ArcVideo®Player SDKv3.5

Player User Manual for Android

目录

第1章: 简介

1.1. 概述

杭州当虹科技有限公司的Android平台多媒体播放引擎v3.5版本SDK主要提供了本地音视频的播放和网络流媒体的播放功能，能够实现较低的性能开销达到较好播放体验。本参考文档将对该SDK的主要函数及使用进行详细的描述，以便开发者能够参考该文档进行快速的开发。本文档中提到的所有接口和定义都是针对java语言。

1.2. SDK 包内容

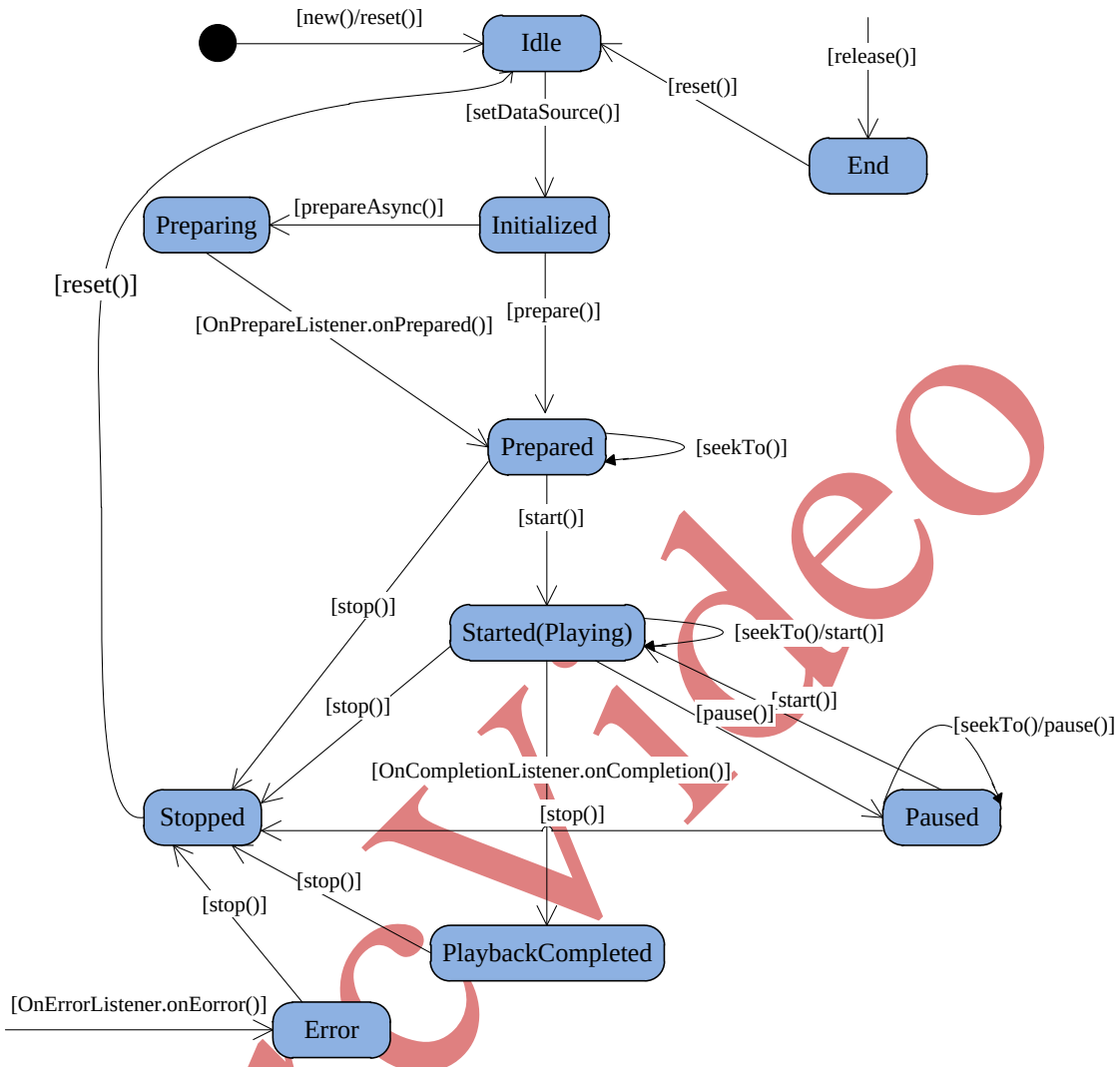
- API 参考文档
- SDK用到的各种库
- 示例代码和测试程序安装包

1.3. 平台支持

- Android 2.0及以上版本

1.4. 播放过程状态图

播放引起通过状态机制来控制本地音视频和网络流媒体的播放过程，如下的状态图展示了整个播放过程从开始到结束的生命周期中各种状态的变化及转换。箭头表示状态转换的方向，箭头上方括号内的函数表示能够引起状态转换的操作。



从这个图可以看出，播放引擎运行过程主要包含如下一些状态：

- 当播放引擎ArcMediaPlayer通过new被创建或者在播放引擎实例对象调用reset()接口之后，就处于Idle状态；当调用了release()接口之后，就处于End状态。在这两个状态之间就是整个播放引擎ArcMediaPlayer的生命周期.
  - 新创建的ArcMediaPlayer实例与调用reset()的实例虽然都处于Idle状态，但是两种情况还是有些重要的细微差别；在Idle状态下调用 getCurrentPosition()、getDuration()、 getVideoHeight()、getVideoWidth()、setAudioStreamType(int)、setLooping(boolean)、 setVolume(float, float)、 pause()、start()、 stop()、seekTo(int)、 prepare() 和 prepareAsync()都被认为是编程错误，虽然不一定会产生严重的错误；如果是用新创建的ArcMediaPlayer实例来调用上述函数，则调用者不会受到任何错误提示或其他消息提示，播放引擎状态不会改变，因为此时还没有注册任何的listener，如果是reset()后调用上述函数，则调用者可能会收到播放引擎内部上报的错误信息或其他消息，播放引擎状态可能会变为Error.
  - 建议调用者在使用完ArcMediaPlayer之后立马调用release()函数来释放播放引擎内部占用的相关资源。如果不释放则可能会导致其占用的硬件资源一直被占用，其他需要访问硬件资源的对象将可能出错。

