**API接口说明**

|  |  |  |
| --- | --- | --- |
| **版本** | **时间** | **修改** |
| V1.0 | 2021/08/29 |  |
| V1.1 | 2021/09/13 |  |
| V1.2 | 2021/12/28 |  |

1. 概述

SDK是一套基于DLL（ Microsoft Windows Dynamic Link Library）的软件组件，提供UVC访问硬件设备和上层功能软件的集成接口，该 DLL 组件的输出接口符合 winapi调用规范，可以被 C++ 高级编程语言所调用。

SDK 内部实现了UVC设备的连接初始化、图像采集、图像校正等功能，同时也提供了设备及控制的参数设置窗口。用户软件系统通过调用相关的函数可以快捷地访问和控制uvc硬件设备。

SDK 的运行需要 Windows XP/Windows 7/Windows 8/windows 10 等 Windows 系列操作系统的支持。

本文档将重点讲解 SDK 软件接口，同时提供简短的接口调用的示例代码和功能实现  
流程。

2. SDK开发包

SDK 软件开发包包含开发所需的接口代码文件、 布署所需的 DLL 文件。

SDK\doc 帮助文档

SDK\dev\include c++开发接口头文件

SDK\dev\lib\32 32位C++静态连接库

SDK\dev\lib\64 64位C++静态连接 库

SDK\dev\dll\32 32位dll文件

SDK\dev\dll\64 64位dll文件

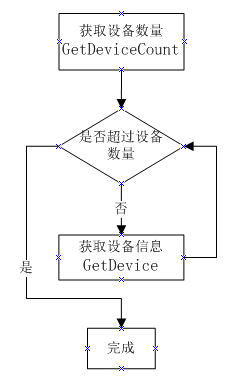
SDK\examples 示例程序

**3. SDK** 集成指南  
3.1 引用接口封装代码  
由于 SDK 的实现采用符合 WINAPI 调用规范的 DLL， 用户可以直接引用头文件和lib dll,在visual studio下方便快捷的二次开发。

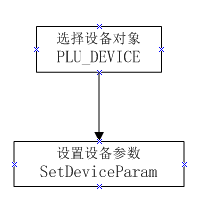
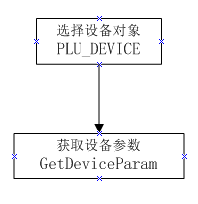
3.2 功能流程

遍历设备，获取设备对象流程，是可以共用的。在硬件没有新的拨下，插上的情况下，获取设备信息的流程是通用的。

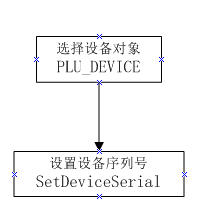
1. 遍历设备，获取设备对象流程。



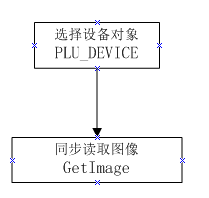
2. 获取或者是设备设备参数流程



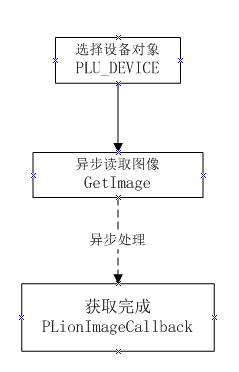
3. 写入设备序列号流程



4. 同步读取图像流程



5. 异步读取图像流程



6. 遍历设备示例代码:

vector<LU\_DEVICE> listDevice;

int nDev = GetDeviceCount(); //遍历UVC

for(int d = 0; d < nDev; d++) //遍历设备

{

PLU\_DEVICE pDev = GetDevice(d);

listDevice.push\_back(\*pDev);

}

7. 获取，设置设备参数示例代码:

//参数对象

LU\_PARAM param;

int nMode = 0;

memset(&param, 0, sizeof(param));

param.param = LUDEVPARAM\_MODE; //模式参数

param.size = sizeof(nMode);

param.data = &nMode;

//获取当前的模式

LU\_RESULT lr = GetDeviceParam(pDevice, &param);

if(lr == LU\_SUCCESS)

{

//获取成功

}

//设置模式参数

nMode = LUMODE\_UTC;

param.param = LUDEVPARAM\_MODE;

param.size = sizeof(nMode);

param.data = &nMode;

lr = SetDeviceParam(pDevice, &param);

if(lr == LU\_SUCCESS)

