

# 移动端播放库文档

## 1 APlayer: Android 端接口说明

### 1.1 接口说明

#### 1.1.1 APlayerAndroid()

APlayerAndroid 构造函数。APlayerAndroid 可同时存在多个实例，各个实例之间互不冲突。

#### 1.1.2 void destroy()

彻底释放播放器资源，该函数一般最后调用。

每一个 APlayerAndroid java 对象在底层都有一个对应的 APlayerAndroid C++对象，Java 的垃圾回收机制并不能释放 C++，为了释放该 C++对象必须调用 destroy，否则会出现内存泄漏。

调用 destroy 时如果当前播放的媒体未关闭，该函数会先去关闭媒体，然后在关闭消息中释放 C++资源。（所以使用时可以不用先关闭媒体，而直接调用 destroy）

和 Close() 区别：Destroy() 会释放内部资源，Close() 不会释放仍会持有资源，Close() 后再次打开播放器更快([1.1.9 int close\(\)](#))。

#### 1.1.3 void useSystemPlayer(boolean use)

不再使用

#### 1.1.4 boolean isSystemPlayer()

不再使用

#### 1.1.5 int setView(SurfaceView surfaceView)

设置渲染表面，就是显示视频图像的组件，如果这个函数没有调用，将无法显示图像。对于软解，可在任何时候设置渲染表面，但是对于硬解和使用系统播放，必须在 Open 之前设置渲染表面，否则无法打开成功。软解和 level 20 以上的系统硬解时画面按照指定的模式来显示（在渲染模式中设置中详叙），level 19 以下的系统硬解时，画面只能以铺满的方式显示，如果你需要保持正确的纵横比，只能根据视频的宽高来设置 surfaceview 的大小。level 19 以下的系统硬解时，surfaceview 销毁会导致视频硬件解码器停止，surfaceview 重新创建时，aplayerandroid 内部会

重建视频硬件解码器，并且 seek 到上次播放的位置。软解和 level 20 以上的系统硬解，不会有这个问题。

#### 1.1.6 int setView(Surface surface)

同 int SetView(SurfaceView surfaceView);

#### 1.1.7 int setView(TextureView textureView)

同 int SetView(SurfaceView surfaceView);

#### 1.1.8 int open(String)

打开一个视频文件，参数是视频文件路径，可以是本地文件，也可以是支持的网络协议媒体流。该函数立即返回，调用成功返回 0（但并不表示视频文件打开成功，只是函数调用成功（可通过注册 [1.2.1 文件打开成功事件](#) 或 [1.2.3 打开调用完毕事件](#) 进行处理））。

注意：如果是替换当前正在播放的文件，需要先调用 close()，并等待播放完成事件 ([1.2.4 播放完成事件](#))，播放完成事件发生后，与当前媒体文件（流）的相关操作失效，需要再次调用 open（）。

strUrl

[输入参数] 一个表征媒体文件地址的字符串，该字符串可以是本地、局域网共享或网络文件，例如：

/test.rmvb

\\192.168.8.188\share\test.rmvb

<http://218.221.12.181/test.rmvb>

<ftp://218.221.12.181/test.rmvb>

<rtmp://218.221.12.181/test.flv>

<rtsp://218.221.12.181/test.sdp>

返回值：

调用成功返回 0

#### 1.1.9 int close()

关闭播放器, 成功返回 0。该函数是非阻塞函数，真正的关闭动作在另外的线程异步执行，关闭成功后，OnPlayCompleteListener 事件回掉。

#### 1.1.10 int play()

开始播放视频。视频打开成功后不会立刻播放，而是转为暂停状态，如果需要播放视频，需要调用该函数，因此该函数一般是在 Open 成功事件中调用，或者从暂停状态恢复播放状态时调用。该函数立即返回，返回值为-1 表示失败，返回值为 0 表示成功。

#### 1.1.11 int pause()

暂停视频播放。暂停音视频解码和渲染，数据读取过程继续直至填满内部缓冲区，成功返回 0。

#### 1.1.12 String getVersion()

返回当前 jar 包版本字符串，形如："1.2.0.100"

#### 1.1.13 int getState()

得到播放器目前的状态。播放器有 7 个如下状态。

```
public class PlayerState
{
    public static final int APLAYER_READ    = 0;
    public static final int APLAYER_OPENING = 1;
```

```

public static final int APLAYER_PAUSING    = 2;

public static final int APLAYER_PAUSED     = 3;

public static final int APLAYER_PLAYING    = 4;

public static final int APLAYER_PLAY      = 5;

public static final int APLAYER_CLOSEING   = 6;

}

```

7 个状态中 *APLAYER\_READ*, *APLAYER\_PAUSED* *APLAYER\_PLAYING* 是稳态 *APLAYER\_OPENING*、*APLAYER\_PAUSING*、*APLAYER\_PLAY*、*APLAYER\_CLOSEING* 是瞬时状态，各个状态之间可以互相转换，状态转换图如下

有多种情况会使播放器进入 PS\_READY 状态，进入原因可通过 GetConfig 方法查询 CONFIGID.*PLAYRESULT* 信息（见 [1.1.16 String getConfig\(int configID\)](#)）。

#### 1.1.14 int getDuration()

得到视频的 duration, 单位为毫秒。视频打开成功后调用该函数获得视频的 duration, 但视频文件可能不包含该字段信息，这时得到的 duration 为-1，内部可以通过读取一定数量的包之后来估算视频的 duration, 播放一段时间可获取到 duration 估计值（等待时间越久相对越精确）。

#### 1.1.15 int setConfig(int configID, String value)

详见 [1.16 String getConfig\(int configID\)](#)。

#### 1.1.16 String getConfig(int configID)

设置和获取 APlayer 的参数，APlayer 中将所有非基本的功能集中在该函数中，以减少接口数量。configID 是属性 ID，所有的属性 ID 都定义在类 CONFIGID 中，value 和 GetConfig 的返回值是属性值，对于不同的属性，value 是字符串形式，但不同参数格式不同。属性配置如下：

```

public class CONFIGID
{

    public static final int CLEAR_CONFIG = 1;
    public static final int STREAM_TYPE = 6;

```

```
public static final int PLAYRESULT = 7;
public static final int AUTO_PLAY = 8;
public static final int HARDWARE_RENDER_TYPE = 9;
public static final int EXTIO = 14;
public static final int READPOSITION = 31;
public static final int UPDATEWINDOW = 40;
public static final int ORIENTATION_ANGLE = 41;
public static final int QUICKLY_MOV_ACTION_COUNT = 42;
public static final int PLAY_SPEED = 104;
public static final int DOWN_SPEED = 105;
public static final int STRETCH_MODE = 202;
public static final int ASPECT_RATIO_NATIVE = 203;
public static final int ASPECT_RATIO_CUSTOM = 204;
public static final int LIVE_MODE = 205;
public static final int ONLY_AUDIO = 206;
public static final int AUTO_COMPLETE_PLAY = 207;

public static final int HW_DECODER_USE = 209;
public static final int HW_DECODER_ENABLE = 230;
public static final int HW_DECODER_DETEC = 231;

public static final int AUDIO_TRACK_LIST = 402;
public static final int AUDIO_TRACK_CURRENT = 403;
public static final int AUDIO_SILENCE = 420;

public static final int SUBTITLE_USABLE = 501;
public static final int SUBTITLE_EXT_NAME = 502;
public static final int SUBTITLE_FILE_NAME = 503;
//public static final int SUBTITLE_SHOW = 504;
public static final int SUBTITLE_LANGLIST = 505;
public static final int SUBTITLE_CURLANG = 506;
public static final int SUBTITLE_SHOW_EXTERNAL = 507;
public static final int SUBTITLE_CORRECTION = 509;

public static final int HTTP_COOKIE = 1105;
public static final int HTTP_REFERER = 1106;
public static final int HTTP_CUSTOM_HEADERS = 1107;
public static final int HTTP_USER_AGENT = 1108;

public static final int HTTP_USER_AHTTP = 1109;
public static final int HTTP_AHTTP_CACHE_DIR = 1110;
public static final int HTTP_AHTTP_USE_CACHE = 1111;
public static final int HTTP_AHTTP_CACHE_PATH = 1112;
public static final int HTTP_AHTTP_DELETE_CACHE = 1113;
public static final int HTTP_AHTTP_ONLY_CACHE_HEAD = 1114;
public static final int HTTP_AHTTP_CACHE_HEAD_LENGTH = 1114;
```

```

public static final int HTTP_USER_AHTTP2 = 1115;

public static final int NET_BUFFER_ENTER = 1001;
public static final int NET_BUFFER_LEAVE = 1002;
public static final int NET_BUFFER_READ = 1003;
public static final int NET_BUFFER_READ_TIME = 1005;
public static final int NET_SEEKBUFFER_WAITTIME = 1004;
public static final int NET_BUFFER_LEAVE_INCR = 1006;

public static final int VR_ENABLE = 2401;
public static final int VR_ROTATE = 2411;
public static final int VR_FOVY = 2412;
public static final int VR_MODEL = 2413;
public static final int VR_ENABLE INNER TOUCH ROTATE = 2414;
public static final int VR_DISTORTION_CORRECTION = 2415;

public static final int SEEK_ENABLE = 3000;

public static final int RECORD_BIT = 4000;
public static final int RECORD_WIDTH = 4001;
public static final int RECORD_HEIGHT = 4002;
public static final int RECORD_MODE = 4003;

public static final int FRAMERATE_CURRENT = 5001;
public static final int FRAMERATE_NUM = 5002;
public static final int FRAMERATE_INTERNAL = 5003;

public static final int FIND_STREAM_INFO = 6001;
public static final int FF_CONFIG_BUFFERS = 6002;

}

```

### ***CLEAR\_CONFIG:***

支持函数: getConfig / setConfig

重新 open 时是否清除上次播放设置的 config 值, “1” 清除, “0” 保留。默认值为 “1”

### ***STREAM\_TYPE:***

支持函数: getConfig

当前播放的流类型, 转换成二进制后 第一位为 1 表示有音频, 第二位为 1 表示有视频。

### ***PLAYRESULT:***

支持函数: getConfig

返回播放结束的结果。有三种原因导致播放结束。

返回 “0” —— 视频播放完成。

返回 “1” —— 用户主动结束播放。

其他字符串（非 0 非 1） —— 播放出错结束，返回错误代码。

### ***AUTO\_PLAY:***

支持函数: getConfig/setConfig

文件打开成功后是否自动播放，默认不播放

“0” —— 打开成功后不自动播放（需要调用 Play（）播放）。

“1” —— 打开成功后自动播放。

### ***HARDWARE\_RENDER\_TYPE***

支持函数: getConfig / setConfig

该配置指定创建硬件解码器时所关联的 surface 来自哪里（所以只有在硬解时才有效），“0” 直接使用传入的 surfaceView 或者 textureView 的 surface，该模式下，硬件解码器解码后会直接渲染到该 surface 中，外部无法控制渲染也无法利用解码后到图像，因此在该模式下，不能指定渲染模式，也无法录制，另外 surfaceview 销毁时需要停止硬件解码器。“2” 使用手动创建的一个跟传入的渲染表面相关的 surface，该方式下可以外部控制渲染，也可以得到硬解之后的纹理再次加工，另外也可以将外部的 surfaceview 和硬件解码器分离，但是仅仅在 level 20 以上的系统中才能使用这个模式。默认情况下，我们并不设置该配置值，level20 以上的系统会自动使用第二种模式，19 以下的系统使用第一种方式。