- 通常在播放过程中会出现各种各样的错误导致播放失败，比如audio\video格式不支持，视频分辨率过高，流媒体数据接收失败等，那么在这种情况下，就需要进行相应的错误上报，有时候甚至是调用者在非法的状态下调用接口也可能导致错误发生；在遇到所有这些错误情况时播放引擎内部会触发错误上报机制将错误信息上报给调用者，错误上报是通过调用者事先通过setOnErrorListener(com.arcvideo.MediaPlayer.ArcMediaPlayer.OnErrorListener)将回调函数设置给播放引擎，播放引擎会通过OnErrorListener.onError()将错误信息上报。
  - 当播放引擎内部发生错误时，ArcMediaPlayer就进入了Error状态，明确这一点很重要，即使是应用层没有注册OnErrorListener，收不到错误信息，但是播放引擎内部状态还是会变成Error；
  - 在Error状态下为了能够重用当前的ArcMediaPlayer实例对象，需要调用者先调用stop()，然后再调用reset()将对象转换到Idle状态；
  - IllegalStateException异常类型的上抛主要是为了避免一些接口调用时序错误，比如在setDataSource被调用之前去调用prepare()\prepareAsync().
- 调用setDataSource(….)的任何一个函数成功之后，播放引擎就会从Idle状态转换到Initialized状态。
- 播放引擎必须先进入Prepared状态，然后才能进入Started状态。
  - 有两种方法(同步和异步)能够进入Prepared状态：一种是调用prepare()同步接口，当该接口返回时就已经进入Prepared状态，另一种是调用prepareAsync()异步接口，该接口返回不代表已经进入Prepared状态，播放引擎内部会异步执行状态转换，先进入Preparing状态，当播放引擎完成了准备工作后，才会进入Prepared状态，此时会通过OnPreparedListener.onPrepared()通知调用者进入Prepared状态，这种情况要求上层调用者提前将OnPreparedListener注册到播放引擎内部，注册函数为setOnPreparedListener(com.arcvideo.MediaPlayer.ArcMediaPlayer.OnPreparedListener);
  - Preparing状态只是一个中间状态，表示正在转换中，在该状态下进行任何其它操作都是不合理的，需要等待状态转换完毕才能进行其它操作；
- 必须通过调用start()接口来开始播放，在start()接口成功返回后，播放引擎就进入到了Started状态，可以通过isPlaying()来查询当前是否处于Started状态。
  - 当播放引擎处于Started状态时，播放引擎内部会通过OnInfoListener.onInfo()回调函数来通知调用者当前播放开始或者结束缓冲数据，开始缓冲消息类型是com.arcvideo.MediaPlayer.ArcMediaPlayer.MEDIA_INFO_BUFFERING_START，结束缓冲的消息类型是com.arcvideo.MediaPlayer.ArcMediaPlayer.MEDIA_INFO_BUFFERING_END，在收到开始缓冲的消息后应用开发者可以通过循环调用getCurrentBufferingPercent()函数来获取缓冲百分比；
  - 如果播放引擎已经处于Started状态，再次调用start()接口不会产生任何影响。
- 播放过程中能够进行暂停和停止播放，通过调用pause()实现暂停，当pause()函数返回后，播放引擎进入Paused状态。
  - 调用start()来恢复处于Paused状态的播放，此时播放会接着暂停时的位置继续播放，当start()函数返回后，播放引擎由Paused状态转换到了Started状态；
  - 如果已经处于Paused状态，继续调用pause()函数不会产生任何影响
- 调用stop()接口停止播放，播放停止后进入Stopped状态，播放引擎在*Started*、*Paused*、*Prepared* 和*PlaybackCompleted*状态下可以调用stop()进入Stopped状态
  - 一旦进入Stopped状态，则播放将无法继续，要想重新进入Started状态，则必须重新调用reset()进入Idle状态，然后重头开始先后调用setDataSource，prepareAsync；
  - 如果已经进入Stopped状态，继续调用stop()不会产生任何影响。
- 播放位置可以通过seekTo(int)进行调整。

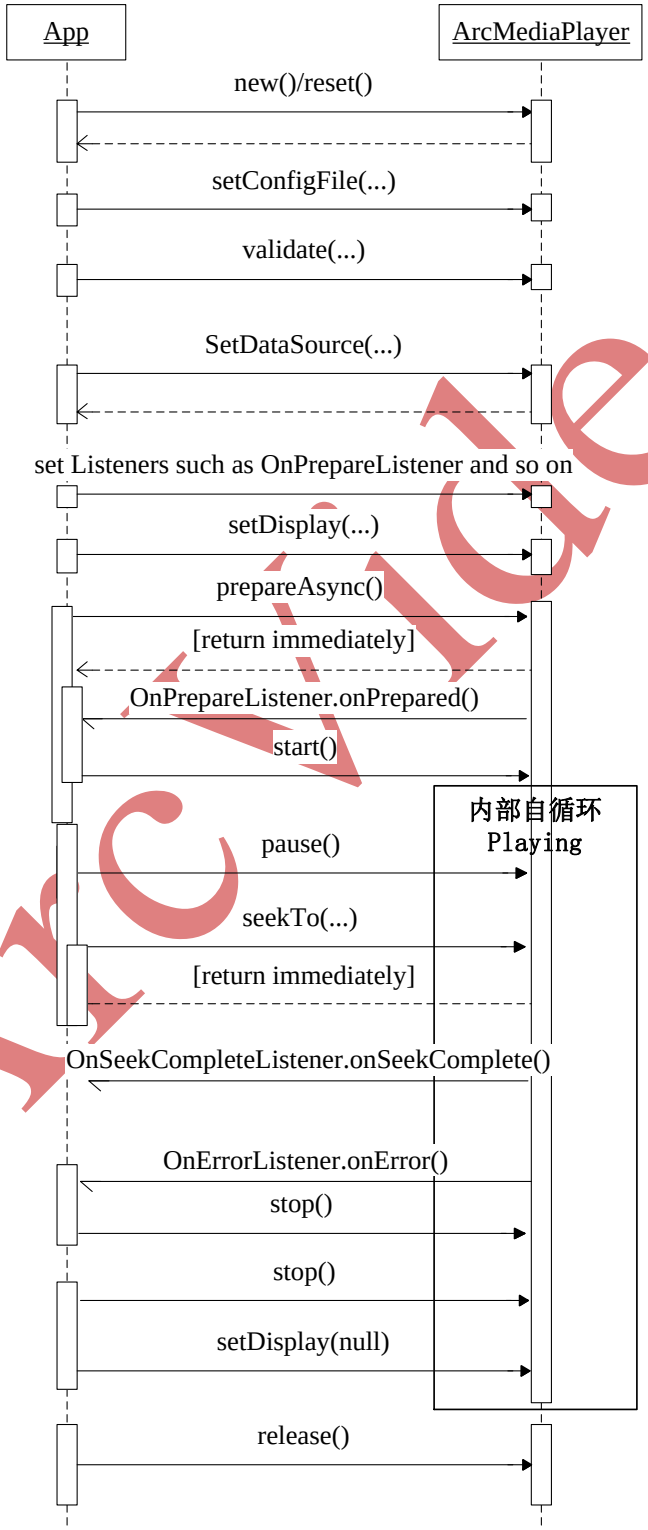
- 虽然seekTo(int)是异步操作，调用时会立即返回，但是实际的seek操作需要一段时间才能完成，特别是网络流媒体播放；当seek操作完成后，播放引擎内部会通过OnSeekComplete.onSeekCompleted()回调函数通知应用开发者，该回调函数是需要应用开发者通过setOnSeekCompleteListener(OnSeekCompleteListener)提前设置给播放引擎；
- SeekTo(int)不仅可以在Started状态下调用，也可以在Prepared、Paused状态下调用；
- 另外，当前的播放位置可以通过getCurrentPosition()接口获取，该接口对于需要使用播放进度条的应用需求很有用。

- 当媒体数据被播放完后，播放引擎进入播放完成状态(PlaybackCompleted)。
  - 当循环播放被设置为true时(setLooping(boolean))，媒体数据被播放完后会继续处于Started状态；
  - 如果没有设置循环播放为true，那么播放完毕后，播放引擎内部会通过OnCompletion.onCompletion()通知应用开发者，该回调函数需要应用开发者提前调用 setOnCompletionListener(OnCompletionListener)函数设置给播放引擎；
  - 当处于PlaybackCompleted状态时，要想重新进入Started状态则必须先调用stop()，然后依次调用reset()、setDataSource()、prepareAsync()、start()。

- 由以上状态说明可知，播放引擎很多操作都是异步执行，内部的状态都是通过设置的回调函数异步通知到应用层，主要的回调函数接口见下表：

interface	ArcMediaPlayer.OnBufferingUpdateListener	当播放网络流媒体，需要进行数据缓冲时触发
interface	ArcMediaPlayer.OnCompletionListener	当媒体数据被播放结束时触发
interface	ArcMediaPlayer.OnErrorListener	当播放引擎内部在播放过程中出现致命错误播放无法继续时触发
interface	ArcMediaPlayer.OnInfoListener	当播放中一些必要的提示性信息需要通知应用层时触发；当播放过程中出现一些非致命错误需要通知应用层有用户来决策播放是否继续时触发
interface	ArcMediaPlayer.OnPreparedListener	当使用异步接口 prepareAsync 进行播放准备，播放引擎内部准备工作完成后触发
interface	ArcMediaPlayer.OnSeekCompleteListener	当 seek 操作完成后触发
interface	ArcMediaPlayer.OnVideoSizeChangedListener	当播放过程中视频宽高出现变化时触发

1.5. 主要接口调用时序图



1.6. 鉴权

在使用 SDK 功能前，要通过鉴权验证。

1.6.1.1. 鉴权逻辑

1、 安装 app 后，是无鉴权信息，需要去更新鉴权信息。但是第一次运行时，允许运行 SDK 功能。此时会更新鉴权信息，知道更新成功。在没更新成功前，属于试用阶段（比如试用 5 天）。可能由于网络原因，在试用 n 天内，一直无法更新鉴权信息，这个阶段会给予权限运行加水印方式；试用 n 天后，还是无更新成功，则直接禁止运行。