{

//设置成功

}

8. 设置序列号示例代码:

//测试序列号

char cSerial[64] = "test1234";

//设置序列号

LU\_RESULT nRet = SetDeviceSerial(pDevice, cSerial);

if(LU\_SUCCESS == nRet)

{

//写入成功, 新的序列号在设备对象中更新。

WCHAR wcText[512] = { 0 };

mbstowcs(wcText, pDevice->uvcIdentity.DevSerial, sizeof(wcText));

MessageBox(wcText);

}

9. 同步读取图像示例代码:

//获取图像，返回图像文件。

char cFile[256] = { 0 };

unsigned int nSize = 0;

unsigned int nFile = 256;

//同步获取

LU\_RESULT lr = GetImage(pDevice, 1, 0, &nSize, cFile, &nFile);

////获取到图像文件。

if(nFile > 0 && strlen(cFile) == nFile)

{

//转换成宽字符。

WCHAR wcFile[256] = { 0 };

MultiByteToWideChar(936, 0, cFile, nFile, wcFile, 256);

//将图像文件显示出来

CImage img;

img.Load(wcFile);

CRect rect;

m\_stImage.GetClientRect(rect);

img.Draw(m\_stImage.GetDC()->m\_hDC, rect);

img.Destroy();

wsprintf(cLog, \_T("设备状态: 获取图像完成"));

}

else

{

wsprintf(cLog, \_T("设备状态: 获取图像 失败"));

}

10. 异步读取图像示例代码:

//获取图像，返回图像文件。

GetImage(pDevice, 0, ImageCallback);

//回调函数

LU\_RESULT CLionSDKDemoDlg::ImageCallback(LU\_DEVICE device, LU\_PCHAR pImgData, LU\_UINT32 nDataBuf, LU\_PCHAR pFile, LU\_UINT32 nFileBuf)

{

//方式，通过界面线程，异步调用

pSDKDemoDlg->ShowImageData(device, pImgData, nDataBuf, pFile, nFileBuf);

return 0;

}

//显示图像

LU\_RESULT CLionSDKDemoDlg::ShowImageData(LU\_DEVICE device, LU\_PCHAR pImgData, LU\_UINT32 nDataBuf, LU\_PCHAR pFile, LU\_UINT32 nFileBuf)

{

///获取到图像文件。

if(nFileBuf > 0 && strlen(pFile) == nFileBuf)

{

//转换成宽字符。

WCHAR wcFile[256] = { 0 };

MultiByteToWideChar(936, 0, pFile, nFileBuf, wcFile, 256);

//将图像文件显示出来

CImage img;

img.Load(wcFile);

CRect rect;

m\_stImage.GetClientRect(rect);

img.Draw(m\_stImage.GetDC()->m\_hDC, rect);

img.Destroy();

}

return 0;

}

**4. SDK** 接口详解

4.1 接口函数  
 DLL 以 stdcall 调用约定导出接口函数。  
 接口函数的返回值，请参考 4.2 节错误码定义。  
 函数参数中涉及到的枚举值，请参考 4.3 节枚举值。  
 函数参数中涉及到的结构体定义，请参考 4.4 节结构体。

函数列表预览：

LIONSDK\_API LU\_RESULT GetDeviceCount(LU\_UINT32 enumLion = 1);

LIONSDK\_API PLU\_DEVICE GetDevice(LU\_UINT32 index, LU\_UINT32 enumLion = 1);

LIONSDK\_API LU\_RESULT OpenDevice(PLU\_DEVICE pDevice);

LIONSDK\_API LU\_RESULT CloseDevice(PLU\_DEVICE pDevice);

LIONSDK\_API LU\_RESULT GetDeviceState(PLU\_DEVICE pDevice, LUDEV\_STATE\* devState);

LIONSDK\_API LU\_RESULT SetDeviceSerial(PLU\_DEVICE pDevice, char\* pSerial);

LIONSDK\_API LU\_RESULT SetDeviceParam(PLU\_DEVICE pDevice, PLU\_PARAM pParam);

LIONSDK\_API LU\_RESULT GetDeviceParam(PLU\_DEVICE pDevice, PLU\_PARAM pParam);

LIONSDK\_API LU\_RESULT GetImage(PLU\_DEVICE pDevice, LU\_UINT32 showUi, LU\_PCHAR pImgData, LU\_UINT32\* nDataBuf, LU\_PCHAR pFile, LU\_UINT32\* nFileBuf);

