

Ascend 310

HiAI Media API 参考

文档版本

01

发布日期

2019-03-12



版权所有 © 华为技术有限公司 2019。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



HUAWEI和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址：深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址：<http://www.huawei.com>

客户服务邮箱：support@huawei.com

客户服务电话：4008302118

目录

1 简介	1
2 媒体库接口	2
2.1 MediaLibInit	2
2.2 IsChipAlive	3
2.3 QueryCameraIds	4
2.4 QueryCameraStatus	4
2.5 OpenCamera	5
2.6 SetCameraProperty	6
2.7 GetCameraProperty	6
2.8 CapCamera	7
2.9 ReadFrameFromCamera	8
2.10 CloseCamera	9
2.11 QueryMICStatus	10
2.12 OpenMIC	10
2.13 SetMICProperty	11
2.14 GetMICProperty	12
2.15 CapMIC	12
2.16 ReadMicSound	13
2.17 CloseMIC	14
3 附录	16
3.1 示例	16

1 简介

HiAI Media是一套帮助开发者轻松获取摄像头图像和音频数据的API接口媒体库，该媒体库运行在Atlas DK开发者板上Ascend 310芯片操作系统的应用层。用户只需要通过简单的API函数调用，就可以方便地获取各种格式的视频流和音频流数据。通过这些API接口，用户也可以方便地控制Atlas DK开发板上的摄像头和MIC设备的工作模式，查询它们的工作状态及其支持的各种工作参数。

HiAI Media具体功能如下：

1. 初始化媒体库。
2. 控制媒体设备，包括打开/关闭摄像头和麦克风，设置工作参数，如图像的帧率、分辨率、声音的采样率等。
3. 查询媒体设备支持的工作模式和当前运行状态，例如支持的分辨率，摄像头在线状态等。
4. 采集音频和视频数据流，并将这些原始音视频数据按照用户自己需要的算法进行后续处理。

2 媒体库接口

HiAI Media的API接口是基于C语言的接口，下面对各个接口进行详细介绍。

接口的定义在\$HOME/tools/che/ddk/ddk/include/inc/driver/peripheral_api.h文件中。

[2.1 MediaLibInit](#)

[2.2 IsChipAlive](#)

[2.3 QueryCameraIds](#)

[2.4 QueryCameraStatus](#)

[2.5 OpenCamera](#)

[2.6 SetCameraProperty](#)

[2.7 GetCameraProperty](#)

[2.8 CapCamera](#)

[2.9 ReadFrameFromCamera](#)

[2.10 CloseCamera](#)

[2.11 QueryMICStatus](#)

[2.12 OpenMIC](#)

[2.13 SetMICProperty](#)

[2.14 GetMICProperty](#)

[2.15 CapMIC](#)

[2.16 ReadMicSound](#)

[2.17 CloseMIC](#)

2.1 MediaLibInit

初始化MediaLib库内部状态，用户在调用媒体库函数之前，首先应该调用该函数执行库的内部状态初始化。

函数格式

int MediaLibInit()

参数说明

参数	说明	取值范围
-	-	-

返回值

返回的错误码请参见[错误码示例](#)中的“错误码”列。

错误码示例

真值	错误码	错误码描述
0	LIBMEDIA_STATUS_FAILED	初始化媒体库失败
1	LIBMEDIA_STATUS_OK	初始化媒体库成功

2.2 IsChipAlive

查询Atlas DK开发者板上与Ascend 310对接的Hi3559芯片的当前连接状态是否正常。

函数格式

int IsChipAlive(char* chipName)

参数说明

参数	说明	取值范围
chipName	与Ascend 310对接的芯片名称。目前只支持hi3559	“3559”

返回值

返回的错误码请参见[错误码示例](#)中的“错误码”列。

错误码示例

真值	错误码	错误码描述
0	LIBMEDIA_STATUS_FAILED	与3559对接失败

真值	错误码	错误码描述
1	LIBMEDIA_STATUS_OK	与3559对接正常

2.3 QueryCameraIds

查询Atlas DK开发者板上当前可用的摄像头通道ID信息，目前开发者板上最多可同时支持两路摄像头。

函数格式

```
uint32_t QueryCameraIds(int* cameraIds, uint32_t *count)
```

参数说明

参数	说明	取值范围
cameraIds	用于保存查询结果的数组。	用户需要提供至少为的 2*sizeof(int)数组用来保存 返回结果。 说明 如果用户提供数组的大小超过实际的摄像头数量，函数会将无效的ID设置为-1。有效的ID是[0,1]。
count	指示cameraIds数组空间能够容纳ID的个数	[1,2]