2、 成功更新鉴权信息后，一切鉴权结果根据最新鉴权信息查询得到。鉴权会定期去更新鉴权信息。

1.6.1.2. 调用方式

1、 首先设置鉴权需要的账号信息。

```
创建player
ArcMediaPlayer mMediaPlayer = new ArcMediaPlayer();
设置鉴权参数--必须
mMediaPlayer.validate(this, accessKey, secretKey, appKey);
```

必须调用 validate。

2、对鉴权结果做判断

```
mMediaPlayer.setDataSource(m_strURL,headers);
```

鉴权结果通过通知得到，鉴权结果一般分为如下三种：

允许：只有返回值是 0 才是这种状态，也是唯一，无任何通知。

禁止：禁止 SDK 运行，会通过 OnError 通知。

允许+水印：像这种状态属于提醒状态，让用户知道目前该 SDK 过期了、无鉴权信息，通过 OnInfo 通知。

第2章: API 参考手册

public class ArcMediaPlayer

2.1. 概述

本章主要对播放引擎的 API 功能和使用进行说明，其中包括使用中需要用到的一些常量定义的说明，为了方便开发者集成，在接口定义上参照了 Android 操作系统的 android.media.MediaPlayer，对于熟悉系统 MediaPlayer 的开发者集成没有任何困难。

2.2. 常量定义

2.2.1. 错误码定义

当出现如下定义的错误类型时，表示当前播放遇到了严重的 bug，播放将无法继续。此时当前播放需要关闭然后进行调查或者尝试重试播放。这些错误类型都是通过 OnErrorListener.onError()回调函数返回。

```
public static final int MEDIA_ERROR_SOURCE_UNSUPPORTED_SCHEME =100001
```
不支持的流媒体协议

```
public static final int MEDIA_ERROR_SOURCE_NETWORK_CONNECTFAIL = 100002
```
网络连接失败(建立TCP连接失败)

```
public static final int MEDIA_ERROR_SOURCE_STREAM_OPEN = 100003
```
打开流媒体失败

```
public static final int MEDIA_ERROR_SOURCE_STREAM_SEEK = 100004
```
Seek流媒体失败

```
public static final int MEDIA_ERROR_SOURCE_DATARECEIVE_TIMEOUT = 100005
```
数据接收超时(比如: 30秒未收到任何数据)

```
public static final int MEDIA_ERROR_SOURCE_FORMAT_UNSUPPORTED = 100006
```
文件格式不支持(比如flv)

```
public static final int MEDIA_ERROR_SOURCE_FORMAT_MALFORMED = 100007
```
播放列表文件格式不对(比如m3u8格式无法识别)

```
public static final int MEDIA_ERROR_SOURCE_DNS_RESOLVE = 100008
```
DNS解析错误

```
public static final int MEDIA_ERROR_SOURCE_DNS_RESOLVE_TIMEOUT = 100009
```
DNS解析超时

```
public static final int MEDIA_ERROR_SOURCE_NETWORK_CONNECTIMEOUT = 100010
```
网络连接超时(在一定长的时间内没有连接成功，也没有收到任何错误)

```
public static final int MEDIA_ERROR_SOURCE_DATARECEIVE_FAIL = 100011
```
数据接收错误(接收数据过程中收到了服务器报的错误)

```
public static final int MEDIA_ERROR_SOURCE_DATASEND_TIMEOUT = 100012
```
数据(网络请求)发送超时

```
public static final int MEDIA_ERROR_SOURCE_DATASEND_FAIL = 100013
```
数据（网络请求）发送失败

```
public static final int MEDIA_ERROR_SOURCE_DATAERROR_HTML = 100014
```
接收到的数据为html网页，而不是媒体数据，请求被劫持或者服务器返回鉴权页面之类

```
public static final int MEDIA_ERROR_SOURCE_BUFFER_TIMEOUT = 100015
```
表示缓冲超时，用来控制允许的缓冲等待时间，超时后停止播放。
假设timeoutVal为超时时间, 单位毫秒，所设值小于1秒时, 按1秒计
以进入缓冲状态为起始计时时间, 在所指定时间内如果未完成缓冲, 则触发超时通知并转换播放器状态为停止态, 以下情况例外:
1) timeoutVal时间内完成缓冲, 则取消超时
2) timeoutVal时间内APP调用seekTo接口进行跳转操作时, 重新计时
3) timeoutVal时间内发生其它严重错误(即onError抛出的消息)
4) timeoutVal时间内APP调用setConfig(BUFFERING_TIMEOUT_ID, 0)取消限时
5) timeoutVal时间内APP调用stop接口停止播放, 或者reset/release等接口释放播放器实例

```
public static final int MEDIA_ERROR_SOURCE_DATARECEIVE_NOBODY =100016
```
表示在请求数据的过程中，http服务器返回的数据为空

```
public static final int MEDIA_ERROR_SOURCE_SEEK_BEYONDFILESIZE =100017
```
表示seek的时间点超出了实际可播放的长度，实际可播放长度小于文件总长度的情况(比如flv可试看5分钟的场景，5分钟的实际可播放长度小于文件总长度)

```
常量值位于100400~100599区间内的定义表示为HTTP返回的状态码
```
HTTP服务器返回的错误值(比如100404表示收到了HTTP 404错误)

```
public static final int MEDIA_ERROR_PLAYER_DISPLAY_INIT_FAILED = 200001
```
渲染器创建失败

```
public static final int MEDIA_ERROR_PLAYER_NOAUDIO_VIDEOUNSUPPORT = 200002
```
媒体数据中没有audio同时video格式不支持

```
public static final int MEDIA_ERROR_PLAYER_NOVIDEO_AUDIOUNSUPPORT = 200003
```
媒体数据中没有video同时audio格式不支持

```
public static final int MEDIA_ERROR_PLAYER_AVCODEC_UNSUPPORT = 200004
```
媒体数据中有video和audio，但是Video和audio格式都不支持

```
public static final int MEDIA_ERROR_PLAYER_OPERATION_CANNOTEXECUTE = 200005
```
当前操作无法执行，比如player初始化失败，app调用play就会通知app该操作无法执行

```
public static final int MEDIA_ERROR_PLAYER_AVCODEC_AUDIOUNSUPPORT = 200006
```

媒体数据中有video和audio，但是audio格式不支持

```
public static final int MEDIA_ERROR_PLAYER_AVCODEC_VIDEOUNSUPPORT= 200007
```
媒体数据中有video和audio，但是Video格式不支持

```
public static final int LICENSE_ERR                                    = 8000
```
鉴权出错返回值识别标志，具体错误值查看参数extra，错误码的定义即2.2.5

2.2.2. 提示性信息定义

提示性信息用于通知调用者播放过程中的一些重要状态的变化，以便调用者进行一些友好性的处理，增强用户体验，这些信息通过OnInfoListener.onInfo()回调函数返回。

```
public static final int MEDIA_INFO_UNKNOWN = 1
```
未知的media info信息

```
public static final int MEDIA_INFO_BUFFERING_START = 701
```
开始缓冲数据，此时播放会临时暂停，extra参数表示进入缓冲的原因，0表示播放过程中，1表示刚开始播放时，2表示seek引起的缓冲。

```
public static final int MEDIA_INFO_BUFFERING_END = 702
```
数据缓冲结束，播放继续，extra参数表示进入缓冲的原因，0表示播放过程中，1表示刚开始播放时，2表示seek引起的缓冲。

```
public static final int MEDIA_INFO_NOT_SEEKABLE = 801
```
当前播放无法进行seek操作(比如直播)

```
public static final int MEDIA_INFO_RENDERING_START = 900
```
播放开始进行第一帧数据的渲染

```
public static final int MEDIA_INFO_BANDWIDTH_SWITCH_SAME  =   32801(0x8021)
```
要切换的码率或清晰度已经正在播放

```
public static final int MEDIA_INFO_BANDWIDTH_SWITCH_SUCCESS = 32802(0x8022)
```
切换成功

```
public static final int MEDIA_INFO_BANDWIDTH_SWITCH_FAILED  = 32803(0x8023)
```
切换失败

```
public static final int MEDIA_INFO_BANDWIDTH_SWITCH_INVALID  = 32804(0x8024)
```
要切换的码率或清晰度是个无效的值