### ***READPOSITION:***

支持函数: getConfig

返回当前视频缓冲位置，单位是毫秒，注意是字符串形式。该缓冲位置只是播放内核的缓冲位置，表示播放内核已经读取到了该位置的音视频数据包，并不表示网络播放中已经读到本地的视频数据位置。

#### *UPDATEWINDOW:*

支持函数：setConfig

不再使用

#### *ORIENTATION\_ANGLE:*

支持函数：setConfig

不再使用

#### *QUICKLY\_MOV\_ACTION\_COUNT:*

未启用。跟 mp4 的加速打开有关

#### *PLAY\_SPEED:*

支持函数：getConfig/setConfig

获取/设置当前播放速率，为正常速率的百分之多少，以 100 为单位，即 100 为正常速度，50 为半速，200 为两倍速。变速的范围是 50 -200, 如果超过这个范围内部会规范到该范围中

#### *DOWN\_SPEED:*

支持函数：getConfig

获取当前的下载速度，只有在设置 HTTP\_USER\_AHTTP2 配置项时才有值，否则为 0.

#### *STRETCH\_MODE:*

支持函数：getConfig / setConfig



获取/设置视频画面的拉伸模式，

“0” 默认模式，在保持纵横比的情况下，如果视频的宽高比大于窗口的宽高比则视频宽顶边，否则高顶边。

“1” 裁剪模式，视频宽高比大于窗口宽高比的情况，视频的高顶边，宽裁剪两边，以保持纵横比。反之，宽顶边，裁剪高度。

“2” 拉伸模式，视频铺满窗口并且不裁剪，如果窗口宽高比不等于视频的宽高比 视频会出现拉伸现象。

### ***ASPECT\_RATIO\_NATIVE:***

支持函数：getConfig

获取视频的自然宽高比，格式：“4;3”。

### ***ASPECT\_RATIO\_CUSTOM:***

支持函数：getConfig/setConfig

视频的自定义宽高比，格式：“4;3”。

例：setConfig(CONFIGID.ASPECT\_RATIO\_CUSTOM, " 4;3" );

纵横比设置在拉伸模式下无效，在 level 19 以下硬解时无效。

### ***LIVE\_MODE:***

支持函数：getConfig/setConfig

设置直播模式（对直播视频，可加快首帧打开, 并且不会进入缓冲，未做追帧操作），open() 前调用有效。

例：SetConfig(CONFIGID.LIVE\_MODE, " 1" );

### ***ONLY\_AUDIO:***

支持函数：getConfig/setConfig

设置是否只播放音频，默认为“0”，如果设置为“1”，视频将不会被解码。

#### ***AUTO\_COMPLETE\_PLAY:***

支持函数：getConfig/setConfig

视频播放完成之后是否关闭该视频，

“0” 不关闭视频，

“1” 关闭视频

默认值是 “1”

#### ***HW\_DECODER\_USE:***

支持函数：getConfig/setConfig

获取/设置硬件解码状态，默认关闭。

” 0” —— 关闭。

” 1” —— 开启。

需要在 Open 之前设置该参数，Open 之后设置对当前播放的媒体无效。如果媒体不支持硬解 设置该参数为” 1”，也只能是软解。。

#### ***HW\_DECODER\_ENABLE:***

支持函数：getConfig

获取系统是否支持当前媒体硬件解码

” 0” —— 不支持。

” 1” —— 支持。

#### ***HW\_DECODER\_DETEC:***

支持函数：getConfig/setConfig

是否开启硬件解码侦测，侦测是否支持当前媒体硬件解码，默认开启侦测，如不需要硬件解码，可在 `Open()` 前设置关闭。

” 0” —— 不开启。

” 1” —— 开启。

如果确定不需要硬解，可以设置为 “0”，毕竟侦测媒体是否支持硬解需要执行一些代码

### ***AUDIO\_TRACK\_LIST:***

支持函数：GetConfig

获取音轨列表。音轨列表，格式：“语言, 音轨 1;语言, 音轨 2”。

### ***AUDIO\_TRACK\_CURRENT:***

支持函数：GetConfig/SetConfig

获取或设置当前音轨索引，索引值从” 0” 开始。

### ***AUDIO\_SILENCE:***

支持函数：GetConfig/SetConfig

获取和设置是否静音

“0” —— 不静音

“1” —— 静音

默认值是 “0”

### ***SUBTITLE\_USABLE:***

支持函数：getConfig

获取字幕加载功能是否可用。

“0” —— 不可用字幕加载

“1” —— 可用字幕加载

### ***SUBTITLE\_EXT\_NAME:*** （待实现）

支持函数：getConfig

支持的字幕格式列表，例如：“srt:ssa;ass;idx”。

### ***SUBTITLE\_FILE\_NAME:***

支持函数：getConfig/setConfig

获取/设置外挂字幕的文件名，例如：“/subtitle.srt”。

### ***SUBTITLE\_LANGLIST:***

支持函数：getConfig

获取当前加载的字幕的可用语言列表，用“;”分割，例如：“chinese;english”。

如果加载了外挂字幕，外挂字幕也在字幕列表中，位于内嵌字幕后面

### ***SUBTITLE\_CURLANG:***

支持函数：getConfig/setConfig

获取/设置 当前选择的字幕语言索引，例 ” 0” , 没有选择是返回无效的负值。

### ***HTTP\_COOKIE:***

支持函数：getConfig/setConfig

获取/设置 HTTP 请求头中带的 Cookie 字符串，默认为无。

### ***HTTP\_REFERER:***

支持函数：getConfig/setConfig

获取/设置 HTTP 请求头中带的 Referer 字符串，默认为无。

### *HTTP\_CUSTOM\_HEADERS:*

支持函数: getConfig/setConfig

获取/设置 HTTP 自定义头字段列表，每个头字段之间用回车换行符号 “\r\n”(即 0x0d, 0x0a) 分割。

### *HTTP\_USER\_AGENT*

支持函数: getConfig/setConfig

获取/设置 HTTP 请求头中带的 User Agent 字符串，默认为 APlayer 的默认值。

### *HTTP\_USER\_AHTTP:*

支持函数: getConfig/setConfig

获取/设置 是否使用 AHTTP。AHTTP 是用 android 组件 HttpURLConnection 实现的一个 httpclient, 设置使用 ahttp 后播放内核将从该 httpclient 获取数据，只有在播放 http 地址的时候才能设置该配置。设置 ahttp 后 如果打开媒体失败，还是会用 ffmpeg 原始的 httpclient 打开一次。

“1” —— 设置使用 ahttp

“0” —— 不使用 ahttp

默认设置为 “0”

### *HTTP\_AHTTP\_USE\_CACHE:*

支持函数: getConfig/setConfig

获取/设置 ahttp 是否使用缓存。如果使用缓存，ahttp 读取数据时先查看要读取的数据是否在缓存中，如果在缓存中，直接从缓存中读取，如果不在从网络中读取并且保存到本地文件中。缓存文件以碎片化形式保存，每一个缓存文件在本地对应两个文件，一个是数据文件后缀是.data，一个是记录碎片化结构的记录文件后缀是.rec。两个文件同名，文件名默认是播放路径的 md5 值，也可以由用户设置文件名。

“1” ——设置使用缓存

“0” ——不使用缓存

默认设置为 “0”

#### ***HTTP\_AHTTP\_CACHE\_DIR:***

支持函数：getConfig/setConfig

获取/设置 ahttp 缓存文件所在的目录。ahttp 不负责管理产生的缓存文件，默认情况下缓存文件目录是根目录 ahttp 文件夹中（ahttp 负责创建该文件夹）。为了更好的管理 ahttp 缓存文件，上层应用程序应该设置缓存文件夹。

#### ***HTTP\_AHTTP\_CACHE\_PATH:***

支持函数：getConfig/setConfig

获取/设置 缓存文件路径。缓存文件名默认是播放路径的 md5 值，应用程序也可以用该配置来设置缓存文件名，以便应对不断变化的播放路径。

#### ***HTTP\_AHTTP\_DELETE\_CACHE:***

支持函数：getConfig/setConfig

获取/设置 播放完成后是否删除缓存文件。

“1” —— 播放完成后删除缓存文件

“0” ——播放完成后不删除缓存文件

默认值为 “1”

#### ***HTTP\_AHTTP\_ONLY\_CACHE\_HEAD:***

支持函数：getConfig/setConfig

获取/设置是否只缓存视频文件头到缓存文件中

“1” ——只缓存文件头

“0” ——缓存所有读到的数据

默认值为 “0”

### ***HTTP\_AHTTP\_CACHE\_HEAD\_LENGTH:***

支持函数: getConfig/setConfig

获取/设置 视频头文件所包含的视频长度, 默认为 5 秒

### ***HTTP\_USER\_AHTTP2:***

支持函数: getConfig/setConfig

获取/设置是否使用 ahttp2, ahttp2 同 ahttp 一样 也是用 android 系统实现的一个 httpclient, 实现上与 ahttp 有一些区别,

它包含一个较大的内存缓存区。HTTP\_AHTTP\_CACHE\_DIR 到 HTTP\_AHTTP\_CACHE\_HEAD\_LENGTH 之间的配置项在 ahttp2 上同样生效。

“1” ——设置使用 ahttp2

“0” ——不使用 ahttp2

默认值为 “0”

### ***NET\_BUFFER\_ENTER:***

支持函数: getConfig/setConfig

获取/设置 当网络没有读取到数据时, 等待多久进入缓冲状态, 单位毫秒, 注意这里的时间是字符串。默认值是 200 毫秒。

备注: 缓冲状态一般会暂停播放持续一段时间, 以填满缓冲队列。

### ***NET\_BUFFER\_LEAVE:***

支持函数: getConfig/setConfig

获取/设置 在缓冲状态下，缓冲多少个帧退出缓冲，默认为 “150”。该值越大意味着进入缓冲后需要更久才能退出缓冲，但是后续的播放会更加流畅。

该值设置多少，并不意味着缓冲多少帧就退出缓冲，设置的值只是一个基础量，实际需要缓冲的帧是动态变化的，随着正常缓冲次数的增加而增加，最大可以到基础量的 4 倍。

如果是因为拖动而进入缓冲，所需缓冲的包数减半。另外我们设置的虽然是包的数目，但是内部计算缓存进度和判断是否结束缓存实际是用的时间，一个包按照 40ms 来计算。

#### ***NET\_BUFFER\_LEAVE\_INCR:***

支持函数：getConfig/setConfig

获取/设置 每次正常缓冲后，所需缓冲包数的增量。默认值为 “100”。比如基础缓冲包是 150，发生了四次正常缓冲，那么此时所需的缓冲包数量是  $150 + 4 * 100 = 550$