返回值

返回的错误码请参见[错误码示例](#)中的“错误码”列。

错误码示例

真值	错误码	错误码描述
0	LIBMEDIA_STATUS_FAILED	查询失败
1	LIBMEDIA_STATUS_OK	查询成功

2.4 QueryCameraStatus

查询指定ID的摄像头的当前工作状态。

函数格式

```
enum CameraStatus QueryCameraStatus(int cameraId)
```

参数说明

参数	说明	取值范围
cameraId	指定的camera ID。	[0, 1]

返回值

返回的错误码请参见[错误码示例](#)中的“错误码”列。

错误码示例

真值	错误码	错误码描述
1	CAMERA_STATUS_OPEN	摄像头处于打开状态
2	CAMERA_STATUS_CLOSED	摄像头处于关闭状态
3	CAMERA_NOT_EXISTS	该摄像头不存在
4	CAMERA_STATUS_UNKOWN	摄像头状态未知

2.5 OpenCamera

打开指定的摄像头，在执行后续摄像头相关工作之前，需要先打开摄像头。

函数格式

```
int OpenCamera(int cameraId)
```

参数说明

参数	说明	取值范围
cameraId	指定的camera ID。	[0, 1]

返回值

返回的错误码请参见[错误码示例](#)中的“错误码”列。

错误码示例

真值	错误码	错误码描述
0	LIBMEDIA_STATUS_FAILED	打开摄像头失败
1	LIBMEDIA_STATUS_OK	打开摄像头成功

2.6 SetCameraProperty

设置和修改指定摄像头的属性参数。

函数格式

```
int SetCameraProperty(int cameraId, enum CameraProperties prop, const void* pInValue)
```

参数说明

参数	说明	取值范围
cameraId	指定的camera ID。	[0, 1]
prop	摄像头属性类型	仅支持以下描述的属性： enum CameraProperties { CAMERA_PROP_RESOLUTION =1, //分辨率, 数据类型为CameraResolution*, 长度为1 [Read/Write] CAMERA_PROP_FPS =2, //帧率, 数据类型为uint32_t, [Read/Write] CAMERA_PROP_CAP_MODE =5, //帧数据获取的方式：主动或者被动, 数据类型为CameraCapMode [Read/Write] };
pInValue	各个属性的值	参见上面说明，每个属性有自己的结构体，注意传入的数据类型

返回值

返回的错误码请参见[错误码示例](#)中的“错误码”列。

错误码示例

真值	错误码	错误码描述
0	LIBMEDIA_STATUS_FAILED	设置摄像头属性失败
1	LIBMEDIA_STATUS_OK	设置摄像头属性成功

2.7 GetCameraProperty

获取指定摄像头的当前指定属性的值。

函数格式

```
int GetCameraProperty(int cameraId, enum CameraProperties prop, void* pValue)
```

参数说明

参数	说明	取值范围
cameraId	指定的camera ID。	0~1
prop	摄像头属性类型	仅支持以下描述的属性： enum CameraProperties { CAMERA_PROP_RESOLUTION =1, //分辨率, 数据类型为CameraResolution*, 长度为1 [Read/Write] CAMERA_PROP_FPS =2, //帧率, 数据类型为uint32_t, [Read/Write] CAMERA_PROP_SUPPORTED_RESOLUTION =4, //用于获取摄像头支持的所有的分辨率列表, 数据类型为CameraResolution*, 数组长度为HIAI_MAX_CAMERARESOLUTION_COUNT [Read] CAMERA_PROP_CAP_MODE =5, //帧数据获取的方式: 主动或者被动, 数据类型为CameraCapMode [Read/Write] };
pInValue	各个属性的值	参见上面说明, 每个属性有自己的结构体, 注意传入的数据类型

返回值

返回的错误码请参见[错误码示例](#)中的“错误码”列。

错误码示例

真值	错误码	错误码描述
0	LIBMEDIA_STATUS_FAILED	获取摄像头属性失败
1	LIBMEDIA_STATUS_OK	获取摄像头属性成功

2.8 CapCamera

使用回调函数的方式捕获摄像头图像数据。在该工作模式下, 用户需要提供一个回调函数给媒体库使用, 该回调函数在底层产生图像的时候, 会被媒体库底层线程调用。建议用户在回调函数中将图像数据copy到自己的buffer中进行后续处理。

函数格式

```
int CapCamera(int cameraId, CAP_CAMERA_CALLBACK , void* param)
```

其中回调函数的格式如下：

```
typedef int (*CAP_CAMERA_CALLBACK) (const void* pdata, int size, void* param)
```