2.2.3. 警示性信息定义

警示性信息用于通知调用者播放过程中出现了某种不期望出现的错误，但是这种错误不是致命的，不会导致播放引擎无法继续运行。这种情况下通过警示性信息通知调用者，由调用者自己来判断是否要停止播放或者采取其它处理措施。这些信息通过OnInfoListener.onInfo()回调函数返回。

```
public static final int MEDIA_INFO_SPLITTER_NOVIDEO = 32769(0x8001)
```
媒体数据中没有video

```
public static final int MEDIA_INFO_SPLITTER_NOAUDIO = 32770(0x8002)
```
媒体数据中没有audio

```
public static final int MEDIA_INFO_VCODEC_DECODE_ERROR = 12297(0x3009)
```
Video解码失败，audio正常

```
public static final int MEDIA_INFO_ACODEC_DECODE_ERROR = 12293(0x3005)
```
Audio解码失败，video正常

```
public static final int LICENSE_INFO                    = 8001
```
鉴权警示返回值识别标志，具体错误值查看参数extra，错误码的定义即2.2.5

2.2.4. Config Type

Config type 是用来对播放引擎进行一些特殊设置来启用一些附加的功能，使用 setConfig()函数进行参数设置。

```
public static final int CONFIG_NETWORK_CONNECT_TIMEOUT = 0x50000F2
```
设置网络连接超时时间，单位是秒

```
public static final int CONFIG_NETWORK_RECEIVE_TIMEOUT = 0x50000F3
```
设置数据接收超时时间，单位是秒

```
public static final int CONFIG_NETWORK_RECONNECT_COUNT = 0x50000F4
```
设置连接失败后的重试次数，比如设置1表示当网络连接失败后会自动重新尝试再连接1次

2.2.5. 鉴权返回值

以下带有DISABLE的是禁止运行,通知给OnError

```
public static final int LICENSE_ERR_DISABLE_APP_NAME              = 1;
```
appname不一致

```
public static final int LICENSE_ERR_DISABLE_AUTHENTICATE_FAIL      = 2
```
鉴权验证不通过

```
public static final int LICENSE_ERR_DISABLE_INVALID_PARAM           = 3;
```
鉴权账号信息无设置，或其中某个为空

```
public static final int LICENSE_ERR_DISABLE_PARAMETER_DISACCORD     = 4;
```
同app里存在多个SDK,但是鉴权的账号信息不一致

```
public static final int LICENSE_ERR_DISABLE_DIRECTORY_ERR           = 5;
```
目录错误（极少）

```
public static final int LICENSE_ERR_DISABLE_MEM_NOT_ENOUGH          = 6;
```
内存不足

```
public static final int LICENSE_ERR_DISABLE_SDK_PLANTFORM_NO_SUPPORT = 7;
```
平台不支持

```
public static final int LICENSE_ERR_DISABLE_SDK_NO_EXITS = 8;
```
无该SDK信息

```
public static final int LICENSE_ERR_DISABLE_NO_UPDATE = 11;
```
试用n天后，鉴权信息无更新（极少出现）

```
public static final int LICENSE_ERR_DISABLE_NETWORK = 12;
```
试用n天后，由于网络错误，鉴权信息没更新成功

```
public static final int LICENSE_ERR_DISABLE_DATA_FORMAT = 13;
```
试用 n 天后，成功更新鉴权信息，但是格式错误（极少出现）

以下带有ENABLE的是允许运行，但是带有水印,通知给OnInfo

```
public static final int LICENSE_INFO_ENABLE_NO_UPDATE = 21
```
试用n天内，鉴权信息无更新（极少出现）

```
public static final int LICENSE_INFO_ENABLE_NETWORK = 22;
```
试用n天内，由于网络错误，鉴权信息没更新成功

```
public static final int LICENSE_INFO_ENABLE_DATA_FORMAT = 23;
```
试用n天内，license数据格式错误，极少出现

```
public static final int LICENSE_INFO_ENABLE_SDK_EXPIREDATE = 31;
```
鉴权信息表示，该 SDK 已经过期 30 天

2.3. 构造函数

2.3.1. ArcMediaPlayer

原型

public ArcMediaPlayer()；

描述

默认构造函数，当使用完毕后调用者需要调用 release()函数来释放所有资源。如果不释放，当多个播放引擎对象被创建使用后可能会导致不可预测的错误。

2.4. 公共函数定义

2.4.1. validate

原型

Public void validate(Context context, String AccessKey, String AccessSecret, String AppKey)

描述

设置鉴权账号信息，且在调用 setDataSource 函数前必须调用该函数。

参数

context	Context	上下文
AccessKey	String	鉴权账号信息
AccessSecret	String	鉴权账号信息
AppKey	String	与鉴权账号信息对应且每个 app 对应唯一一个 ID

2.4.2. getCurrentPosition

原型

publicintgetCurrentPosition ()

描述

获取当前播放的位置，单位为毫秒.

对于直播流，则表示从播放开始到当前状态所经过的自然时间长度.

返回值

返回当前播放时间点，以毫秒为单位，

如果播放引擎状态不对则返回 0.

2.4.3. getDuration

原型

publicintgetDuration ()

描述

获取当前正在播放的媒体资源的总长度.

对于直播流，则返回 0.

返回值

返回当前播放的媒体文件的总长度，以毫秒为单位，
如果播放引擎状态不对则返回 0

2.4.4. getVideoHeight

原型

publicintgetVideoHeight ()

描述

获取当前播放的视频流的原始画面高，考虑到播放过程中视频的尺寸可能会发生变化建议开发者注册 OnVideoSizeChangedListener 监听函数用来接收视频尺寸变化的通知

返回值

返回值为视频尺寸的高

如果没有视频流数据或者没有设置用于显示的 surface，则会返回 0

2.4.5. getVideoWidth

原型

publicintgetVideoWidth ()

描述

获取当前播放视频流的原始画面宽，考虑到播放过程中视频的尺寸可能会发生变化建议开发者注册 OnVideoSizeChangedListener 监听函数用来接收视频尺寸变化的通知

返回值

返回值为视频尺寸的宽

如果没有视频流数据或者没有设置用于显示的 surface，则会返回 0.

2.4.6. isLooping

原型

publicbooleanisLooping ()

描述

查询当前播放引擎是否处于循环播放模式.

返回值

True---→表示为循环播放

False--→表示为非循环播放

2.4.7. isPlaying

原型

publicbooleanisPlaying ()

描述

查询播放引擎当前是否处于播放状态，即 Started 状态

返回值

True---→表示为播放状态，即正在播放

False--→表示为非播放状态

2.4.8. pause

原型

public void pause ()

描述

暂停播放，调用 start()来恢复播放，如果要结束播放则可调用 stop()，如果当前媒体数据不支持暂停操作则调用该函数不会产生任何结果(比如直播)。

异常上抛

IllegalStateException 播放引擎还没有被初始化(Initialized)时调用 pause

2.4.9. prepare

原型

public void prepare ()

描述

播放前的准备工作的同步接口，包括获取媒体信息，缓存少量数据，创建音视频解码器，只有在准备工作完成后才返回；调用时序上该接口需要在 setDataSource 和 setDisplay 接口之后，在 start 接口之前调用。

异常上抛

IllegalStateException　在非法的状态下调用(Idle\Initialized)

IOException

2.4.10. prepareAsync

原型

public void prepareAsync ()

描述

播放前准备工作的异步接口，包括获取媒体信息，缓存少量数据，创建音视频解码器，该接口会立即返回而不会等待准备工作完成后才返回；调用时序上该接口需要在 setDataSource 和 setDisplay 接口之后，在 start 接口之前调用。该接口可以避免出现因准备工作耗时较长而引起的主线程阻塞。

异常上抛

IllegalStateException　在非法的状态下调用(Idle\Initialized)