LIONSDK\_API LU\_RESULT GetImage(PLU\_DEVICE pDevice, LU\_UINT32 showUi, PLionImageCallback callback);

LIONSDK\_API LU\_RESULT AbandonGetImage(PLU\_DEVICE pDevice);

回调函数：

typedef LU\_RESULT (\*PLionImageCallback)(LU\_DEVICE device, LU\_PCHAR pImgData, LU\_UINT32 nDataBuf, LU\_PCHAR pFile, LU\_UINT32 nFileBuf);

函数详解:

LIONSDK\_API LU\_RESULT GetDeviceCount(LU\_UINT32 enumLion = 1);

GetDeviceCount: 获取设备数量

Parameters:

|  |  |
| --- | --- |
| enumLion | [in]Lion设备类型，默认为1 |

Returns:

获取到的设备数量

Remarks:

当前版本，只支持enumLion = 1。

LIONSDK\_API PLU\_DEVICE GetDevice(LU\_UINT32 index, LU\_UINT32 enumLion = 1);

GetDevice:获取单个设备

Parameters:

|  |  |
| --- | --- |
| index | [in] 设备索引，从0开始，不能超过GetDeviceCount返回的设备个数。 |
| enumLion | [in] 设备类型，默认为1 |

Returns:

成功，返回设备对象的指针，否则返回0

Remarks:

enumLion只支持1

LIONSDK\_API LU\_RESULT OpenDevice(PLU\_DEVICE pDevice);

OpenDevice:打开设备

Parameters:

|  |  |
| --- | --- |
| pDevice | [in] 待打开的设备 |
|  |  |

Returns:

LU\_SUCCESS: 打开成功。

其他值打开失败。

Remarks:

已不使用。兼容设备类型。

LIONSDK\_API LU\_RESULT CloseDevice(PLU\_DEVICE pDevice);

CloseDevice:关闭设备

Parameters:

|  |  |
| --- | --- |
| pDevice | [in] 待打开的设备 |
|  |  |

Returns:

LU\_SUCCESS: 关闭成功。

其他值关闭失败。

Remarks:

已不使用。兼容设备类型。

LIONSDK\_API LU\_RESULT GetDeviceState(PLU\_DEVICE pDevice, LUDEV\_STATE\* devState);

GetDeviceState:获取设备状态

Parameters:

|  |  |
| --- | --- |
| pDevice | [in] 待检测的设备 |
| devState | [out] 获得的设备状态值 |

Returns:

LU\_SUCCESS: 获取成功。

其他值获取失败。

Remarks:

只适用于UVC访问设备。

LIONSDK\_API LU\_RESULT SetDeviceSerial(PLU\_DEVICE pDevice, char\* pSerial);

SetDeviceSerial:设置设备参数

Parameters:

|  |  |
| --- | --- |
| pDevice | [in] 待检测的设备 |
| pSerial | [in] 待设置的序列号, 最大长度64位 |

Returns:

LU\_SUCCESS: 成功。

其他值失败。

LIONSDK\_API LU\_RESULT SetDeviceParam(PLU\_DEVICE pDevice, PLU\_PARAM pParam);

SetDeviceParam:设置设备参数

Parameters:

|  |  |
| --- | --- |
| pDevice | [in] 待检测的设备 |
| pParam | [out] 待设置的设备参数 |

Returns:

LU\_SUCCESS: 成功。

其他值失败。

LIONSDK\_API LU\_RESULT GetDeviceParam(PLU\_DEVICE pDevice, PLU\_PARAM pParam);

GetDeviceParam: 获取设备参数

Parameters:

|  |  |
| --- | --- |
| pDevice | [in] 待检测的设备 |
| pParam | [out] 获得的设备参数值 |

Returns:

LU\_SUCCESS: 成功。

其他值失败。

LIONSDK\_API LU\_RESULT GetImage(PLU\_DEVICE pDevice, LU\_UINT32 showUi,

LU\_PCHAR pImgData, LU\_UINT32\* nDataBuf,

LU\_PCHAR pFile, LU\_UINT32\* nFileBuf);