参数说明

参数	说明	取值范围
cameraId	指定的camera ID。	[0, 1]
CAP_CAMERA_CALLBACK	用户用于接收数据的回调函数	有效的用户回调函数地址
param	用户自定义的回调参数	媒体库接收到图像数据后会将该参数在回调函数中回传给用户。
pdata	一帧图像数据的地址	这块buffer由媒体库维护，对用户只读。用户不可以自行释放该地址指向的内存，否则行为不可预期。
size	图像数据的大小	根据图像格式、分辨率而变化
param	用户自定义的回调参数	媒体库接收到图像数据后会将该参数在回调函数中回传给用户。

返回值

返回的错误码请参见[错误码示例](#)中的“错误码”列。

错误码示例

真值	错误码	错误码描述
0	LIBMEDIA_STATUS_FAILED	捕获图像数据失败
1	LIBMEDIA_STATUS_OK	捕获图像数据成功

2.9 ReadFrameFromCamera

在用户自己的线程中主动读取摄像头返回的图像数据，注意该函数为阻塞调用，直到收到图像数据之后才会返回。用户负责提供用于接收图像数据的缓冲区，媒体库在接收到一帧完整图像数据之后，会将该帧完整图像数据copy到用户提供的缓冲区。

函数格式

```
int ReadFrameFromCamera(int cameraId, void* pdata, int* size)
```

参数说明

参数	说明	取值范围
cameraId	指定的camera ID。	[0, 1]
pdata	用户提供的缓冲区指针	-
size	用户提供的缓冲区大小，以字节为单位	用户需要根据设定的图像格式和分辨率事先计算好缓冲区大小，以确保缓冲区大小足够容纳输出的图像数据

返回值

返回的错误码请参见[错误码示例](#)中的“错误码”列。

错误码示例

真值	错误码	错误码描述
0	LIBMEDIA_STATUS_FAILED	捕获图像数据失败
1	LIBMEDIA_STATUS_OK	捕获图像数据成功

2.10 CloseCamera

关闭指定的摄像头，停止图像捕捉。

函数格式

```
int CloseCamera(int cameraId)
```

参数说明

参数	说明	取值范围
cameraId	指定的camera ID。	[0, 1]

返回值

返回的错误码请参见[错误码示例](#)中的“错误码”列。

错误码示例

真值	错误码	错误码描述
0	LIBMEDIA_STATUS_FAILED	关闭摄像头失败
1	LIBMEDIA_STATUS_OK	关闭摄像头成功

2.11 QueryMICStatus

查询Atlas DK开发者板上MIC麦克风的当前工作状态。

函数格式

```
enum MICStatus QueryMICStatus()
```

参数说明

参数	说明	取值范围
-	-	-

返回值

返回的错误码请参见[错误码示例](#)中的“错误码”列。

错误码示例

真值	错误码	错误码描述
1	MIC_STATUS_OPEN	MIC处于打开状态
2	MIC_STATUS_CLOSED	MIC处于关闭状态
3	MIC_NOT_EXISTS	MIC不存在
4	MIC_STATUS_UNKOWN	MIC状态未知

2.12 OpenMIC

打开Atlas DK开发者板上的麦克风，单板上内置“硅麦克风”，具体位置请查看产品使用手册文档。用户如果需要录音，请尽量把声源靠近硅麦后再开始录制。

函数格式

```
int OpenMIC()
```

参数说明

参数	说明	取值范围
-	-	-

返回值

返回的错误码请参见[错误码示例](#)中的“错误码”列。

错误码示例

真值	错误码	错误码描述
0	LIBMEDIA_STATUS_FAILED	打开麦克风失败
1	LIBMEDIA_STATUS_OK	打开麦克风成功

2.13 SetMICProperty

设置麦克风相关属性和工作参数。

函数格式

```
int SetMICProperty(struct MICProperties *propties)
```

参数说明

参数	说明	取值范围
propties	麦克风属性数据结构	struct MICProperties { enum AudioSampleRate sample_rate; 【Read/Write】 采样率, 数据类型为uint32_t enum MICCapMode cap_mode; 【Read/Write】 MIC捕获声音的模式 enum AudioBitWidth bit_width; 【Read/Write】 每个样本的bit位宽 enum AudioSampleNumPerFrame frame_sample_rate; 【Read/Write】 每帧的样本数量 enum AudioMode sound_mode; 【Read/Write】 单声道还是立体声. };