2.4.11. release

原型

public void release ()

描述

释放与播放引擎相关的所有资源，该接口调用后 player 将不能继续使用，所以这里 release 的使用建议是在 activity 销毁时调用(onDestroy)，如果在使用过程中调用了 release，那么当要继续使用 player 时，player 必须重新创建，同时 setConfigFile 和 validate 函数也必须重新调用，所以播放过程中在 activity 未退出情况下停止播放或者切流，只需要调用 stop 和 reset 即可。

2.4.12. reset

原型

public void reset ()

描述

重置播放引擎到 Idle 状态，调用该接口之后，必需从头开始进行所有的播放逻辑调用：setDataSource\prepareAsync 等。

2.4.13. seekTo

原型

public void seekTo (intmsec)

描述

跳转到指定的时间点进行播放，对于不支持 seek 功能的流，调用该接口不会产生任何结果(比如直播)。

参数

msec	int	要跳转到的目标播放时间点

异常上抛

IllegalStateException	在非法的状态下调用(Idle \Initialized\Preparing\Playbackcompleted\Stop\End\Error)

2.4.14. setDataSource

原型

public void setDataSource (String path)

描述

首先在内部做鉴权判断，鉴权通过才会设置要播放的媒体资源的路径(Url)。在调用该函数前必须先调用 validate 设置鉴权需要的账号信息。

参数

path	String	媒体资源的路径, 如果是本地视频 path 为视频文件路径, 如果是网络视频则为网络 URL

异常上抛

IllegalStateException	在非 Idle 状态调用均视为非法状态调用
IOException	
IllegalArgumentException	非法参数

鉴权通知

禁止状态，通过 OnError 里 LICENSE_ERR 标识识别，具体的禁止状态可由 extra 参数获取。

提醒状态，通过 OnInfo 里 LICENSE_ERR 标识识别，具体的禁止状态可由 extra 参数获取。

2.4.15. setDataSource

原型

public void setDataSource (String path, Map<String, String> headers)

描述

首先在内部做鉴权判断，鉴权通过才会设置要播放的媒体资源的路径(Url)，同时可附带 HTTP 头部信息。在调用该函数前必须先调用 validate 设置鉴权需要的账号信息。

参数

path	String	媒体资源的路径, 如果是本地视频 path 为视频文件路径, 如果是网络视频则为网络 URL
headers	Map<String, String>	HTTP 头部信息，当发起 http 请求时该信息会添加到 http 请求中

异常上抛

IllegalStateException 在非 Idle 状态调用均视为非法状态调用
IOException
IllegalArgumentException 非法参数

鉴权通知

禁止状态，通过 OnError 里 LICENSE_ERR 标识识别，具体的禁止状态可由 extra 参数获取。
提醒状态，通过 OnInfo 里 LICENSE_ERR 标识识别，具体的禁止状态可由 extra 参数获取。

2.4.16. setDataSource

原型

public void setDataSource (FileDescriptor fd, long offset, long length)

描述

首先在内部做鉴权判断，鉴权通过才会设置可用的视频文件数据描述器(FileDescriptor)，该视频文件数据描述器(FileDescriptor)必须可 seek，该视频文件数据描述器使用后需要由调用者负责关闭和释放。在调用该函数前必须先调用 validate 设置鉴权需要的账号信息。

参数

fd	FileDescriptor	要播放的视频文件数据描述器(FileDescriptor)
offset	long	播放的起始位置，即相对于开始位置的字节偏移量
length	long	要播放的数据长度，单位为字节

异常上抛

IllegalStateException 在非 Idle 状态调用均视为非法状态调用
IOException
IllegalArgumentException 非法参数

鉴权通知

禁止状态，通过 OnError 里 LICENSE_ERR 标识识别，具体的禁止状态可由 extra 参数获取。

提醒状态，通过 OnInfo 里 LICENSE_ERR 标识识别，具体的禁止状态可由 extra 参数获取。

2.4.17. setDataSource

原型

public void setDataSource (FileDescriptor fd)

描述

首先在内部做鉴权判断，鉴权通过才会设置可用的视频文件数据描述器(FileDescriptor)，该视频文件数据描述器(FileDescriptor)必须可 seek，该视频文件数据描述器使用后需要由调用者负责关闭和释放。在调用该函数前必须先调用 validate 设置鉴权需要的账号信息。

参数

fd	FileDescriptor	要播放的视频文件数据描述器(FileDescriptor)

异常上抛

IllegalStateException 在非 Idle 状态调用均视为非法状态调用

IOException

IllegalArgumentException 非法参数

鉴权通知

禁止状态，通过 OnError 里 LICENSE_ERR 标识识别，具体的禁止状态可由 extra 参数获取。

提醒状态，通过 OnInfo 里 LICENSE_ERR 标识识别，具体的禁止状态可由 extra 参数获取。

2.4.18. setDataSource

原型

public void setDataSource (Context context, Uri uri)

描述

首先在内部做鉴权判断，鉴权通过才会以内容 uri 的方式设置播放数据源。在调用该函数前必须先调用 validate 设置鉴权需要的账号信息。

参数

context	Context	处理内容 Uri 时需要用到的当前应用的上下文对象
uri	Uri	播放要使用的内容 Uri 对象实例

异常上抛

IllegalStateException 在非 Idle 状态调用均视为非法状态调用
IOException
IllegalArgumentException 非法参数

鉴权通知

禁止状态，通过 OnError 里 LICENSE_ERR 标识识别，具体的禁止状态可由 extra 参数获取。
提醒状态，通过 OnInfo 里 LICENSE_ERR 标识识别，具体的禁止状态可由 extra 参数获取。

2.4.19. setDataSource

原型

public void setDataSource(Context context, Uri uri, Map<String, String> headers)

描述

首先在内部做鉴权判断，鉴权通过才会以内容 uri 的方式设置播放数据源，同时可设置附加的 http 头部信息。在调用该函数前必须先调用 validate 设置鉴权需要的账号信息。

参数

context	Context	处理内容 Uri 时需要用到的当前应用的上下文对象
uri	Uri	播放要使用的内容 Uri 对象实例
headers	Map<String, String>	HTTP 头部信息，当发起 http 请求时该信息会添加到 http 请求中

异常上抛

IllegalStateException 在非 Idle 状态调用均视为非法状态调用
IOException
IllegalArgumentException 非法参数

鉴权通知

禁止状态，通过 OnError 里 LICENSE_ERR 标识识别，具体的禁止状态可由 extra 参数获取。
提醒状态，通过 OnInfo 里 LICENSE_ERR 标识识别，具体的禁止状态可由 extra 参数获取。

2.4.20. setDisplay

原型

public void setDisplay (SurfaceHoldersh)

描述

设置 SurfaceHolder，用来为视频画面提供显示窗口，该接口为可选接口，如果不调用该接口设置 SurfaceHolder，那么播放时将只有声音没有画面，如果设置了 SurfaceHolder，那么在 surface 失效时或者销毁时必须设置 setDisplay(null)

参数

sh	SurfaceHolder	用来显示视频画面的 SurfaceHolder 对象

2.4.21. setLooping

原型

public void setLooping (boolean looping)

描述

设置循环播放模式(开启或关闭)

参数

looping	boolean	true 表示启用循环播放，反之为 false

2.4.22. setOnCompletionListener

原型

public void setOnCompletionListener (com.arcvideo.MediaPlayer.ArcMediaPlayer.OnCompletionListener listener)

描述

注册一个回调函数，当数据源被播放完毕后会触发该回调通知调用者数据已播放完毕

参数

listener	com.arcvideo.MediaPlayer.ArcMediaPlayer.OnCompletionListener	回调函数对象

2.4.23. setOnErrorListener

原型

public void setOnErrorListener (com.arcvideo.MediaPlayer.ArcMediaPlayer.OnErrorListener listener)

描述

注册一个回调函数，当播放过程中出现致命错误时通知调用者

参数

listener	com.arcvideo.MediaPlayer.ArcMediaPlayer.OnErrorListener	回调函数对象

2.4.24. setOnInfoListener

原型

public void setOnInfoListener (com.arcvideo.MediaPlayer.ArcMediaPlayer.OnInfoListener listener)