#### ***NET\_BUFFER\_READ:***

支持函数：getConfig/setConfig

获取/设置 内存队列中视频包的长度，默认值是 2500。队列越长，占用内存越多，网络播放时可能更流畅

#### ***NET\_BUFFER\_READ\_TIME:***

支持函数：getConfig/setConfig

获取/设置 内存队列中视频包的总时长，单位秒。默认是 “0”。也可以用时长表示视频队列的长度，这样更直观，时长乘以帧率就是队列长度。

如果有设置这个值，将会优先使用该值来计算队列长度，该值为 0 忽略该值。

#### ***NET\_SEEKBUFFER\_WAITTIME:***

支持函数：GetConfig/SetConfig

获取/设置 拖动播放进度后，没有数据时多久进入缓冲（缓冲一直到队列满，较耗时），单位毫秒，默认值是 2000。拖动时将会清空队列中的数据，如果队列中没有数据后

按照常规方式处理，大部分拖动，即使在速度很快的情况也会进入缓冲。



*VR\_ENABLE:*

不再支持

*VR\_ROTATE:*

不再支持

*VR\_FOVY:*

不再支持

*VR\_MODEL:*

不再支持

*VR\_ENABLE\_INNER\_TOUCH\_ROTATE:*

不再支持

#### 1.1.17 int setPosition (int msec)

设置当前播放进度，单位为毫秒，成功返回 0。

#### 1.1.18 int setPosition (int msec, boolean precise)

设置当前播放进度，单位为毫秒，参数 `precise` 用于指定是否做精确拖动。

#### 1.1.19 int getPosition()

得到当前播放进度，单位为毫秒。

### 1.1.20 int getWidth()

返回获取到的视频宽度，单位像素(pixel)，如不包含视频流，则返回0。打开成功后才能正确得到视频的宽度，如果视频是 90 度旋转的，该函数得到的值是经过调整的

### 1.1.21 int getHeight()

返回获取到的视频高度，单位像素(pixel)，如不包含视频流，则返回0。打开成功后才能正确得到视频的高度，如果视频是 90 度旋转的，该函数得到的值是经过调整的

### 1.1.22 int getBufferProgress()

获取当前缓冲进度 [0, 100], 不在缓冲和返回负值，缓冲进度也可以在缓冲消息中获取。进度 0 表示开始缓冲，100 表示结束缓冲。

### 1.1.23 int MediaInfo static parseThumbnail(String mediaPath, long timeMs, int width, int height, int luma, int maxSideSize)

获取媒体信息。这是一个静态函数，可以直接调用，跟当前的播放没有任何关系。

mediaPath: 媒体路径

timeMs: 指定时间点的截图（单位毫秒）

width: 生成的截图宽度（如 <=0，则使用视频的宽）

height: 生成的截图高度（如 <=0，则使用视频的高）

luma 获取画面的最低亮度返回 0-255

maxSideSize 获取画面的最大尺寸。

返回值：成功返回 MediaInfo, 失败返回 NULL

注意，对于部分直播数据不支持 Seek 操作的，建议设置为 0（timeMs 过大会导致调用耗时，timeMs 应该控制在 2000 ms 内）；

```
public static class MediaInfo
{
```

```

public static class BitMapInfo {
    public int width;    //获取到的位图的宽
    public int height;   //获取到的位图的高
    public int mSec;     //获取到的位图在视频中的精确时间
    public int luma;     //获取到的位图的亮度
    public int maxSideSize = -1;
}

public int width;           //视频的高
public int height;          //视频的宽
public long duration_ms;    //视频时长
public long file_size;      //视频文件长度
public long show_ms;        //不再使用
public boolean is_key_frame; //获取到的位图是否为
public Bitmap bitMap;        //获取到的缩略图 课呢给你为空
public BitMapInfo bitmapInfo = null; //获取到的缩略图的信息
public String angle;         // 视频的角度
}

```

其他重载函数

```
public static MediaInfo parseThumbnail(String mediaPath)
```

如果只想获取视频的相关信息 而不需要缩略图，调用该函数。

```
public MediaInfo parseThumbnail( long timeMs, int width, int height)
```

该函数是获取当前正在播放的视频的缩略图，视频播放期间，多次调用该函数不需要重复打开视频，增加获取位图的速度。

### 1.1.24 boolean isSupportRecord ()

判断当前播放视频是否支持录制功能，当前实现总是返回 true

### 1.1.25 boolean startRecord(String outMediaPath)

开始视频录制截取，请确保当前视频没有处于录制状态，录制的视频默认格式为原视频格式，和输入的文件名无关，文件路径需要保证目录存在，内部不创建目录。该函数在 open() 成功后有效。

注意：视频录制中不能手动改变视频播放进度，不能改变播放速度。

返回值：成功返回 true，失败返回 false。

重载函数：

```
public boolean startRecord(String outMediaPath, int len)
```

len : 指定录取长度。录制了指定长度的视频后自动结束。

```
public boolean startRecord(String outMediaPath, int start, int end)
```

start：视频录制的开始位置

end： 视频录制的结束位置

视频录制到结束位置后自动结束录制。

### 1.1.26 boolean isRecording()

获取当前媒体，是否处于录取状态中。

当视频处于录制中，返回 true。

### 1.1.27 void endRecord()

停止录制。视频录制如果不自动停止，那么一定需要调用该函数来结束录制，否则不会生成录制文件。该函数是阻塞函数，不会立刻返回。

### 1.1.28 void stopRead(boolean stopRead)

指定是否停止读取包，该函数不会暂停播放，只是会暂停读取数据。暂停读数据后，队列中的包播完之后，会进入到缓冲状态。

### 1.1.29 Bitmap getCurrentScreenshot()

获取当前显示帧的位图。

### 1.1.30 static long deleteCacheData(String cachePath)

删除缓存文件中的数据部分，保存媒体头数据。

#### 1.1.31 static long getCacheDataLength(String cachePath)

获取缓存文件中数据的长度。

#### 1.1.32 long[] getTimeFromFileOffset(long[] fileOffset)

将文件偏移转换成时间偏移值。

#### 1.1.33 StatisticsInfo getStatisticsInfo()

获取整个播放过程的统计信息

```
public static class StatisticsInfo {  
  
    public int playTime; //播放时间  
    public int normalBufferCount; //播放时缓冲次数  
    public int normalBufferTime; //播放时缓冲总时间  
    public int seekBufferCount; //拖动引起的缓冲次数  
    public int seekBufferTime; //拖动引起的缓冲总时间  
    public int seekCount; //拖动次数  
    public int seekAverageDrawingTime; //拖动平均出图时间  
    public int firstRenderTime; //首帧渲染时间  
    public int openSuccessTime; //打开成功时间  
    public long fileSize; //文件大小  
    public int duration; //视频时长  
    public String error; //错误的文字描述  
    public float videoFrameRate; //播放的平均帧率  
    public String videoCodecName; //视频编码格式  
    public String audioCodecName; //音频编码格式  
    public String subtitleCodecName; //字幕编码格式  
    public String containerName; //容器格式  
    public int streamType; //流类型  
    public int errorId; //错误码 跟 error 对应  
    public boolean isHwdecoder; //是否硬解  
    public int skipFrameCount; //跳帧统计  
    public boolean isFindStreamInfo; //是否寻找流信息  
    public int IOSeekCount; //ffmpeg 内部 seek 次数  
    public int IOSeekTimeMS; //ffmpeg 内部 seek 消耗时间 MS
```

```

public boolean ffConfigureBuffersForIndexFlag; // false: 执行优化逻辑, true: 走原始逻辑
public long ffConfigureBuffersForIndexExeTimeUS; // ffmpeg/libavformat/utils.c ff_configure_buffers_for_index 函数执行的时间
public int firstBufferDuration; //第一次缓冲持续的时间
public int firstBufferInterval; //第一次缓冲发生的时间间隔
public int secondBufferDuration; //第二次缓冲持续的时间
public int secondBufferInterval; //第二次缓冲发生的时间间隔

// 以下所有的统计都只记录到首帧渲染为止(时间单位 ms)
public float bxbbReadAverageSpeedKB_S; // 读取流的平均速度 KB/S
public long bxbbIoSeekSum; // IO seek 总次数 (open 和 seek)
public long bxbbIoSeekSumTime; // IO seek 总时长 (open 和 seek)
public long bxbbOpenSucceedDataSum; // 成功打开时的数据量
public long bxbbFirstRenderDataSum; // 到首帧时的数据量
public long bxbbIoOperateExceptTime; // IO 操作以外的时间
public long bxbbOpenToAvformatOpenTime; // 调 open 到 ffmpeg open 之前的时间
public long bxbbAvformatOpenTime; // ffmpeg open 的时间
public long bxbbAvformatFindStreamTime; // 寻找流的时间
public long bxbbFindStreamToOpenSucceedTime; // 寻找流信息到成功打开的时间
public long bxbbOpenSucceedToFirstFrameRenderTime; // 打开成功到首帧时间
public long bxbbOpenToFirstFrameRenderTime; // 调 open 到首帧时间
public Map<String, String> map = new HashMap<String, String>();

}

```

其中 map 成员有所有其它成员的信息。

**1.1.34**     `static long preOpen(String url, AHttp ahttp, int readPacket, int time, OnPreOpenProgressListener onPreOpenProgressListener, PreFragmentParameter preFragmentParameter, int iSetPosition);`

重载函数 1: `static long preOpen(String url, OnPreOpenProgressListener onPreOpenProgressListener)`

重载函数 2: `static long preOpen(String url, OnPreOpenProgressListener onPreOpenProgressListener, int iSetPosition)`

重载函数 3: `static long preOpen(String url, boolean useHttp, OnPreOpenProgressListener onPreOpenProgressListener)`

重载函数 4: `static long preOpen(String url, boolean useHttp, OnPreOpenProgressListener onPreOpenProgressListener, int iSetPosition)`

重载函数 5: `static long preOpen(String url, boolean useHttp, int readPacket, OnPreOpenProgressListener onPreOpenProgressListener)`

重载函数 6: `static long preOpen(String url, boolean useHttp, int readPacket, OnPreOpenProgressListener onPreOpenProgressListener, int iSetPosition)`