GetImage: 同步获取设备图像

Parameters:

|  |  |
| --- | --- |
| pDevice | [in] 待检测的设备 |
| showUi | [in] 是否显示参数界面 |
| pImgData | [in] 图像数据缓冲区 |
| nDataBuf | [in, out] 图像数据缓冲区大小 |
| pFile | [in] 图像保存文件缓冲区 |
| nFileBuf | [in,out] 图像保存的文件缓冲区大小 |

Returns:

LU\_SUCCESS: 获取成功。

其他值获取失败。

Remarks:

Lion UVC设备，只支持获取的图像数据，通过文件缓冲，返回给调用者。

pImgData 图像缓冲区不支持。

获取到的图像文件是临时缓冲图像文件，在下次获取设备图像时会被覆盖掉。如果需要对图像进行复杂处理，需要用户将临时文件中的数据自己存储。

LIONSDK\_API LU\_RESULT GetImage(PLU\_DEVICE pDevice, LU\_UINT32 showUi, PLionImageCallback callback);

GetImage:异步获取设备图像

Parameters:

|  |  |
| --- | --- |
| pDevice | [in] 待检测的设备 |
| showUi | [in] 是否显示参数界面 |
| callback | [in] 获取图像的回调函数 |

Returns:

LU\_SUCCESS: 获取成功。

其他值获取失败。

typedef LU\_RESULT (\*PLionImageCallback)(LU\_DEVICE device, LU\_PCHAR pImgData, LU\_UINT32 nDataBuf, LU\_PCHAR pFile, LU\_UINT32 nFileBuf);

PLionImageCallback:异步获取设备图像回调函数

Parameters:

|  |  |
| --- | --- |
| pDevice | [in] 待检测的设备 |
| pImgData | [in] 图像数据缓冲区 |
| nDataBuf | [in] 图像数据缓冲区大小 |
| pFile | [in] 图像保存文件缓冲区 |
| nFileBuf | [in] 图像保存的文件缓冲区大小 |

Returns:

LU\_SUCCESS: 获取成功。

其他值获取失败。

Remarks:

Lion UVC设备，只支持获取的图像数据，通过文件缓冲，返回给调用者。

pImgData 图像缓冲区不支持。

获取到的图像文件是临时缓冲图像文件，在下次获取设备图像时会被覆盖掉。如果需要对图像进行复杂处理，需要用户将临时文件中的数据自己存储。

LIONSDK\_API LU\_RESULT AbandonGetImage(PLU\_DEVICE pDevice);

AbandonGetImage:中止获取设备图像

Parameters:

|  |  |
| --- | --- |
| pDevice | [in] 待检测的设备 |
|  |  |

Returns:

LU\_SUCCESS: 中止成功。

其他值获取失败。

4.2 错误码

SDK 错误码用于提供给用户诊断信息，本节将列出所有的错误码定义。接口函数的  
返回值是一个错误码。错误码列表如下

|  |  |
| --- | --- |
| 错误值 | 描述 |
| LU\_SUCCESS | 成功 |
| LU\_FAIL | 失败 |
| LU\_OPENED | 设备已打开 |
| LU\_OTHERDEVOPENED | 其他设备已打开 |
| LU\_UNOPEN | 设备未打开 |
| LU\_PARAMINVALID | 参数无效 |
| LU\_WRITEFAILED | 写入失败 |
| LU\_NOTSUPPORT | 不支持 |
| LU\_MEMLOW | 存储空间不足 |
| LU\_OVERTIME | 超时 |
| LU\_TRIGGEROVERTIME | 触发超时 |
| LU\_READIMAGEOVERTIME | 读图超时 |

4.3 枚举值

//设备状态

enum LUDEV\_STATE{

LUDEVSTATE\_UNOPNE = 1,

LUDEVSTATE\_OPEN,

LUDEVSTATE\_WAITTRIGGER,

LUDEVSTATE\_TRIGGERIMAGE,

LUDEVSTATE\_IMAGESAVE,

LUDEVSTATE\_OVERTIME,

};

//出图模式

enum LU\_MODE{

LUMODE\_AC, //AC

LUMODE\_UTC, //UTC

};

//BINNING

enum LUDEV\_BINNING{

LUDEVBINNING\_NO, //NO

LUDEVBINNING\_YES, //Yes

};

//FPGA Filter

enum LUDEV\_FILTER{

LUDEVFILTER\_NO, //No

LUDEVFILTER\_YES, //Yes

};