描述

注册一个回调函数，当播放过程中的一些提示性信息需要通知调用者是触发

参数

listener	com.arcvideo.MediaPlayer.ArcMediaPlayer.OnInfoListener	回调函数对象

2.4.25. setOnPreparedListener

原型

public void setOnPreparedListener
(com.arcvideo.MediaPlayer.ArcMediaPlayer.OnPreparedListener listener)

描述

注册一个回调函数，当播放引擎到达 Prepared 状态时触发，该回调函数是与 prepareAsync 配对使用的

参数

listener	com.arcvideo.MediaPlayer.ArcMediaPlayer.OnPreparedListener	回调函数对象

2.4.26. setOnSeekCompleteListener

原型

public void setOnSeekCompleteListener
(com.arcvideo.MediaPlayer.ArcMediaPlayer.OnSeekCompleteListener listener)

描述

注册一个回调函数，当 Seek 操作完成时触发

参数

listener	com.arcvideo.MediaPlayer.ArcMediaPlayer.OnSeekCompleteListener	回调函数对

象

2.4.27. setOnVideoSizeChangedListener

原型

public void setOnVideoSizeChangedListener (com.arcvideo.MediaPlayer.ArcMediaPlayer.OnVideoSizeChangedListener listener)

描述

注册一个回调函数，当播放视频的宽高变化时触发，以便调用者及时了解视频宽高变化进而进行一些而外的操作，比如调整显示区域以适应新的视频宽高

参数

listener	com.arcvideo.MediaPlayer.ArcMediaPlayer.OnVideoSizeChangedListener	回调函数对象

2.4.28. setScreenOnWhilePlaying

原型

public void setScreenOnWhilePlaying (booleanscreenOn)

描述

用来通过控制 SurfaceHolder 实现视频播放过程中的屏幕能一直处于非休眠状态，改接口比 setWakeMode(Context, int)更好，因为它不需要进行权限开通即可使用，setWakeMode(Context, int)需要配置权限才能用

参数

screenOn	boolean	true 表示保持屏幕打开，反之则表示允许屏幕休眠

2.4.29. setVolume

原型

public void setVolume (float leftVolume, float rightVolume)

描述

设置音量

参数

leftVolume	float	左音量大小，范围[0,1]
rightVolume	float	右音量大小, 范围[0,1]

2.4.30. setWakeMode

原型

public void setWakeMode (Contextcontext, int mode)

描述

用来为播放设置唤醒锁，可以设定 CUP、屏幕、键盘等的各种保持唤醒的状态，该方法在播放引擎没有设置 SurfaceHolder 来渲染的情况下使用，如果设置了 SurfaceHolder 则建议使用更高级的 setScreenOnWhilePlaying(boolean)方法来配置唤醒状态。

setWakeMode 方法的使用需要配置 WAKE_LOCK 权限许可。

参数

context	Context	当前上下文
mode	int	唤醒模式，定义在 PowerManager 这个 final 类中

2.4.31. start

原型

public void start ()

描述

开始播放或者恢复播放(比如从暂停状态恢复播放)

异常上抛

IllegalStateException 在非法状态下调用

2.4.32. stop

原型

public void stop ()

描述

停止播放，任何情况下都可以停止播放，停止播放后如果要开始新的播放或再次播放必须先 reset，然后 setDataSource

2.4.33. getAspectRatio

原型

public float getAspectRatio()

描述

获取当前视频的宽高比率，该比率可用于调整显示区域的真实大小

当 onVideoSizeChanged()触发时调用该接口获取宽高比

返回值

宽高比率的值，为浮点数，当没有视频时返回 0.0f

2.4.34. getCurrentBufferingPercent

原型

public int getCurrentBufferingPercent()

描述

获取当前媒体数据的缓冲百分比

返回值

缓冲百分比的值，范围[0,100]

2.4.35. setConfigFile

原型

public void setConfigFile(Context context, String path)

描述

设置当前上下文和插件配置文件路径，该接口需要在任何其它接口之前被调用

参数

context	Context	当前上下文对象实例
path	String	配置文件路径, 如果传入 null, 播放器内部会启用一个默认值, 覆盖最常见的应用场景(包含对 HLS, MP4, FLV, H.264, AAC 的支持)

2.4.36. setUserInfo

原型

public void setUserInfo(intitemType, String itemValue)

描述

该接口作用是让 app 端能够设置 string 类型的配置信息给 SDK，具体的设置类型请参见下表，同时请注意每个设置项的设置时机说明。

参数

itemType	int	要设置的项对应的类型 type，已知类型请看如下表格
itemValue	String	要设置的项对应的值

ItemType	说明	备注
DRM_CUSTOM_ID	客户 id，是指企业客户 id，需要保证调用时序上在 **setDataSource 和 prepareAsync(prepare)之**间进行调用	MEDIAFILE.*DRM_CUSTOM_ID*
DRM_CONTENT_ID	内容 id，需要保证调用时序上在 **setDataSource 和 prepareAsync(prepare)之**间进行调用	MEDIAFILE.*DRM_CONTENT_ID*
CURRENT_APP_COLLECTDATA	由 app 收集的数据，需要将这些数据组合成 json 格式再调用该接口传给 SDK，需要保证调用时序上在 **setDataSource 之**后，**stop** 之前进行调用	PLAYER.*CURRENT_APP_COLLECTDATA*

2.4.37. getVersionInfo

原型

public String getVersionInfo()

描述

获取当前 SDK 的版本号信息，版本号位字符串，格式类似于:3.5.0.15347

返回值

版本号字符串

2.4.38. setBandwidthSwitchType

原型

public void setBandwidthSwitchType(ARC_BANDWIDTH_SWITCH_MODE switchType)

描述

设置码率切换(清晰度切换)的模式，如果不调用该接口则默认启用码率自动切换，通过计算网络带宽进行，如果需要调用接口则需要在 setDataSource 和 prepareAsync 两个接口之间进行调用。

参数

swtichType	ARC_BANDWIDTH_SWITCH_MODE	码率(分辨率)切换模式 type，有两种：SWITCHMODE_AUTO 和 SWITCHMODE_MANUAL

返回值

无

2.4.39. setCurrentBandwidthByIndex

原型

public void setCurrentBandwidthByIndex(int nIndex)

描述

设置要切换的码率 (清晰度)的流对应的索引 index, 设置后会立马进行切换，切换过程会有相应的 Info 信息通知到 App。

参数

nIndex	int	索引值

返回值

无

2.4.40. getBandwidthCount

原型

public int getBandwidthCount()

描述

获取当前可切换的多码率(多清晰度)流数量；

返回值

流的数量

2.4.41. setPlayerMode

原型

public void setPlayerMode(double playbackRate)

描述

设置倍速播放模式，该接口可设置快速、慢速播放，(0,1)范围表示慢速播放，(1, 8]范围表示快速播放，2 倍速及以下有声音，超过 2 倍速无声音。该接口的调用时机，在播放过程中进行调用，可动态设置不同的值并起效果。如果程序启动时就需要从代码中进行倍速设置，则设置时机为在收到 ArcMediaPlayer.MEDIA_INFO_RENDERING_START 消息通知时进行设置。

参数

playbackRate	double	倍速播放的倍速值

返回值

无

第3章: 集成说明

3.1. 概述

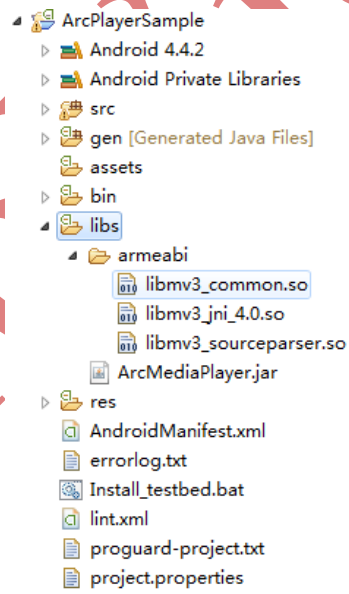
本章节主要是为了帮助开发者对 SDK 进行快速的集成，内容包括 jar 包和 so 库在开发者工程中的使用，app 权限和接口使用时序及注意事项等。

