定义】

int update_player_state(player_state_t *s);

【参数】

参数名称	描述	输入/输出
player_state_t *s	播放器状态	输入

【返回值】

返回值	描述

PLAYER_SUCCESS	播放器状态更新成功

【注意】

调用此函数需要先注册 callback

- **register_update_callback**

【描述】

注册更新 callback 函数。

【定义】

int register_update_callback(update_state_fun_t up_fn,int interval_s);

【参数】

参数名称	描述	输入/输出
update_state_fun_t up_fn	更新 callback 函数	输入
int interval_s	调用 callback 的时间间隔	输入

【返回值】

返回值	描述
PLAYER_SUCCESS	播放器注册 callback 成功

【注意】

无

- **player_error_msg**

【描述】

播放器错误码与错误信息转换。

【定义】

char * player_error_msg(int error);

【参数】

参数名称	描述	输入/输出
int error	播放器错误码	输入

【返回值】

返回值	描述
char *	播放器错误信息

【注意】

无

- **set_player_error_no**

【描述】
设置播放器错误码。

【定义】
void set_player_error_no(int error_no);

【参数】

参数名称	描述	输入/输出
int error	播放器错误码	输入

【返回值】
无

【注意】
无

3.4.2 音频控制接口

- **audio_set_mute**

【描述】
音频静音设置。

【定义】
int audio_set_mute(int mute);

【参数】

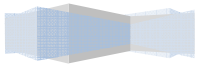
参数名称	描述	输入/输出
int mute	静音标志（0:unmute 1:mute）	输入

【返回值】

返回值	描述
0	音频静音设置成功
-1	音频静音设置失败

【注意】
无

- **audio_get_volume_range**



【描述】

获取音量范围。

【定义】

int audio_get_volume_range(int *min,int *max);

【参数】

参数名称	描述	输入/输出
int *min	音量最小值	输入
int *max	音量最大值	输入

【返回值】

返回值	描述
0	获取音量范围成功
-1	获取音量范围失败

【注意】

无

- **audio_set_volume**

【描述】

设置音量。

【定义】

int audio_set_volume(int val);

【参数】

参数名称	描述	输入/输出
int val	要设置的音量值	输入

【返回值】

返回值	描述
0	获取音量范围成功
-1	获取音量范围失败

【注意】

无

- **audio_get_volume**

【描述】

获取当前音量。

【定义】

int audio_get_volume();

【参数】

无。

【返回值】

返回值	描述
int	当前音量值

【注意】

无

- **audio_set_volume_balance**

【描述】

设置音量平衡。

【定义】

int audio_set_volume_balance(int balance);

【参数】

参数名称	描述	输入/输出
int balance	平衡标志（1：平衡）	输入

【返回值】

返回值	描述
-CODEC_ERROR_IO	

【注意】

此函数尚未实现

- **audio_swap_left_right**

【描述】

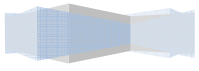
交换左右声道

【定义】

int audio_swap_left_right();

【参数】

无



【返回值】

返回值	描述
0	交换声道成功
-1	交换声道失败

【注意】

无

- **audio_left_mono**

【描述】

屏蔽右声道。

【定义】

int audio_left_mono();

【参数】

无

【返回值】

返回值	描述
0	屏蔽右声道成功
-1	屏蔽右声道失败

【注意】

无。

- **audio_right_mono**

【描述】

屏蔽左声道。

【定义】

int audio_right_mono();

【参数】

无。

【返回值】

返回值	描述
0	屏蔽左声道成功
-1	屏蔽左声道失败

【注意】
无。

- **audio_stereo**

【描述】
取消声道屏蔽。

【定义】
int audio_stereo();

【参数】
无。

【返回值】

返回值	描述
0	取消声道屏蔽成功
-1	取消声道屏蔽失败

【注意】
无。

3.4.3 消息接口

- **message_alloc**

【描述】
分配消息空间。

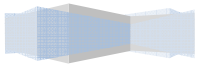
【定义】
player_cmd_t * message_alloc(void);