返回值

返回的错误码请参见[错误码示例](#)中的“错误码”列。

错误码示例

真值	错误码	错误码描述
0	LIBMEDIA_STATUS_FAILED	设置麦克风属性失败
1	LIBMEDIA_STATUS_OK	设置麦克风属性成功

2.14 GetMICProperty

获取麦克风的当前工作属性值。

函数格式

```
int GetMICProperty(struct MICProperties *propties)
```

参数说明

参数	说明	取值范围
propties	用户提供的属性数据结构指针。	参见 2.13 SetMICProperty 中的属性取值说明

返回值

返回的错误码请参见[错误码示例](#)中的“错误码”列。

错误码示例

真值	错误码	错误码描述
0	LIBMEDIA_STATUS_FAILED	获取麦克风属性失败
1	LIBMEDIA_STATUS_OK	获取麦克风属性成功

2.15 CapMIC

使用回调函数方式采集音频数据，用户需要按照如下形式定义自己的回调函数。在该工作模式下，该回调函数在底层产生一帧音频数据的时候，会被媒体库底层线程调用。建议用户在回调函数中将音频数据copy到自己的buffer中进行后续处理。

函数格式

```
int CapMIC(CAP_MIC_CALLBACK, void* param)
```

回调函数形式如下：

```
typedef int (*CAP_MIC_CALLBACK) (const void* pdata, int size, void* param)
```

参数说明

参数	说明	取值范围
CAP_CAMERA_CALLBACK	用户用于接收数据的回调函数	有效的用户回调函数地址
param	用户自定义的回调参数	媒体库接收到音频数据后会将该参数在回调函数中回传给用户。
pdata	一帧音频数据的地址	这块buffer由媒体库维护，对用户只读。用户不可以自行释放该地址指向的内存，否则行为不可预期。
size	音频数据的大小	根据音频格式、采样率而变化
param	用户自定义的回调参数	媒体库接收到音频数据后会将该参数在回调函数中回传给用户。

返回值

返回的错误码请参见[错误码示例](#)中的“错误码”列。

错误码示例

真值	错误码	错误码描述
0	LIBMEDIA_STATUS_FAILED	采集麦克风音频失败
1	LIBMEDIA_STATUS_OK	采集麦克风音频成功

2.16 ReadMicSound

在用户自己的线程中主动读取MIC采集的音频数据，注意该函数为阻塞调用，直到收到音频数据之后才会返回。用户负责提供用于接收音频数据的缓冲区，媒体库在接收到一帧完整音频数据之后，会将该帧数据copy到用户提供的缓冲区。

函数格式

```
int ReadMicSound(void* pdata, int *size)
```


参数说明

参数	说明	取值范围
pdata	用户提供的缓冲区指针	-
size	用户提供的缓冲区大小，以字节为单位	用户需要根据设定的音频格式和采样率事先计算好缓冲区大小，以确保缓冲区大小足够容纳输出的音频数据

返回值

返回的错误码请参见[错误码示例](#)中的“错误码”列。

错误码示例

真值	错误码	错误码描述
0	LIBMEDIA_STATUS_FAILED	采集麦克风音频失败
1	LIBMEDIA_STATUS_OK	采集麦克风音频成功

2.17 CloseMIC

关闭麦克风。

函数格式

```
int CloseMIC()
```

参数说明

参数	说明	取值范围
-	-	-

返回值

返回的错误码请参见[错误码示例](#)中的“错误码”列。

错误码示例

真值	错误码	错误码描述
0	LIBMEDIA_STATUS_FAILED	关闭麦克风失败

真值	错误码	错误码描述
1	LIBMEDIA_STATUS_OK	关闭麦克风成功

3 附录

基于本文描述的API接口，给出了一个使用上述接口的完整例程，该例程给出打开摄像头和MIC，修改和查询它们的属性参数，调用视频和音频数据采集接口，并把音视频数据流简单保存到文件中，最后关闭设备的完整过程。用户可参考这个例子代码来熟悉API的使用，然后根据实际项目的需要灵活使用这些接口。