3.2. 集成步骤

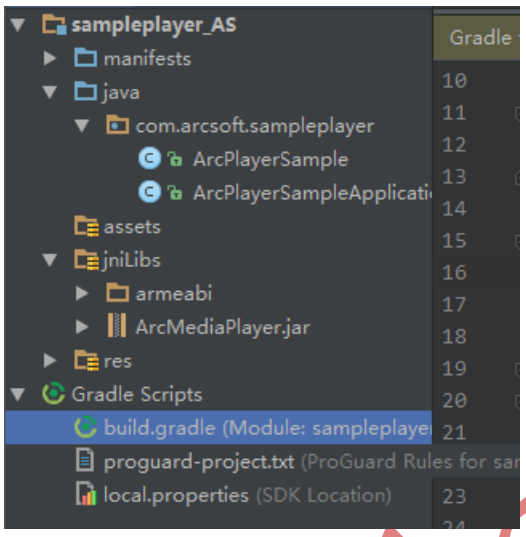
3.2.1. 工程创建及库的加载

开发者需要根据自己的需要创建 APP 的 android 工程，然后将播放器 SDK 中的相关 jar 包及 so 库加载到工程中，只有正确加载了这些库才能使用相应的功能。

1. Eclipse 开发工具加载库方法如下：
   a. Eclipse 工程创建好之后，会有如下目录结构，如果 libs 目录或者 armeabi 目录不存在则可以手动创建；



   b. 将 ArcMediaPlayer.jar 包拷贝到 libs 目录下，将所有的 so 库拷贝到 libs\armeabi\目录下即可。
2. Android Studio 加载库方法如下：
   a. Android Studio 工程创建好之后，目录结构与 Eclipse 类似，如果 libs 目录或者 armeabi 目录不存在则可以手动创建，下图中出现 jniLibs 而不是 libs 是由于 build.gradle 文件中进行了配置，将 libs 映射为 jniLibs 了；



b. 将 ArcMediaPlayer.jar 拷贝到 libs 目录下，将所有的 so 库拷贝到 libs\armeabi\目录下。

c. 对 build.gradle 文件进行配置，以便 android studio 工程能正确引用库，配置如下：

```
dependencies {
    compile fileTree(dir: 'libs', include: '*.jar')
}

android {
    compileSdkVersion 16
    buildToolsVersion "23.0.3"

    sourceSets {
        main {
            manifest.srcFile 'AndroidManifest.xml'
            java.srcDirs = ['src']
            resources.srcDirs = ['src']
            aidl.srcDirs = ['src']
            renderscript.srcDirs = ['src']
            res.srcDirs = ['res']
            assets.srcDirs = ['assets']
            jniLibs.srcDirs = ['libs']
        }

        // Move the tests to tests/java, tests/res, etc...
        instrumentTest.setRoot('tests')

        // Move the build types to build-types/<type>
        // For instance, build-types/debug/java, build-types/debug/AndroidManifest.xml, ...
        // This moves them out of them default location under src/<type>/... which would
        // conflict with src/ being used by the main source set.
        // Adding new build types or product flavors should be accompanied
        // by a similar customization.
        debug.setRoot('build-types/debug')
        release.setRoot('build-types/release')
    }
}
```

d. 由于 android studio 提供了比较灵活的库加载方法，开发者也可以从网上搜索其它加载方法，如下是两个参考：

加载 jar 包：http://blog.csdn.net/a739697044/article/details/25998619

加载 so 库：http://blog.csdn.net/aplixy/article/details/51592035

e. 本文档使用的是 android studio2.1.2，由于 android studio 还在持续更新和完善，如果本文档提供方法不再适用最新版本，我们会尽快更新文档，在此之前请开发者自行从网上查询最新方法。

f. 开发者如果要直接将 SDK 提供的 samplecode 工程直接导入到 android studio，可以参考如下网址：

http://www.cnblogs.com/ct2011/p/4183553.html

https://developer.android.google.cn/studio/intro/migrate.html?hl=zh-cn#android_studio

3.2.2. 集成代码指导

1. 首先需要开发者或公司在当虹云后台进行注册账号，然后按照指定的方式获取 AccessKey/Secret；
2. APP 需要给 SDK 开一些权限，权限配置文件为 AndroidManifest.xml，权限项如下：

```
<uses-permission android:name="android.permission.WRITE_SETTINGS" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.RAISED_THREAD_PRIORITY" />
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
<uses-permission android:name="android.permission.CHANGE_CONFIGURATION" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.READ_LOGS"/>
```

3. 在调用播放接口之前需要先进行一些必要 so 库的预加载和 ini 配置文件的生成操作，只有这一步准备好了，播放器接口才能顺利运行，可参考 samplecode 中的 ArcPlayerSampleApplication.java 中的 copyPlayerIni()和 LoadLibraray()函数：

4. 对播放SDK接口操作详细逻辑可参考samplecode中的ArcPlayerSample.java，该samplecode是一个完整的简单播放应用，除了调用ArcMediaPlayer的播放接口实现播放逻辑之外，还牵涉到一些错误处理，界面元素状态转换和控制，Activity的消息监听和处理，控件和屏幕点击事件处理，整个测试程序是按照一个功能完整的普通播放器逻辑来实现的。所以除了集成ArcMediaPlayer播放器接口之外还有很多其它代码，如果使用者只想知道ArcMediaPlayer播放器接口如何调用，则可参考如下几个函数中对于mMediaPlayer的使用即可：

   a. onCreate
   b. openFileStr
   c. mRefreshHandler.handleMessage
   d. onPrepared
   e. onVideoSizeChanged
   f. onInfo
   g. onError
   h. onSeekComplete
   i. onCompletion
   j. stopPlayback
5. ArcMediaPlayer 的主要调用接口和时序如下：
   a. 创建 player 对象： ArcMediaPlayer mMediaPlayer = new ArcMediaPlayer();
   b. 设置 context 和 configfile: mMediaPlayer.setConfigFile(context, configFile);
   c. 设置鉴权参数：mMediaPlayer.validate(this, accessKey, secretKey, appKey);
   d. 设置播放 url 之前调用：mMediaPlayer.reset();
   e. 设置播放地址 mMediaPlayer.setDataSource(m_strURL,headers); headers 可以为空

f. 设置各种监听 Listener：

mMediaPlayer.setOnCompletionListener(this);

mMediaPlayer.setOnPreparedListener(this);

mMediaPlayer.setOnVideoSizeChangedListener(this);

mMediaPlayer.setOnInfoListener(this);

mMediaPlayer.setOnErrorListener(this);

mMediaPlayer.setOnSeekCompleteListener(this);

g. 设置 SurfaceHolder 给播放器，用于播放显示：mMediaPlayer.setDisplay(SurfaceHolder)，如果没有 SurfaceHolder 则可以直接设置 Surface：mMediaPlayer.setSurface(Surface);

h. 开始播放准备(使用前面设置的 m_strURL 和其它参数进行播放准备工作)调用接口 mMediaPlayer.prepareAsync()，准备工作做完会通过 onPrepared 通知 APP;

i. 如果播放过程中需要 seek，则可调用：mMediaPlayer.seekTo(position), seek 成功后会通过 onSeekComplete 通知 APP；

j. 播放过程中如果出错会通过 onError 通知(比如：网络连接失败等等)，有提示信息通过 onInfo 通知(比如：缓冲开始、缓冲结束等等);

k. 播放结束通过 onCompletion 通知 App，App 收到消息后应该主动停止播放；

l. 停止播放调用 mMediaPlayer.stop()；

m. 进程退出时调用 mMediaPlayer.release()释放所有资源，调用该函数后下次播放必须全部重新来过。

第4章: 常见问题及故障排查

4.1. 功能说明

4.1.1. 如何获取缓冲状态，并设置和刷新缓冲百分比？

解答：当缓冲开始时 SDK 会通过 OnInfoListener 发出 MEDIA_INFO_BUFFERING_START 消息表示缓冲开始，当收到此消息后开发者可以启动一个循环，按照自己的要求每隔一定时间来 更新缓冲百分比，获取缓冲百分比的函数为 getCurrentBufferingPercent()，当缓冲到达 100%时表示缓冲结束，此时 SDK 会通过 OnInfoListener 发出 MEDIA_INFO_BUFFERING_END 消息 。