【参数】
无。

【返回值】

返回值	描述
player_cmd_t *	分配到的消息空间
NULL	未分配到内存

【注意】
无



- **message_free**

【描述】

释放消息空间。

【定义】

int message_free(player_cmd_t * cmd);

【参数】

参数名称	描述	输入/输出
player_cmd_t * cmd	消息空间指针	输入

【返回值】

返回值	描述
0	释放成功
-1	释放失败

【注意】

无

- **send_message**

【描述】

发送消息。

【定义】

int send_message(player_cmd_t *cmd);

【参数】

参数名称	描述	输入/输出
player_cmd_t * cmd	消息指针	输入

【返回值】

返回值	描述
0	发送消息成功
-1	发送消息失败

【注意】

无。

- **clear_all_message**

【描述】

清除所有消息

【定义】

void clear_all_message(void);

【参数】

无

【返回值】

无

【注意】

无

- **get_message**

【描述】

获取消息。

【定义】

player_cmd_t * get_message(void);

【参数】

无

【返回值】

返回值	描述
player_cmd_t *	获取到的消息
NULL	空消息

【注意】

无

3.5 播放器控制模式

3.5.1 Socket 控制接口

1）Socket 基本设置：

Log 文件路径："/tmp/amplayer.log"

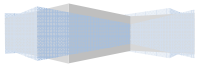
基本 socket 控制接口路径： "/tmp/player_socket"
消息反馈路径："/tmp/player_response"
若同时启动 2 个 player 独立播放，则第二组 socket 控制路径为：
socket 控制接口路径： "/tmp/player_socket_d"
消息反馈路径："/tmp/player_response_d"

2）Command 发送数据格式：

len	Command String

3）String 的控制命令格式

控制命令	命令格式	描述
play	play: filename	
stop	stop	
pause	Pause	暂停播放
quit	Exit	Amplayer 退出，退出后不能再接收命令
Resume	Resume	恢复播放
Search	Search:time	Time 以秒为单位，表示从文件头到跳转地的秒数；
Loop	loop	为循环播放模式
Mute	mute	退出静音
Unmute	unmute	解除静音模式
GetState	GetState	获取播放状态
Volume set	Volset:volume	设置当前音量
State	State:Current;fulltime;	返回播放状态
FastPlay	ff:num	快速播放模式（1，2x，4x，8x，16x）
slowPlay	fr:num	快退播放模式（1，2x，4x，8x，16x）
info	info	获取播放器状态，信息直接返回到命令 socket
media	media:aspect_ratio:%d:bit_rate:%d:width:%d:height:%d	获取媒体信息，信息直接返回到命令 socket



3.5.2 D-BUS 控制接口

关于 D-BUS 的详细介绍可以查看 http://www.freedesktop.org/wiki/Software/dbus

1）名称定义

Amplayer2 挂在 dbus-daemon 上的名字为：org.mpris.amplayer

2）对象路径(Object Path)

Amplayer2 提供 4 个对象路径，分别为：/ 、/Player 、/TrackrList 、/Amplayer

3）接口(Interface)

对象路径/ 、/Player 、/TrackrList 下的接口为：org.freedesktop.MediaPlayer，对象路径/Amplayer 下的接口为：org.aml.Amplayer

4）方法(Methods)

/ 、/Player 、/TrackrList 对象的方法和信号，我们符合 MPRIS 标准，MPRIS 的具体定义请参考：http://xmms2.org/wiki/MPRIS

下面详细介绍/Amplayer 对象的方法和信号。

/Amplayer objects methods

- **LeftMono**

  【描述】

  屏蔽右声道

  【参数】

  无

- **RightMono**

  【描述】

  屏蔽左声道

  【参数】

无

- **Stereo**

【描述】

取消声道屏蔽

【参数】

无

- **SwapChanl**

【描述】

交换左右声道

【参数】

无

- **Seek**

【描述】

跳跃到指点时间点

【参数】

int point ;