重载函数 7: `static long preOpen(String url, String cachePath, int time, OnPreOpenProgressListener onPreOpenProgressListener)`

重载函数 8: `static long preOpen(String url, String cachePath, int time, OnPreOpenProgressListener onPreOpenProgressListener, int iSetPosition)`

重载函数 9: `static long preOpen(String url, AHttp ahttp, int readPacket, int time, OnPreOpenProgressListener onPreOpenProgressListener)`

重载函数 10: `static long preFragment(String url, OnPreOpenProgressListener onPreOpenProgressListener, PreFragmentParameter preFragmentParameter)`

本函数用于预先打开一个媒体，用于下载库，或者文件缓存。

**url:** 启动的任务链  
地址或者本地文件路径。

**ahttp:** android 的 http  
接口。

**readPacket:** 读取的包个数。//与  
time, preFragmentParameter 互斥

**time:** 读取媒体前多少  
毫秒，单位 ms。//与 readPacket, preFragmentParameter 互斥

**onPreOpenProgressListener:** 预加载过程的进度回调。

**preFragmentParameter:** 片段式预加载方式。// 与 readPacket, time 互斥。

**iSetPosition:** 预加载的起始位置，单位  
ms。

**返回值:** preOpen 的返回值  
是 c++类的指针。

配合: `static void closePreOpen(long preOpenObj);` 使用。传入的 preOpenObj 是 preOpen 返回的指针。

重载函数: `static void closePreOpen(String url);` 输入 url 将会关闭 preOpen 启动的 url 任务。

```

public static class OnPreOpenProgressListener {

    public final static String WIDTH = "Width";
    public final static String HEIGHT = "Height";
    public final static String DURATION_MS = "Duration-MS";
    public final static String FILE_SIZE = "File-Size";

    public int preOpenMediaInfo(Map<String, String> map) {

        return 0;

    }

    public int preOpenProgress(int progress) {

        return 0;

    }

}

```

其中 class OnPreOpenProgressListener 继承后可重写 preOpenMediaInfo() 跟 preOpenProgress() 可捕获回调消息。

preOpenMediaInfo(Map<String, String> map) 中回调的 map 有 String WIDTH, String HEIGHT, String DURATION\_MS, String FILE\_SIZE 等四个信息。

preOpenProgress(int progress) 中回调的 progress 为预加载进度。正常情况下为：0-100，失败时为负数。

## 1.2 事件注册

可注册关注的事件，以响应特定事件，一个事件只能注册一个监听器，后注册的监听器会覆盖先注册的监听器。

注意：事件函数中，不要执行耗时操作。

### 1.2.1 文件打开成功事件

```
void setOnOpenSuccessListener (OnOpenSuccessListener listener):
```



```
public interface OnOpenSuccessListener{

    void onOpenSuccess();

}
```

在文件打开成功后，会产生该事件。

如果没有设置自动播放，在该事件接口函数中调用Play（）就可以开始播放了。要获得视频的宽高，时长的信息 也需要在视频打开成功之后。

该事件与 OnOpenCompleteListener 事件在回调函数的参数为 TRUE 时一模一样。

### 1.2.2 播放状态改变事件

```
void setOnPlayStateChangeListener (OnPlayStateChangeListener listener):
```

```
public interface OnPlayStateChangeListener{

    void onPlayStateChange(int nCurrentState,int nPreState);

}
```

在播放器状态改变时，会产生该事件。

nCurrentState: 当前状态。

nPreState: 改变前的状态。

如（1.12）对状态的描述可知，状态的改变可以知道很多的事情，但其中很多事情都有了独立的事件，因此一般情况下没有必要使用该事件。

### 1.2.3 打开调用完毕事件

```
void setOnOpenCompleteListener (OnOpenCompleteListener listener):
```

```
public interface OnOpenCompleteListener{

    void onOpenComplete(boolean isOpenSuccess);

}
```

在文件打开操作执行完毕后，会产生该事件。

isOpenSuccess: 文件是否打开成功

当参数 isOpenSuccess 为 TRUE 时 该事件与 OnOpenSuccessListener 事件一样。

当参数 isOpenSuccess 为 FLASE 时 该事件与 OnPlayCompleteListener 事件在参数为 PLAYRE\_RESULT\_OPENRROR(“0x80000001”)时一样。

从上面可以看出，这个事件完全可以被别的事件替代，因此需要注意可能接受相同的消息而导致一些动作重复执行而发生错误

#### 1.2.4 播放完成事件

```
void setOnPlayCompleteListener(OnPlayCompleteListener listener):
```

```
public interface OnPlayCompleteListener{

    void onPlayComplete(String playRet);

}
```

播放完成，该事件将会被调用。

playRet: 播放完成返回的结果码。

为“0”—— 视频播放完成。

为“1”—— 用户主动结束播放。

其他字符串（非 0 非 1）—— 播放出错结束，返回错误代码。

当 playRet 为 PLAYRE\_RESULT\_OPENRROR 时 表示打开失败与 OnOpenCompleteListener 事件在参数为 FLASE 时表示相同的事件。

当 playRet 为 PLAYRE\_RESULT\_HARDDECODERERROR 时表示因为硬件解码出错而导致播放结束。因为硬解出错可能无法恢复，因此建议在进程的此次生命周期之内不再使用硬解。播放事件发生后，当前媒体文件（流）的相关操作失效，需要再次调用 Open（）。

### 1.2.5 缓冲进度改变事件

```
void setOnBufferListener(OnBufferListener listener):
```

```
public interface OnBufferListener{  
    void onBuffer(int progress);  
}
```

缓冲数据，当缓冲进度改变时，会产生该事件。

Progress: 当前新的缓冲进度。

在非缓冲状态下，如果首次接受到该消息，则播放器进入缓冲状态（不论缓冲进度是多少），当缓冲进度为 100 时 缓冲结束 退出缓冲状态。

注意播放器进入缓冲后，播放器的状态并没有发生改变。

### 1.2.6 媒体 Seek 完成事件

```
void setOnSeekCompleteListener(OnSeekCompleteListener listener):
```

```
public interface OnSeekCompleteListener{  
    void onSeekComplete();  
}
```

当 Seek 结束时，会产生该事件。

如果用户想在 Seek 结束后给用户提示，可以定制该事件。

### 1.2.7 Surface 销毁事件：

```
void setOnSurfaceDestroyListener(OnSurfaceDestroyListener listener):
```

```
public interface OnSurfaceDestroyListener{  
    void OnSurfaceDestroy();  
}
```

当媒 Surface 销毁时，会产生该事件。

当使用硬解或者系统播放器时，Surface 销毁后需要停止播放器，不然可能导致崩溃或者黑屏。注意 Surface 销毁时 Activity 也可能销毁了，你可能在 Activity 销毁事件里面已经做了处理，然后又在该事件中又重复做了处理，然后导致冲突。

### 1.2.8 字幕更新事件

```
void setOnShowSubtitleListener(OnShowSubtitleListener listener):
```

```
public interface OnShowSubtitleListener{  
    void OnShowSubtitle(String subtitle);  
}
```

当字幕需要更新时，会产生该事件。

Subtitle 内容不为空 —— 显示新的字幕

Subtitle 内容为空 —— 清除显示的字幕。

可以注册该事件，来自定义实现字幕显示，用户无需关心字幕显示和消失的时间，消息会在正确的时间出现，但自定义显示时，应关闭 APlayer 的内部显示 `SetConfig(APlayerAndroid.SUBTITLE_SHOW, "0");`。

## 1.3 规格说明

### 1.3.1 支持的文件格式:

```
aa flic m4v
aac flv matroska
ac3 flv mgsts
acm fourxm microdvd
act frm mjpeg
adf fsb mjpeg_2000
adp g722 mlp
ads g723_1 mlv
adx g729 mm
aea genh mmf
afc gif mov
aiff gsm mp3
aix gxf mpc
amr h261 mpc8
anm h263 mpegps
apc h264 mpegts
ape hevc mpegtsraw
apng hls mpegvideo
aqtitle hnm mpjpeg
asf ico mpl2
asf_o idcin mpsub
ass idf msf
ast iff msnwc_tcp
au ilbc mtaf
avi image2 mtv
avr image2_alias_pix musx
avs image2_brender_pix mv
bethsoftvid image2pipe mvi
bfi image_bmp_pipe mxf
bfstm image_dds_pipe mxg
bink image_dpx_pipe nc
bintext image_exr_pipe nistsphere
bit image_j2k_pipe nsv
bmv image_jpeg_pipe nut
boa image_jpegls_pipe nuv
brstm image_pam_pipe ogg
c93 image_pbm_pipe oma
caf image_pcx_pipe paf
cavsvideo image_pgm_pipe pcm_alaw
cdg image_pgmyuv_pipe pcm_f32be
cdxl image_pictor_pipe pcm_f32le
```

cine image\_png\_pipe pcm\_f64be  
concat image\_ppm\_pipe pcm\_f64le  
data image\_psd\_pipe pcm\_mulaw  
daud image\_qdraw\_pipe pcm\_sl6be  
dcstr image\_sgi\_pipe pcm\_sl6le  
dfa image\_sunrast\_pipe pcm\_s24be  
dirac image\_tiff\_pipe pcm\_s24le  
dnxhd image\_webp\_pipe pcm\_s32be  
dsf image\_xpm\_pipe pcm\_s32le  
dsicin ingenient pcm\_s8  
dss ipmovie pcm\_ul6be  
dts ircam pcm\_ul6le  
dtshd iss pcm\_u24be  
dv iv8 pcm\_u24le  
dvbsub ivf pcm\_u32be  
dvbtxt ivr pcm\_u32le  
dxa jacosub pcm\_u8  
ea jv pjs  
ea\_cdata live\_flv pmp  
eac3 live\_flv pva  
epaf lmlm4 pvf  
ffm loas qcp  
ffmetadata lrc r3d  
filmstrip lvf rawvideo  
flac lxf realtext  
redspark sox vivo  
rl2 spdif vmd  
rm srt vobsub  
roq stl voc  
rpl str vpk  
rsd subviewer vplayer  
rso subviewer1 vqf  
rtp sup w64  
rtsp svag wav  
sami swf wc3  
sap tak webm\_dash\_manifest  
sbg tedcaptions webvtt  
scc thp wsaud  
sdp threedostr wsd  
sdr2 tiertexseq wsvqa  
sds tmv wtv  
sdx truehd wv  
segafilm tta wve  
shorten tty xa  
siff txd xbin  
sln v210 xmv