//图片格式

enum LUIMG\_TYPE{

LUIMG\_RAW = 1, //raw

LUIMG\_BMP, //bmp

LUIMG\_JPEG, //jpeg

LUIMG\_PNG, //png

};

//设备参数

enum LUDEV\_PARAM{

LUDEVPARAM\_UNKNOWN = 0, //

LUDEVPARAM\_MODE = 1, //出图模式

LUDEVPARAM\_BINNING, //BINNING //暂不支持

LUDEVPARAM\_FILTER, //FPGA进行坏点过滤

LUDEVPARAM\_XRAY, //暂不支持

LUDEVPARAM\_IMAGE, //图片格式 //暂不支持

LUDEVPARAM\_DELAYTIME, //延迟时间 ms单位 //暂不支持

LUDEVPARAM\_TRIGGERTIME, //触发超时设置 s单位 AC=5s VTC=60S

LUDEVPARAM\_READIMAGETIME, //读图超时设置 s秒 5s

LUDEVPARAM\_XRESOLUTION, //暂不支持

LUDEVPARAM\_YRESOLUTION, //暂不支持

LUDEVPARAM\_IMGWIDTH, //图片宽度 //暂不支持

LUDEVPARAM\_IMGHEIGHT, //图片高宽 //暂不支持

LUDEVPARAM\_SAMPLESPERPIXEL, //暂不支持

LUDEVPARAM\_BITESPERPIXEL, //暂不支持

LUDEVPARAM\_BITSPERSAMPLE, //暂不支持

LUDEVPARAM\_PLANAR, //暂不支持

LUDEVPARAM\_PIXELTYPE, //暂不支持

LUDEVPARAM\_COMPRESSION, //暂不支持

LUDEVPARAM\_BRIGHTNESS, //亮度 0 - 100 //暂不支持

LUDEVPARAM\_CONTRAST, //对比度 0 - 100 //暂不支持

LUDEVPARAM\_SATURATION, //饱和度 0 - 100 //暂不支持

LUDEVPARAM\_SHARPEN, //锐化 0 - 100 //暂不支持

LUDEVPARAM\_EMBOSS, //浮雕 0 - 100 //暂不支持

LUDEVPARAM\_BLUR, //模糊 0 - 100 //暂不支持

LUDEVPARAM\_GAMMA, //直方图 0 - 100 //暂不支持

LUDEVPARAM\_FFCDARKENABLE, //暗场FFC校准使能 //暂不支持

LUDEVPARAM\_FFCLIGHTENABLE, //明场FFC校准使能 //暂不支持

LUDEVPARAM\_FCCDARK, //暗场FFC校准 文件路径 //暂不支持

LUDEVPARAM\_FCCLIGHT, //明场FFC校准 文件路径 //暂不支持

};

};

4.4 结构体

typedef struct {

LU\_UINT16 MajorNum;

LU\_UINT16 MinorNum;

LU\_UINT16 Language;

LU\_UINT16 Country;

LU\_STR32 Info;

} LU\_VERSION, FAR \* PLU\_VERSION;

/////

typedef struct {

LU\_UINT32 Id;

LU\_VERSION Version;

LU\_UINT16 ProtocolMajor;

LU\_UINT16 ProtocolMinor;

LU\_UINT32 SupportedGroups;

LU\_STR32 Manufacturer;

LU\_STR32 ProductFamily;

LU\_STR32 ProductName;

} TWD\_IDENTITY, \*PTWD\_IDENTITY;

//

typedef struct{

LU\_UINT32 Id; //设备ID

LU\_STR128 Name; //设备名称

LU\_STR64 DevSerial; //设备序列号

LU\_UINT16 MajorNum; //固件版本高位

LU\_UINT16 MinorNum; //固件版本低位

}LU\_IDENTITY, \*PLU\_IDENTITY;

////

typedef struct {

LU\_IDENTITY uvcIdentity;

TWD\_IDENTITY twIdentity;

}LU\_DEVICE, \*PLU\_DEVICE;

//参数

typedef struct

{

LU\_UINT16 param; //参数 ,开方式参数形式，用于兼容

LU\_UINT16 type; //存储空间数据类型, data实际数据类型为int, short, char.....

LU\_UINT32 size; //存储空间大小,byte个数

LU\_PVOID data; //存储空间，用于存储实际参数值

}LU\_PARAM, \*PLU\_PARAM;