3.1 示例

3.1 示例

```
#include "stdio.h"
#include "stdlib.h"
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "pthread.h"
#include <unistd.h>
#include "peripheral_api.h"

int Camera0UserFunc(const void* pdata, int size, void* param)
{
    // pdata为指向图像内存的首地址指针，建议用户可以直接读取，但是不建议修改内容，如果需要修改
    // 建议先拷贝到用户自定义的内存空间内再修改
    printf("Camera0UserFunc Get frame size = %d\r\n", size);
    return 1;
}

int Camera1UserFunc(const void* pdata, int size, void* param)
{
    // pdata为指向图像内存的首地址指针，建议用户可以直接读取，但是不建议修改内容，如果需要修改
    // 建议先拷贝到用户自定义的内存空间内再修改
    printf("Camera1UserFunc Get frame size = %d\r\n", size);
    return 1;
}

int main()
{
    int ret;
    int para, i;
    unsigned char *framebuffer;
    int framebuffer_size;
    struct CameraResolution resolution;
    struct MICProperties micprop;
    struct CameraResolution supported_resolution[HIAI_MAX_CAMERARESOLUTION_COUNT];
```

```
// 媒体库初始化
ret = MediaLibInit();
if(LIBMEDIA_STATUS_OK != ret) {
    printf("MediaLibInit failed %d\r\n", ret);
    return -1;
}
printf("MediaLibInit success %d\r\n", ret);

ret = IsChipAlive(NULL);
if(LIBMEDIA_STATUS_OK != ret) {
    printf("success failed %d\r\n", ret);
    return -1;
}
printf("IsChipAlive success ret %d\r\n", ret);

// 打开摄像头
ret = OpenCamera(0);
if(LIBMEDIA_STATUS_OK != ret) {
    printf("OpenCamera 0 failed %d\r\n", ret);
    return -1;
}
ret = OpenCamera(1);
if(LIBMEDIA_STATUS_OK != ret) {
    printf("OpenCamera 0 failed %d\r\n", ret);
    return -1;
}

ret = GetCameraProperty(0, CAMERA_PROP_SUPPORTED_RESOLUTION, supported_resolution);
if(LIBMEDIA_STATUS_OK != ret) {
    printf("GetCameraProperty 0 failed %d\r\n", ret);
    return -1;
}
i = 0;
do{
    printf("Camera0 supported width = %d, height = %d\r\n", supported_resolution[i].width,
supported_resolution[i].height);
    i++;
}while(supported_resolution[i].width != -1);

ret = GetCameraProperty(1, CAMERA_PROP_SUPPORTED_RESOLUTION, supported_resolution);
if(LIBMEDIA_STATUS_OK != ret) {
    printf("GetCameraProperty 1 failed %d\r\n", ret);
    return -1;
}
i = 0;
do{
    printf("Camera1 supported width = %d, height = %d\r\n", supported_resolution[i].width,
supported_resolution[i].height);
    i++;
}while(supported_resolution[i].width != -1);

// 帧率设置
para = 20;
ret = SetCameraProperty(0, CAMERA_PROP_FPS, &para);
if(LIBMEDIA_STATUS_OK != ret) {
    printf("SetCameraProperty 0 failed %d\r\n", ret);
    return -1;
}
ret = SetCameraProperty(1, CAMERA_PROP_FPS, &para);
if(LIBMEDIA_STATUS_OK != ret) {
    printf("SetCameraProperty 0 failed %d\r\n", ret);
    return -1;
}

// 分辨率设置
resolution.height = 1080;
resolution.width = 1920;
ret = SetCameraProperty(0, CAMERA_PROP_RESOLUTION, &resolution);
if(LIBMEDIA_STATUS_OK != ret) {
```

```
        printf("SetCameraProperty 0 failed %d\r\n", ret);
        return -1;
    }
    ret = SetCameraProperty(1, CAMERA_PROP_RESOLUTION, &resolution);
    if(LIBMEDIA_STATUS_OK != ret) {
        printf("SetCameraProperty 1 failed %d\r\n", ret);
        return -1;
    }

    // 注册用户回调函数
    ret = CapCamera(0, Camera0UserFunc, NULL);
    if(LIBMEDIA_STATUS_OK != ret) {
        printf("CapCamera 0 failed %d\r\n", ret);
        return -1;
    }
    ret = CapCamera(1, Camera1UserFunc, NULL);
    if(LIBMEDIA_STATUS_OK != ret) {
        printf("CapCamera 1 failed %d\r\n", ret);
        return -1;
    }

    // 睡眠5秒模拟用户使用流程
    sleep(5);
    // 关闭摄像头
    ret = CloseCamera(0);
    if(LIBMEDIA_STATUS_OK != ret) {
        printf("CloseCamera 0 failed %d\r\n", ret);
        return -1;
    }
    printf("CloseCamera 1 success %d\r\n", ret);
    ret = CloseCamera(1);
    if(LIBMEDIA_STATUS_OK != ret) {
        printf("CloseCamera 1 failed %d\r\n", ret);
        return -1;
    }
    printf("CloseCamera 0 success %d\r\n", ret);

    return 0;
}
```