smacker v210x xvag  
smjpeg vag xwma  
smush vcl yop  
sol vclt yuv4mpegpipe

## 1.3.2 支持的解码器格式

### 1.3.2.1 软解:

aac aura ffv1  
aac\_fixed aura2 ffvhuff  
aac\_latm avrn ffwavesynth  
aasc avrp fic  
ac3 avs flac  
ac3\_fixed avui flashsv  
adpcm\_4xm ayuv flashsv2  
adpcm\_adx bethsoftvid flic  
adpcm\_afc bfi flv  
adpcm\_aica bink fmvc  
adpcm\_ct binkaudio\_dct fourxm  
adpcm\_dtk binkaudio\_rdft fraps  
adpcm\_ea bintext frwu  
adpcm\_ea\_maxis\_xa bmp g2m  
adpcm\_ea\_r1 bmv\_audio g723\_1  
adpcm\_ea\_r2 bmv\_video g729  
adpcm\_ea\_r3 brender\_pix gif  
adpcm\_ea\_xas c93 gsm  
adpcm\_g722 cavs gsm\_ms  
adpcm\_g726 ccaption h261  
adpcm\_g726le cdgraphics h263  
adpcm\_ima\_amv cdxl h263i  
adpcm\_ima\_apc cfhd h263p  
adpcm\_ima\_dat4 cinepak h264  
adpcm\_ima\_dk3 clearvideo hap  
adpcm\_ima\_dk4 cljr hevc  
adpcm\_ima\_ea\_eacs cllc hnm4\_video  
adpcm\_ima\_ea\_sead comfortnoise hq\_hqa  
adpcm\_ima\_iss cook hqx  
adpcm\_ima\_oki cpia huffyuv  
adpcm\_ima\_qt cscd iac  
adpcm\_ima\_rad cyuv idcin  
adpcm\_ima\_smjpeg dca idf  
adpcm\_ima\_wav dds iff\_ilbm  
adpcm\_ima\_ws dfa imc  
adpcm\_ms dirac indeo2  
adpcm\_mtaf dnxhd indeo3

adpcm\_psx dpx indeo4  
adpcm\_sbpro\_2 dsd\_lsbfs indeo5  
adpcm\_sbpro\_3 dsd\_lsbfs\_planar interplay\_acm  
adpcm\_sbpro\_4 dsd\_msbfs interplay\_dpcm  
adpcm\_swf dsd\_msbfs\_planar interplay\_video  
adpcm\_thp dsicinaudio jacosub  
adpcm\_thp\_le dsicinvideo jpeg2000  
adpcm\_vima dss\_sp jpegls  
adpcm\_xa dst jv  
adpcm\_yamaha dvaudio kgv1  
aic dvbsub kmvc  
alac dvbsub lagarith  
alias\_pix dvvideo loco  
als dxa ml01  
amrnb dxtory mace3  
amrwb dxv mace6  
amv eac3 magicyuv  
anm eacmv mdec  
ansi eamad metasound  
ape eatgq microdvd  
apng eatgv mimic  
ass eatqi mjpeg  
asv1 eightbps mjpegb  
asv2 eightsvx\_exp mlp  
atrac1 eightsvx\_fib mmvideo  
atrac3 escapel24 motionpixels  
atrac3al escapel30 movtext  
atrac3p evrc mpl  
atrac3pal exr mplfloat  
mp2 pcm\_u32be svq1  
mp2float pcm\_u32le svq3  
mp3 pcm\_u8 tak  
mp3adu pcm\_zork targa  
mp3adufloat pcx targa\_y216  
mp3float pgm tdsc  
mp3on4 pgmyuv text  
mp3on4float pgssub theora  
mpc7 pictor thp  
mpc8 pixlet tiertexseqvideo  
mpeg1video pjs tiff  
mpeg2video png tmv  
mpeg4 ppm truehd  
mpegvideo prores truemotion1  
mpl2 prores\_lgpl truemotion2  
msa1 psd truemotion2rt  
msmpeg4v1 ptx truespeech



mjpeg4v2 qcelp tscc  
mjpeg4v3 qdm2 tscc2  
msrle qdmc tta  
mss1 qdraw twinvq  
mss2 qpeg txd  
msvideol qtrle ulti  
mszh rl0k utvideo  
mts2 r210 v210  
mvc1 ra\_144 v210x  
mvc2 ra\_288 v308  
mpeg ralf v408  
nellymoser rawvideo v410  
nuv realtext vb  
on2avc rl2 vble  
opus roq vcl  
paf\_audio roq\_dpcm vclimage  
paf\_video rpza vcrl  
pam rscv vmdaudio  
pbm rv10 vmdvideo  
pcm\_alaw rv20 vmnc  
pcm\_bluray rv30 vorbis  
pcm\_dvd rv40 vp3  
pcm\_f16le s302m vp5  
pcm\_f24le sami vp6  
pcm\_f32be sanm vp6a  
pcm\_f32le scpr vp6f  
pcm\_f64be screenpresso vp7  
pcm\_f64le sdx2\_dpcm vp8  
pcm\_lxf sgi vp9  
pcm\_mulaw sgirle vplayer  
pcm\_sl6be sheervideo vqa  
pcm\_sl6be\_planar shorten wavpack  
pcm\_sl6le sipr webp  
pcm\_sl6le\_planar smackaud webvtt  
pcm\_s24be smacker wmalossless  
pcm\_s24daud smc wmapro  
pcm\_s24le smvjpeg wmv1  
pcm\_s24le\_planar snow wmv2  
pcm\_s32be sol\_dpcm wavoice  
pcm\_s32le sonic wmv1  
pcm\_s32le\_planar sp5x wmv2  
pcm\_s64be speedhq wmv3  
pcm\_s64le srt wmv3image  
pcm\_s8 ssa wnv1  
pcm\_s8\_planar stl ws\_snd1  
pcm\_ul6be subrip xan\_dpcm

pcm\_ul6le subviewer xan\_wc3  
pcm\_u24be subviewer1 xan\_wc4  
pcm\_u24le sunrast xbin  
xbm xsub yuv4  
xface xwd zerol2v  
xl y4lp zerocodec  
xma1 ylc zlib  
xma2 yop zmbv  
xpm

#### 1.3.2.2 硬解:

"video/avc"

"video/hevc"

"video/mp4v-es"

#### 1.3.3 支持字幕格式:

##### 1.3.3.1 内部字幕:

ass

##### 1.3.3.2 外挂字幕:

ass smi srt pjs stl psb

#### 1.3.4 支持的过滤器:

atempo // 用于变速不变调

#### 1.3.5 支持的协议:

async http rtmpt  
cache httpproxy rtmpte  
concat rtmpts  
crypto icecast rtp  
data md5 srtp  
ffrtmpcrypt mmsh subfile  
ffrtmphttp mmst tcp  
file pipe tee  
ftp prompeg tls\_openssl  
gopher rtmp udp  
hls rtmpe udplite  
hook rtmpts unix

## 1.4 常见问题

1. 部分机型，在软硬解切换后，屏幕变黑或图像不动，一般是 SurfaceView 无效导致，将 SurfaceView 重绘即可。

```
//stop player

//reopen player

displayView.setVisibility(View.GONE);

...

displayView.setVisibility(View.VISIBLE);
```

## 1.5 更新记录

### V1.2.0.201 更新

- 1) 录制支持网络流

录制支持网络流 (http, rtmp, rtsp 等)。

- 2) 媒体信息获取优化

媒体信息获取支持网络流 (http, rtmp, rtsp 等)。

截图参数宽高参数无效（为非正数）时设置为原始视频宽高。

### V1.2.0.200 更新

- 1) 设置图像原始播放比例接口，只支持获取，不支持设置。

*ASPECT\_RATIO\_NATIVE* 属性只支持 getConfig(), 设置为原始比例，可结合 *ASPECT\_RATIO\_CUSTOM* 属性实现。

1.

1. 录制接口改变

isRecord() 方法更名为 isRecording()

添加 isSupportRecord() 接口，判断当前视频是否支持录制。

1.

1. 支持播放速度设置

添加新的 ConfigID ( *PLAY\_SPEED* ), 支持播放速率设置。

## V1. 2. 0. 100 更新

1) 支持 H265 硬件解码。

2) 提高 H264 硬件解码兼容性。

3) 支持视频截取功能 (MP4、FLV、AVI、3GP、MKV、MOV、TS) 。

4) 支持视频缩略图获取。

5) 修复了一些崩溃问题。

## V1. 1. 0. 30 更新

1) setConfig() VR\_MODEL 暂不支持 左右 3D，同步文档。

## V1. 1. 0. 29 更新

1) PlayCompleteRet 中，自动播放结束事件、手动关闭完成事件 进行更新。

自动播放结束事件 PLAYRE\_RESULT\_COMPLETE 返回值由 " 0 " 更新为 " 0x0 " 。

手动关闭完成事件 PLAYRE\_RESULT\_CLOSE 返回值由 " 1 " 更新为 " 0x1 " 。

# 2，APlayer: IOS 端接口说明

## 2. 1 事件注册:

```
#pragma mark - APlayerIOS Delegate
@protocol APlayerIOSDelegate <NSObject>
```

```
@optional
```

### 2.1.1 播放状态改变事件

```
- (void) player:(APlayerIOS *)player OnStateChanged:(NSInteger)nOldState
nNewState:(NSInteger)nNewState;
```

在播放器状态改变时，会产生该事件。

nOldState: 当前状态。

nNewState: 改变前的状态。

### 2.1.2 文件打开成功事件

```
- (void) OnOpenSucceeded:(APlayerIOS *)player;
```

在文件打开成功后，会产生该事件。

如果没有设置自动播放，在该事件接口函数中调用 Play（）就可以开始播放了。要获得视频的宽高，时长的信息 也需要在视频打开成功之后。

该事件与(- (void) player:(APlayerIOS \*)player OnOpenComplete:(BOOL) isOpenSuccess)事件在回调函数的参数 isOpenSuccess 为 TRUE 时一模一样。

### 2.1.3 打开调用完毕事件

```
- (void) player:(APlayerIOS *)player OnOpenComplete:(BOOL) isOpenSuccess;
```

在文件打开操作执行完毕后，会产生该事件。

isOpenSuccess: 文件是否打开成功

当参数 isOpenSuccess 为 TRUE 时 该事件与 OnOpenSuccessListener 事件一样。

当参数 isOpenSuccess 为 FLASE 时 该事件与(- (void) player:(APlayerIOS \*)player OnPlayComplete:(NSString\*) playRet;) 事件在参数为 AP\_PLAYRE\_RESULT\_OPENRROR(“0x80000001”)时一样。

从上面可以看出，这个事件完全可以被别的事件替代，因此需要注意可能接受相同的消息而导致一些动作重复执行而发生错误

### 2.1.4 播放完成事件

- (void) player:(APlayerIOS \*)player OnPlayComplete:(NSString\*) playRet;