4.1.2. 如何设置连接超时，连接失败自动重连和接收数据超时？

解答：超时设置开发者不设置播放器内部也会有自己的默认值，如果开发者想自己更改默认设置，则可以参考如下示例代码进行设置，接口要求的时间单位为毫秒(ms):

```
int mConnectTimeout = 5;
int mReceiveTimeout = 10;
int mReconnectCount = 2;
//网络连接超时时间设置
mMediaPlayer.setConfig(ArcMediaPlayer.CONFIG_NETWORK_CONNECT_TIMEOUT, mConnectTimeout * 1000);
//网络数据接收超时时间设置
mMediaPlayer.setConfig(ArcMediaPlayer.CONFIG_NETWORK_RECEIVE_TIMEOUT, mReceiveTimeout * 1000);
//网络连接失败自动重连次数设置
mMediaPlayer.setConfig(ArcMediaPlayer.CONFIG_NETWORK_RECONNECT_COUNT, mReconnectCount);
```

4.1.3. 如何播放当虹云加密视频？

解答：当虹云加密视频的播放 App 需要做两件事：

1. 按照当虹云提供的方式获取 customer id 和 content id；
2. 将得到的 customer id 和 content id 设置给播放器，设置必须在 mMediaPlayer.prepareAsync()函数被调用之前进行，设置示例代码如下：

```
String customidString = "xxxxxx";
String contentidString = "xxxxxxssxxxxxxx";
mMediaPlayer.setUserInfo(MEDIAFILE.DRM_CUSTOM_ID, customidString);
mMediaPlayer.setUserInfo(MEDIAFILE.DRM_CONTENT_ID, contentidString);
```

4.2. 故障排查

4.2.1. 横竖屏切换画面显示区域不对如何调整？

分析：Android 横竖屏切换的屏幕调整是由 app 层直接进行，设置的步骤如下：

1. 在 AndroidManifest.xml 中配置 android:configChanges=”orientation| keyboardHidden|screenSize” 属性，这样横竖屏切换时才不会销毁 Activity;
2. 继承并实现 onConfigurationChanged 函数，横竖屏切换时系统会触发该函数通知应用 层；
3. 在 onConfigurationChanged 函数被触发时，重新设置屏幕宽高和显示宽高，具体设置 方法请参见 SDK 的 sampleCode。

4.2.2. 画面被拉伸怎么调整？

分析：画面被拉升通常是由于设置的 surface 显示宽高比与视频源的实际显示宽高比不一 致造成的，可以在 APP 层对显示宽高比进行正确的计算并设置给 surfaceView 来解决此问题 ， 方法是当 SDK 内部获取到真实 videoInfo 之后会通过 onVideoSizeChanged 事件将真实宽高 告诉 app,收到该消息后，app 层继续从 SDK 内部去获取 aspec-ratio，然后使用屏幕宽高， video 宽高和 aspec-ratio 这些参数来计算真正的显示宽高并设置给 surfaceView,具体的计算方法请参见 SDK 的 sampleCode 的 onVideoSizeChanged 函数.

4.2.3. 设置 surface 时收到 Invalid Surface detected 错误是什么原因？

分析：该错误是通过 Surface 自带的接口 mSurface.isValid()来查询 surface 状态是否可用，如果为 false，表示不可用即 invalid，这种情况下就会收到 Invalid Surface detected 错误消息，那么为什么会 invalid 呢，可能是：

1. Surface 刚被创建，还未初始化完成，如果使用的是 surfaceView，那么只有等到 surfaceCreated 被执行了以后才能有效；
2. Surface 已经被销毁了，却还在使用。

4.2.4. 使用 SDK 自带的测试程序播放正常，开发者集成之后播放黑屏有声音。

分析：首先检查接口调用时序，对于 surfaceHolder 的设置要求 setDisplay 接口必须在 setDataSource 之后和 prepareAsync（prepare）之前进行调用；如果还是黑屏，请进一步确认开发者工程下是否已经存在 libmv3_videorenderer_4.0.so 和 libmv3_videorenderer_4.2.so 库。

4.2.5. 在 Android5.0 以上 64 位设备出现无法播放问题

分析：通常是由于用户的 app 打包时配置了 arm64-v8a 的库文件，而我们的库是不进行 arm64 优化的，暂时没有什么优化的意义。这样如果设置了 arm64-v8a 文件夹，那么 apk 安装时就会默认只安装 arm64-v8a 目录下的库，丢掉其它的库，所以就导致我们的库没有被安装到设备中，从而导致播放时无法加载需要的库(即库找不到)；解决的办法是建议 app 不要使用 arm64-v8a 目录来存放库，直接使用 armeabi 即可。

4.2.6. new ArcMediaPlayer()时出现 Can't create handler inside thread that has not called Looper.prepare()

分析：原因是在 ArcMediaPlayer 中使用到了 Handler, 调用 handler 的方法前必须执行 Looper.prepare()才能正确的使用 Handler 处理消息。Looper 用于封装了 android 线程中的消息循环，默认情况下一个线程是不存在消息循环（message loop）的，需要调用 Looper.prepare()来给线程创建一个消息循环，调用 Looper.loop()来使消息循环起作用。那么为什么我们在 activity 中可以直接使用 handler 呢，因为 activity 的主线程是自己会去创建 Looper 消息队列的，所以在 Activity 中新建 Handler 时，不需要先调用 Looper.prepare()，综上所述如果要在用户自定义的线程中使用 handler，那么就必须在创建线程时手动创建 Looper。也可以使用 Looper.getMainLooper()直接使用主线程的 Looper，但是这样会对主线程的运行有一定的影响。

4.2.7. 出现 java.lang.UnstatisfiedLinkError 错误如何定位

分析： java.lang.UnstatisfiedLinkError 错误的原因是 App 调用某个 JNI 接口来调用 native 的功能，但是这个 JNI 接口不存在。不存在的原因通常是如下几种之一：

a. 含有该接口的 so 库没有被加载，一般通过 java 的 System.loadLibrary 函数进行加载；

b. 含有该接口的 so 库加载失败，可以通过 java.lang.UnstatisfiedLinkError 类型的 exception 的如下接口 ex.getMessage()来获取失败的具体原因，可能是该 so 库并没有编译到 apk 中；

c. 含有该接口的 so 库有更新，且更新后接口定义有变化导致按照原来的接口调用失败。

```
AndroidRuntime   FATAL EXCEPTION: main
AndroidRuntime   Process: com.arcvideo.sampleplayer, PID: 5878
AndroidRuntime   java.lang.UnsatisfiedLinkError: No implementation found for void com.arcvideo □
                 .MediaPlayer.ArcMediaPlayer.native_setup(java.lang.Object) (tried Java_com_ar □
                 cvideo_MediaPlayer_ArcMediaPlayer_native_1setup and Java_com_arcvideo_MediaPl □
                 ayer_ArcMediaPlayer_native_1setup__Ljava_lang_Object_2)
AndroidRuntime   at com.arcvideo.MediaPlayer.ArcMediaPlayer.native_setup(Native Method)
```

附录一：文档修订记录

Version	Date	Modifier	Summary of changes
1.0.0	09/08/2015	文志平	初始版本
1.0.1	11/04/2015	文志平	增加设置 customID 和 contentID 的接口
1.0.2	14/03/2016	文志平	增加设置由 app 收集的用户数据的接口说明
1.0.3	10/08/2016	裘昊	增加对 setConfigFile 方法的补充说明
1.0.4	09/02/2017	文志平	增加集成说和常见问题章节以便开发者集成
1.0.5	08/03/2017	文志平	将包名由 com.arcsoft 改为 com.arcvideo
1.0.6	07/09/2017	文志平	增加获取版本号的接口说明
1.0.7	01/11/2018	文志平	增加多码率切换相关接口及消息定义
1.0.8	03/19/2018	文志平	增加倍速播放接口说明