【说明】

时间点，单位为秒

- **FastForward**

【描述】

快进

【参数】

Int speed ;

【说明】

Speed 为快进速度；当 speed 为 0 时，取消快进。

- **FastBackward**

【描述】

快退

【参数】

Int speed ;

【说明】

Speed 为快退速度；当 speed 为 0 时，取消快退。

- **Duration**

【描述】

或得播放文件的总时长

【参数】

int * duration;

【说明】

取得的总时长存放在参数 duration 中。

- **Time**

【描述】

获得当前时间点

【参数】

int * time;

【说明】

取得的当前时间点存放在参数 time 中，单位为秒。

- **Mute**

【描述】

静音

【参数】

boolean on ;

【说明】

当 on 为 TRUE 时，进入静音模式；当 on 为 FALSE 时，退出静音模式。

- **Loop**

【描述】

循环播放

【参数】

boolean on ;

【说明】

当 on 为 TRUE 时，进入循环播放模式；当 on 为 FALSE 时，退出循环播放模式。

- **VideoInfo**

【描述】

获取视频信息

【参数】

{int frame_rate, int video_width, int video_height, int video_format}

【说明】

取得的视频信息分别存放在 4 个参数中

- **AudioInfo**

【描述】

获取音频信息

【参数】

{int audio_format, int audio_channels, int audio_samplerate}

【说明】

取得的音频信息分别存放在 4 个参数中

/Amplayer objects signals

- **PlayEnd**

【描述】

发送 play end 或 play start 信号

【参数】

char * message

【说明】

播放开始时，发送 message 为“play start”的信号；播放结束时，发送 message 为“play end”的信号。

- **PlayErroe**

【描述】

发送播放出错的信号

【参数】

int errno

【说明】

播放发生错误时，会发送该信号，信号中包含了错误号。

4 编写外接输入流库

4.1 头文件

```
#include "libavutil/avstring.h"
#include "libavformat/avformat.h"
#include <fcntl.h>
#if HAVE_SETMODE
#include <io.h>
#endif
#include <unistd.h>
#include <sys/time.h>
#include <stdlib.h>
... …
```

4.2 输入库的结构体

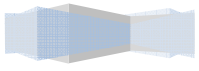
当播放流的路径名以 “sss”开头时，
使用该结构体对应的 open、read 等函数来操作流；
具体可以参考
Packages/cntv-stream/
Packages/ffmpeg/src/libavformat/http.c*/

```
URLProtocol sss_protocol = {
        "sss",
        file_open,
        file_read,
        file_write,
        file_seek,
        file_close,
        .url_get_file_handle = file_get_handle,
};
```

4.3 库的注册

```
void __attribute__ ((constructor)) cnfile_init(void);
/*DLOpen 时自动运行该函数，把该库注册到 FFMPEG 库里*/
void cnfile_init(void )
{
        printf("CNTV:register bxstream_protocol\n\n");
```

```
        av_register_protocol(&bxstream_protocol);
}
void __attribute__ ((destructor)) cnfile_exit(void);
void cnfile_exit(void)
{
    printf("CNTV:exit bxstream_protocol\n\n");
}
```

5 Amplayer 的外接控制器接口

具体可以参考 pakages/amplayer2/src/controller/shell_control.c
使用 void __attribute__ ((constructor)) cdbus_init(void);来注册接口；

```
typedef struct player_controler
{
        char name[16];
        int front_mode;
        int (*init)(struct control_para *);
        int (*get_command)(player_cmd_t *);
        int (*update_state)(struct control_para *,player_state_t*);
        int (*release)(struct control_para *);
}player_controler_t;

int register_controler(player_controler_t *controler)
```