播放完成，该事件将会被调用。

playRet: 播放完成返回的结果码。

为“0”—— 视频播放完成。

为“1”—— 用户主动结束播放。

其他字符串（非 0 非 1）—— 播放出错结束，返回错误代码。

当 playRet 为 AP\_PLAYRE\_RESULT\_OPENRROR 时表示打开失败与 (- (void) player:(APlayerIOS \*)player OnOpenComplete:(BOOL) isOpenSuccess;) 事件在参数为 FLASE 时表示相同的事件。

当 playRet 为 AP\_PLAYRE\_RESULT\_HARDDECODERROR 时表示因为硬件解码出错而导致播放结束。因为硬解出错可能无法恢复，因此建议在进程的此次生命周期之内不再使用硬解。播放事件发生后，当前媒体文件（流）的相关操作失效，需要再次调用 Open（）。

### 2.1.5 首帧渲染事件

- (void) OnFirstFrameRender:(APlayerIOS \*)player;

首帧渲染出画面。

### 2.1.6 视频宽高改变事件

- (void) OnVideoSizeChange:(APlayerIOS \*)player width:(int)width height:(int)height;

视频的原始宽高事件。

width:视频的宽

height:视频的高

### 2.1.7 媒体 Seek 完成事件

- (void) player:(APlayerIOS \*)player OnSeekCompleted:(NSInteger)nPosition;

当 Seek 结束时，会产生该事件。

如果用户想在 Seek 结束后给用户提示，可以定制该事件。

### 2.1.8 字幕更新事件

- (void) player:(APlayerIOS \*)player OnShowSubtitle:(NSString\*) subtitle;

当字幕需要更新时，会产生该事件。

Subtitle 内容不为空 —— 显示新的字幕

Subtitle 内容为空 —— 清除显示的字幕。

### 2.1.9 缓冲进度改变事件

- (void) player:(APlayerIOS \*)player OnBufferProgress:(CGFloat)nProgress;

缓冲数据，当缓冲进度改变时，会产生该事件。

Progress: 当前新的缓冲进度。

在非缓冲状态下，如果首次接受到该消息，则播放器进入缓冲状态（不论缓冲进度是多少），当缓冲进度为 100 时 缓冲结束 退出缓冲状态。

注意播放器进入缓冲后，播放器的状态并没有发生改变。

@end

## 2.2 接口说明

2.2.1 APlayerIOS\* aplayer = [[APlayerIOS alloc] init];

创建 APlayerIOS Object-c 对象。

2.2.2 - (void) setDelegate:(\_\_weak id<APlayerIOSDelegate>) delegate

注册事件。

2.2.3 @property (atomic, readonly)NSDictionary\* playStatisticsInfoDict;

获取 NSDictionary 类型的统计信息，Close 之后被赋值，Close 之前为 nil

2.2.4 - (APlayerIOS \*) Init: (UIView \*)APlayerView \_\_deprecated;

已经弃用。

2.2.5 - (void) SetView: (UIView \*)videoView;

设置播放的 UIView; //只能设置一次，不可更换。尺寸变换不需要重设，内部铺满整个 UIView。

2.2.6 - (NSInteger) Open: (NSString \*)strURL;

打开一个视频文件，参数是视频文件路径，可以是本地文件，也可以是支持的网络协议媒体流。该函数立即返回，调用成功返回 0（但并不表示视频文件打开成功，只是函数调用成功）

- (NSInteger) Open: (NSString \*)strURL at\_postion:(int)time\_ms;

多一个参数，time\_ms：在某个时间点打开媒体，单位 ms；

### 2.2.7 - (NSInteger) OpenWithCustomIO: (NSObject<APlayerCustomIO>\*)customIO;

以自定义 IO 的方式打开播放内核。

- (NSInteger) OpenWithCustomIO: (NSObject<APlayerCustomIO>\*)customIO  
at\_postion:(int)time\_ms;

多一个参数，time\_ms：在某个时间点打开媒体，单位 ms；

### 2.2.8 - (NSInteger) Close;

关闭播放器，成功返回 0。该函数是非阻塞函数，真正的关闭动作在另外的线程异步执行

### 2.2.9 - (void) Destroy;

彻底释放播放器资源，该函数一般最后调用。先调用 Close() 再调用 Destroy()；

### 2.2.10 - (NSInteger) Play;

开始播放视频。视频打开成功后不会立刻播放，而是转为暂停状态，如果需要播放视频，需要调用该函数，因此该函数一般是在 Open 成功事件中调用，或者从暂停状态恢复播放状态时调用。该函数立即返回，返回值为-1 表示失败，返回值为 0 表示成功。

### 2.2.11 - (NSInteger) Pause;

暂停视频播放。暂停音视频解码和渲染，数据读取过程继续直至填满内部缓冲区，成功返回 0。

### 2.2.12 - (NSString \*) GetVersion;

返回当前 jar 包版本字符串，形如：“1.2.0.100”

### 2.2.13 - (NSInteger) GetState;

得到播放器目前的状态。播放器有 7 个如下状态。

```
#import "APIOSConstant.h"
```

```
enum PlayerState
```

```
{
```



```
AP_STATE_READY = 0,  
AP_STATE_OPENING = 1,  
AP_STATE_PAUSING = 2,  
AP_STATE_PAUSED = 3,  
AP_STATE_PLAYING = 4,  
AP_STATE_PLAY = 5,  
AP_STATE_CLOSEING = 6,  
AP_STATE_ERROR = 100,  
  
};
```

7 个状态中 AP\_STATE\_READY, AP\_STATE\_PAUSED, AP\_STATE\_PLAYING 是稳态

AP\_STATE\_OPENING、AP\_STATE\_PAUSING、AP\_STATE\_PLAY、*AP\_STATE\_CLOSEING* 是瞬时状态，各个状态之间可以互相转换，状态转换图如下

有多种情况会使播放器进入 AP\_STATE\_READY 状态，进入原因可通过 `GetConfig` 方法查询 `CONFIGID_PLAYRESULT` 信息。

#### 2.2.14 - (NSInteger) GetDuration;

得到视频的 duration, 单位为毫秒。视频打开成功后调用该函数获得视频的 duration, 但视频文件可能不包含该字段信息，这时得到的 duration 为-1，内部可以通过读取一定数量的包之后来估算视频的 duration, 播放一段时间可获取到 duration 估计值（等待时间越久相对越精确）。

#### 2.2.15 - (NSInteger) GetPosition;

得到当前播放进度，单位为毫秒。

#### 2.2.16 - (NSInteger) SetPosition: (NSInteger)nPosition; //ms

设置当前播放进度，单位为毫秒，成功返回 0。

#### 2.2.17 - (NSInteger) GetVideoWidth;

返回获取到的视频宽度，单位像素(pixel)，如不包含视频流，则返回 0。打开成功后才能正确得到视频的宽度，如果视频是 90 度旋转的，该函数得到的值是经过调整的

#### 2.2.18 - (NSInteger) GetVideoHeight;

返回获取到的视频高度，单位像素(pixel)，如不包含视频流，则返回 0。打开成功后才能正确得到视频的高度，如果视频是 90 度旋转的，该函数得到的值是经过调整的

#### 2.2.19 - (APlayerStatisticsInfo\*) GetStatisticsInfo;

获取整个播放过程的统计信息，Close 之后有效，Close 之前无效。

## 2.2.20 - (NSDictionary \*)GetStatisticsInfoDict;

获取 NSDictionary 类型的统计信息，Close 之后被赋值，Close 之前为 nil

//deprecated, see GetState

## 2.2.21 - (BOOL) IsSeeking \_\_deprecated;

已经弃用

## 2.2.22 - (NSInteger) GetBufferProgress;

获取当前缓冲进度 [0, 100], 不在缓冲和返回负值，缓冲进度也可以在缓冲消息中获取。进度 0 表示开始缓冲，100 表示结束缓冲。

## 2.2.23 - (NSString \*) GetConfig: (NSInteger)nConfigId;

说明见: [ - (NSInteger) SetConfig: (NSInteger)nConfigId strValue: (NSString \*)strValue; ]

## 2.2.24 - (NSInteger) SetConfig: (NSInteger)nConfigId strValue: (NSString \*)strValue;

设置和获取 APlayer 的参数，APlayer 中将所有非基本的功能集中在该函数中，以减少接口数量。configID 是属性 ID，所有的属性 ID 都定义在类 CONFIGID 中，value 和 GetConfig 的返回值是属性值，对于不同的属性，value 是字符串形式，但不同参数格式不同。属性配置如下：

```
#define CONFIGID_CURRENTURL 4 //str, R, //Media URL when current play
#define CONFIGID_FILESIZE 5 //int64, R, //Media file size

#define CONFIGID_PLAYRESULT 7 //int, R, //Play Result, 0-play complete, 1-manual
close,
//error code:0x8xxxxxxx error
code: (0x80000001:OpenFileError), (0x80000002:OpeningError), (0x80000003:NotReachable
断网)
#define CONFIGID_AUTO_PLAY 8 //str R,W //auto flage, if set 1 when open
success, will auto paly
//0 - not auto play, 1 - auto play after open success

//#define CONFIGID_READSIZE 29 //int, R, //Read Size, unit:byte

#define CONFIGID_READPOSITION 31 //int, R, //Read Position, unit:ms

#define CONFIGID_UPDATE_WINDOWS 40 //int W //Update and refreshe video frame

#define CONFIGID_PLAY_SPEED 104 //int R,W //play speed, 100 is unit 1, eg.50 is
half speed, 200 is double speed
```

```

#define CONFIGID_DAR 106 //str, R, //get video display aspect ratio,note this
value may be from video_width/video_height

#define CONFIGID_CACHE_PATH 107 // R,W //文件缓存路径 p, 不包含文件后缀,对应缓存文
件为 p.rec 和 p.dat
#define CONFIGID_USE_FILE_CACHE 108 // R,W //是否使用文件缓存,“1” or “0”,默认不
使用文件缓存(0)
#define CONFIGID_MAX_PRECACHE_BYTES 109 // R,W //最大缓存预取字节数,单位字节,-1
为无限制,默认无限制,最大为 MAX_CACHE_BYTES
#define CONFIGID_MAX_CACHE_BYTES 110 // R,W //最大缓存字节数,单位字节,-1为无限
制,默认无限制,最大为 2040MB
#define CONFIGID_MAX_SAVE_MEDIA_HEADER_MS 111 // R,W //close时保留缓存时长(ms),默
认不限时长,不删除的配置下有多少存多少
#define CONFIGID_RETAIN_CACHE_AFTER_USED 112 // R,W //close时是否保留缓存,默认删
除
#define CONFIGID_RENDER_RATE 113 //R, //获取渲染帧率
#define CONFIGID_ASPECT_RATIO_NATIVE 203 //str R, //video Original aspect ration,
eg.“4;3”
#define CONFIGID_ASPECT_RATIO_CUSTOM 204 //str R,W //user set aspect ration, eg.
“16;9”

#define CONFIGID_HW_DECODER_USE 209 //int, R,W, //if support,use hardware decode,
1-enable, 0-disable
#define CONFIGID_LIVE_MODE 205 //int, R,W, //if support,use live mode, 1-enable,
0-disable
#define CONFIGID_LIVE_DELAY_STEP 206
#define CONFIGID_LIVE_DELAY_MAX 207
#define CONFIGID_HW_DECODER_ENABLE 230 //int, R, //query staus, 1-support
hardware, 0-none
#define CONFIGID_HW_DECODER_DETEC 231 //int, R,W, //flag about check system
whthere support haedware decode
//1-detec, 0-not detec

//#define CONFIGID_VIEWMODE 251 //str, R,W, //View Content Mode, 0-
UIViewContentModeScaleAspectFit,
// // // 1-UIViewContentModeScaleAspectFill,

//#define CONFIGID_AUDIOPROCESSUSABLE 401 //int, R, //audio function usable, 1-
usable
#define CONFIGID_AUDIOTRACKLIST 402 //int, R, //Audio track list, split by“;”,
example:“chinese;english”
#define CONFIGID_AUDIOTRACKCURRENT 403 //int, R,W, //Current selected audio track
index, select 0, ..., count - 1
#define CONFIGID_AUDIO_SILENCE 420 //int R,W //Audio Silence,0-normal,has audio,1-
silence

```

```

#define CONFIGID_SUBTITLEUSABLE 501 //int, R //Query subtitle function is usable,
1-usable
#define CONFIGID_SUBTITLEEXTNAMELIST 502 //str, R //Subtitle extname list that
supported, example:"srt;ass"
#define CONFIGID_SUBTITLEFILENAME 503 //str, R,W, //Subtitle filename,
example:"/var/mobile/.../subtitle.srt"
#define CONFIGID_SUBTITLESHOW 504 //int, R,W, //Subtitle visible, 1-show, 0-hide,
default:1
#define CONFIGID_SUBTITLELIST 505 //str, R //Subtitle list, split by";",
example:"chinese;english"
#define CONFIGID_SUBTITLESELECTED 506 //int, R,W, //Subtitle selected indenx,
select 0, count - 1] count(internal subtitle + external subtitle)

#define CONFIGID_NETWORK_BUFFER_ENTER 1001 //int, R,W //how many milliseconds wait
to enter buffer state,when the network does not read the data
#define CONFIGID_NETWORK_BUFFERLEAVE 1002 //int, R,W, //Network buffer leave,
prebuffer packets size to leave buffering, default:300 frame
//#define CONFIGID_NETWORK_NOBUFFERDRY 1003 //int, R,W, //Network bobuffer dry,
drying packets size when no buffer, default:5MB
#define CONFIGID_NETWORK_BUFFER_READ 1003 //int R,W //the max frame number,Pre-
read into memory default:2500 frame
#define CONFIGID_NETWORK_BUFFER_TIME 1005 //int R,W //the max frames duration
time,Pre-read into memory,unit is milliseconds
#define CONFIGID_NETWORK_SEEKBUFFER_WAITTIME 1004 //int R,W //the max time wait to
enter buffer state,after seek operation, unit is milliseconds
#define CONFIGID_NETWORK_SPEED 1006 //int R //the io speed

#define CONFIGID_HTTP_COOKIE 1105 //str R,W //cookie in http request
header,deafult is empty
#define CONFIGID_HTTP_REFERER 1106 //str R,W //referer in http request
header,deafult is empty
#define CONFIGID_HTTP_CUSTOM_HEADERS 1107 //str R,W //set a list of http custom
hrader fields, each field use "\r\n" to split
#define CONFIGID_HTTP_USER_AGENT 1108 //str R,W //User Agent fields, use aplayer's
default value
#define CONFIGID_VIDEO_DECODER_TYPE 1109 //str R //"0": IOS hard decoder
//#define CONFIGID_VALID_AUDIOVIDEO 1199 //int, R, //audio: 1; video: 2

```

#### 2.2.24.1 CONFIGID\_CURRENTURL

支持函数: GetConfig

获取当前播放地址的字符串。

#### 2.2.24.2 CONFIGID\_FILESIZE 5 //int64, R, //Media file size

支持函数：GetConfig

获取媒体文件的大小，单位：byte

2.2.24.3 CONFIGID\_PLAYRESULT 7 //int, R, //Play Result, 0-play complete, 1-manual close,

支持函数：GetConfig

获取播放结果对应如下：

```
const static char* AP_PLAYRE_RESULT_COMPLETE           =
"0x0";//播放自动完成
const static char*
AP_PLAYRE_RESULT_CLOSE                               = "0x1";//主动
退出播放
const static char* AP_PLAYRE_RESULT_OPENRROR           =
"0x80000001";    // 打开媒体错误
const static char* AP_PLAYRE_RESULT_SEEKERROR          =
"0x80000002";    // 拖动后错误
const static char* AP_PLAYRE_RESULT_READDEFRAMERROR    =
"0x80000003";    // 读包错误
const static char* AP_PLAYRE_RESULT_CREATEGRAPHERROR =
"0x80000004";    // 创建视频渲染器错误
const static char* AP_PLAYRE_RESULT_DECODEERROR       =
"0x80000005";    // 解码错误
const static char* AP_PLAYRE_RESULT_HARDDECODERROR    =
"0x80000006";    // 硬解错误
```

2.2.24.4 CONFIGID\_AUTO\_PLAY 8 //str R,W //auto flage, if set 1 when open success, will auto paly

支持函数：GetConfig/SetConfig

文件打开成功后是否自动播放，默认不播放

“0” —— 打开成功后不自动播放（需要调用 Play（）播放）。

“1” —— 打开成功后自动播放。

默认为“0”；

2.2.24.5 CONFIGID\_READSIZE 29 //int, R, //Read Size, unit:byte

支持函数：GetConfig

已经读取了多少个字节。

#### 2.2.24.6 CONFIGID\_READPOSITION 31 //int, R, //Read Position, unit:ms

支持函数: GetConfig

已读取数据的进度, 单位 ms。

#### 2.2.24.7 CONFIGID\_UPDATE\_WINDOWS 40 //int W //Update and refreshe video frame

支持函数: SetConfig

已经弃用。

#### 2.2.24.8 CONFIGID\_PLAY\_SPEED 104 //int R,W //play speed, 100 is unit 1, eg.50 is half speed, 200 is double speed

支持函数: GetConfig/SetConfig

获取/设置当前播放速率, 为正常速率的百分之多少, 以 100 为单位, 即 100 为正常速度, 50 为半速, 200 为两倍速。变速的范围是 50 -200, 如果超过这个范围内部会规范到该范围中

#### 2.2.24.9 CONFIGID\_DAR 106 //str, R, //get video display aspect ratio,note this value may be from video\_width/video\_height

支持函数: getConfig

获取视频显示长宽比, 此值来自 video\_width/video\_height

#### 2.2.24.10 CONFIGID\_CACHE\_PATH 107 // R,W //文件缓存路径 p, 不包含文件后缀, 对应缓存文件为 p.rec 和 p.dat

支持函数: GetConfig/SetConfig

设置缓存文件的路径, 不包含文件后缀, 对应缓存文件为 p.rec 和 p.dat

#### 2.2.24.11 CONFIGID\_USE\_FILE\_CACHE 108 // R,W //是否使用文件缓存, “1” or “0”, 默认不使用文件缓存(0)

支持函数: GetConfig/SetConfig

是否使用文件缓存, “1” or “0”, 默认不使用文件缓存(0)

#### 2.2.24.12 CONFIGID\_MAX\_PRECACHE\_BYTES 109 // R,W //最大缓存预取字节数, 单位字节, -1 为无限制, 默认无限制, 最大为 MAX\_CACHE\_BYTES

支持函数: GetConfig/SetConfig

最大缓存预取字节数, 单位字节, -1 为无限制, 默认为-1。且最大值不会超过 CONFIGID\_MAX\_CACHE\_BYTES 设置的值

2.2.24.13 CONFIGID\_MAX\_CACHE\_BYTES 110 // R,W //最大缓存字节数，单位字节，-1 为无限制，默认无限制，最大为 2040MB

支持函数：GetConfig/SetConfig

缓存文件的最大缓存字节数，单位字节，-1 为无限制，默认为-1，且最大为 2040MB

2.2.24.14 CONFIGID\_MAX\_SAVE\_MEDIA\_HEADER\_MS 111 // R,W //close 时保留缓存时长(ms)，默认不限时长，不删除的配置下有多少存多少

支持函数：GetConfig/SetConfig

缓存文件 close 时保留缓存时长(ms)，默认不限时长，不删除的配置下有多少存多少

2.2.24.15 CONFIGID\_RETAIN\_CACHE\_AFTER\_USED 112 // R,W //close 时是否保留缓存，默认删除

支持函数：GetConfig/SetConfig

close 时是否保留缓存，默认删除

2.2.24.16 CONFIGID\_RENDER\_RATE 113 //R, //获取渲染帧率

支持函数：GetConfig

获取渲染帧率，单位：帧/s

2.2.24.17 CONFIGID\_ASPECT\_RATIO\_NATIVE 203 //str R, //video Original aspect ration, eg. "4;3"

支持函数：getConfig

获取视频的自然宽高比，格式："4;3"。

2.2.24.18 CONFIGID\_ASPECT\_RATIO\_CUSTOM 204 //str R,W //user set aspect ration, eg. "16;9"

支持函数：getConfig/setConfig

视频的自定义宽高比，格式："4;3"。

2.2.24.19 CONFIGID\_HW\_DECODER\_USE 209 //int, R,W, //if support,use hardware decode, 1-enable, 0-disable

支持函数：getConfig/setConfig

需要配合 CONFIGID\_HW\_DECODER\_DETEC 一起使用。

获取/设置硬件解码状态，默认关闭。

” 0” —— 关闭。

” 1” —— 开启。

需要在 Open 之前设置该参数，Open 之后设置对当前播放的媒体无效。如果媒体不支持硬解 设置该参数为” 1”，也只能是软解。。

2.2.24.20 CONFIGID\_LIVE\_MODE 205 //int, R,W, //if support,use live mode, 1-enable, 0-disable

支持函数：getConfig/setConfig

设置直播模式（对直播视频，可加快首帧打开，并且不会进入缓冲，未做追帧操作），open() 前掉用有效。

例：SetConfig(CONFIGID.LIVE\_MODE, ” 1” );

2.2.24.21 CONFIGID\_LIVE\_DELAY\_STEP 206

支持函数：setConfig

直播模式下的延迟时间步长，单位 ms

2.2.24.22 CONFIGID\_LIVE\_DELAY\_MAX 207

支持函数：setConfig

直播模式下的最大延迟时间，单位 ms

2.2.24.23 CONFIGID\_HW\_DECODER\_ENABLE 230 //int, R, //query staus, 1-support hardware, 0-none

支持函数：getConfig

弃用。

2.2.24.24 CONFIGID\_HW\_DECODER\_DETEC 231 //int, R,W, //flag about check system whthere support haedware decode //1-detec, 0-not detec

支持函数：getConfig/setConfig

配合 CONFIGID\_HW\_DECODER\_USE 使用，“1”：探测硬解，“0”：不探测硬解

2.2.24.25 CONFIGID\_AUDIOTRACKLIST 402 //int, R, //Audio track list, split by”;”, example:”chinese;english”

支持函数：GetConfig



获取音轨列表。音轨列表， 格式：“语言, 音轨 1;语言, 音轨 2”。

2.2.24.26 CONFIGID\_AUDIOTRACKCURRENT 403 //int, R,W, //Current selected audio track index, select 0, ..., count - 1

支持函数：GetConfig/SetConfig

获取或设置当前音轨索引， 索引值从” 0” 开始。

2.2.24.27 CONFIGID\_AUDIO\_SILENCE 420 //int R,W //Audio Silence, 0-normal, has audio, 1-silence

支持函数：GetConfig/SetConfig

获取和设置是否静音

“0”	——	不	静	音
“1”	——	静音		

默认值是 “0”

2.2.24.28 CONFIGID\_SUBTITLEUSABLE 501 //int, R //Query subtitle function is usable, 1-usable

支持函数：getConfig

获取字幕加载功能是否可用。

“0”	——	不可用字幕加载
“1”	——	可用字幕加载

2.2.24.29 CONFIGID\_SUBTITLEEXTNAMELIST 502 //str, R //Subtitle extname list that supported, example:”srt;ass”

支持函数：getConfig

未实现

2.2.24.30 CONFIGID\_SUBTITLEFILENAME 503 //str, R,W, //Subtitle filename, example:”/var/mobile/.../subtitle.srt”

支持函数：getConfig/setConfig

获取/设置外挂字幕的文件名，例如：”/subtitle.srt”。

2.2.24.31 CONFIGID\_SUBTITLESHOW 504 //int, R,W, //Subtitle visible, 1-show, 0-hide, default:1

支持函数: getConfig/setConfig

未实现

2.2.24.32 CONFIGID\_SUBTITLELIST 505 //str, R //Subtitle list, split by";",  
example:"chinese;english"

支持函数: getConfig

获取当前加载的字幕的可用语言列表, 用";"分割, 例如: "chinese;english"。

如果加载了外挂字幕, 外挂字幕也在字幕列表中, 位于内嵌字幕后面

2.2.24.33 CONFIGID\_SUBTITLESELECTED 506 //int, R,W, //Subtitle selected index,  
select 0, count - 1] count(internal subtitle + external subtitle)

支持函数: getConfig/setConfig

选择的下标 index, 选择[0, count - 1] count(内部字幕+外部字幕)

2.2.24.34 CONFIGID\_NETWORK\_BUFFER\_ENTER 1001 //int, R,W //how many milliseconds wait  
to enter buffer state,when the network does not read the data

支持函数: getConfig/setConfig

获取/设置 当网络没有读取到数据时, 等待多久进入缓冲状态, 单位毫秒, 注意这里的时间是字符串。默认值是 200 毫秒。

备注: 缓冲状态一般会暂停播放持续一段时间, 以填满缓冲队列。

2.2.24.35 CONFIGID\_NETWORK\_BUFFERLEAVE 1002 //int, R,W, //Network buffer leave,  
prebuffer packets size to leave buffering, default:300 frame

支持函数: getConfig/setConfig

获取/设置 在缓冲状态下, 缓冲多少个帧退出缓冲, 默认为 “300”。该值越大意味着进入缓冲后需要更久才能退出缓冲, 但是后续的播放会更加流畅。

该值设置多少, 并不意味着缓冲多少帧就退出缓冲, 设置的值只是一个基础量, 实际需要缓冲的帧是动态变化的, 随着正常缓冲次数的增加而增加, 最大可以到基础量的 4 倍。

如果是因为拖动而进入缓冲, 所需缓冲的包数减半。另外我们设置的虽然是包的数目, 但是内部计算缓存进度和判断是否结束缓存实际是用的时间, 一个包按照 40ms 来计算

2.2.24.36 CONFIGID\_NETWORK\_BUFFER\_READ 1003 //int R,W //the max frame number,Pre-read  
into memory default:2500 frame

支持函数: getConfig/setConfig

获取/设置 内存队列中视频包的长度，默认值是 2500。队列越长，占用内存越多，网络播放时可能更流畅

**2.2.24.37 CONFIGID\_NETWORK\_BUFFER\_TIME 1005 //int R,W //the max frames duration time,Pre-read into memory,unit is milliseconds**

支持函数: getConfig/setConfig

获取/设置 内存队列中视频包的总时长，单位秒。默认是“0”。也可以用时长表示视频队列的长度，这样更直观，时长乘以帧率就是队列长度。

如果有设置这个值，将会优先使用该值来计算队列长度，该值为 0 忽略该值。

**2.2.24.38 CONFIGID\_NETWORK\_SEEKBUFFER\_WAITTIME 1004 //int R,W //the max time wait to enter buffer state,after seek operation, unit is milliseconds**

支持函数: GetConfig/SetConfig

获取/设置 拖动播放进度后，没有数据时多久进入缓冲（缓冲一直到队列满，较耗时），单位毫秒，默认值是 2000。拖动时将会清空队列中的数据，如果队列中没有数据后

按照常规方式处理，大部分拖动，即使在速度很快的情况也会进入缓冲。

**2.2.24.39 CONFIGID\_NETWORK\_SPEED 1006 //int R //the io speed**

支持函数: GetConfig

获取当前网络 IO 读取速度.

**2.2.24.40 CONFIGID\_HTTP\_COOKIE 1105 //str R,W //cookie in http request header,deafult is empty**

支持函数: getConfig/setConfig

获取/设置 HTTP 请求头中带的 Cookie 字符串，默认为无。

**2.2.24.41 CONFIGID\_HTTP\_REFERER 1106 //str R,W //referer in http request header,deafult is empty**

支持函数: getConfig/setConfig

获取/设置 HTTP 请求头中带的 Referer 字符串，默认为无。

**2.2.24.42 CONFIGID\_HTTP\_CUSTOM\_HEADERS 1107 //str R,W //set a list of http custom hrader fields, each field use "\r\n" to split**

支持函数: getConfig/setConfig

获取/设置 HTTP 自定义头字段列表，每个头字段之间用回车换行符号 “\r\n”(即 0x0d, 0x0a) 分割。

2.2.24.43 CONFIGID\_HTTP\_USER\_AGENT 1108 //str R,W //User Agent fields, use aplayer's default value

支持函数: getConfig/setConfig

获取/设置 HTTP 请求头中带的 User Agent 字符串，默认为 APlayer 的默认值。

2.2.24.44 CONFIGID\_VIDEO\_DECODER\_TYPE 1109 //str R //“0”: IOS hard decoder

支持函数: getConfig

返回值为“0”时为硬解。其它为软解

2.2.25 - (void) setAudioInterrupted:(BOOL) is\_interrupted;

把音频焦点抢回来

2.2.26 -(BOOL) isSupportRecord;

是否支持录制

2.2.27 -(BOOL) isRecording;

是否在录制中

2.2.28 -(BOOL) startRecord:(NSString \*)outPath;

开启录制

2.2.29 -(void) endRecord;

停止录制

2.2.30 +(int) shrinkCacheData:(NSString\*) cacheName;

弃用

2.2.31 +(void) updateLogConfig;

更新日志配置文件。内部使用。

2.2.32 获取缩略图:

头文件: APMediaInfo.h

对象: MediaInfo

```
@interface ScreenShotInfo : NSObject    // 返回参数类
@property(assign, nonatomic) int displayWidth;
@property(assign, nonatomic) int displayHeight;
@property(assign, nonatomic) u_long displayMs;
@property(assign, nonatomic) int imageAvgLuma;
@property(assign, nonatomic) BOOL isKeyFrame;
@property(retain, nonatomic) UIImage* image;
@end

@interface ScreenShotParam : NSObject    // 传入参数类
@property(assign, nonatomic) int displayWidth; //截图宽（按最终显示方向）
@property(assign, nonatomic) int displayHeight; //截图高（按最终显示方向）
@property(assign, nonatomic) NSInteger displayMs; // 获取缩略图的时间点，单位 ms
@property(assign, nonatomic) BOOL isCalcLuma; //是否计算亮度
@property(assign, nonatomic) BOOL isRotationCorrection; //是否进行方向校正
@end

@interface MediaInfo : NSObject

- (id) initWithPath: (NSString*)mediaPath;
- (BOOL) parse;

- (NSInteger) getDisplayWidth;
- (NSInteger) getDisplayHeight;

- (NSInteger) getDurationMs;
- (NSInteger) getFileSize;
- (NSInteger) getBitRate;
- (NSInteger) getVideoRotation;
- (float) getFrameRate ;
- (NSDictionary*) getMetaData;
- (bool) hasVideo;
- (bool) hasAudio;
- (bool) destroy;    // 可以异步调用，让阻塞在 parse 或阻塞在 createBitMap 的函数退出来。

- (ScreenShotInfo*) createBitMap: (ScreenShotParam*)screenShotParam;

@end
```

使用方法:

```
MediaInfo* p_MediaInfo = [[MediaInfo alloc] initWithPath:mediaPath];

if ( [p_MediaInfo parse] == YES ) { // 阻塞函数
```

```

ScreenShotParam* p_ScreenShotParam = [[ScreenShotParam alloc] init];

p_ScreenShotParam.displayWidth          = width;
p_ScreenShotParam.displayHeight         = height;
p_ScreenShotParam.isCalcLuma             = FALSE;
p_ScreenShotParam.isRotationCorrection = FALSE;

for ( int i=0;TRUE; i++ ) {

    p_ScreenShotParam.displayMs = i * 10 * 1000;

    ScreenShotInfo* p_ScreenShotInfo = [p_MediaInfo
    createBitMap:p_ScreenShotParam];    // 可多次调用。

    if ( p_ScreenShotInfo ) {

        UIImage *image = pScreenShotInfo.image;
        dispatch_async(dispatch_get_main_queue(), ^{    // 主线程更新 UI

            _thumImageView.image = image;

        });

    } else {

        break;

    }

}

} else {

    NSLog("媒体解析失败！");

}

```

## 2.3 规格说明:

同 android 端规格说明:

硬解支持: h264, hevc.