UVC USB Camera
DirectShow SDK Specification

深圳市锐尔威视科技有限公司
RERVISION TECHNOLOGY CO., LTD

2014.9.16 V1.0

目录

一. DirectShow SDK 简介 3
二. 开发环境的搭建 3
三. 捕捉和预览图像 4
1. 建立 GRAPH BUILDER 对象 4
2. 设备列举和捕捉接口 4
3. 开启视频预览 5
4. 设置捕捉文件 6
5. 捕捉视频 8
6. 保存捕捉视频 8
四. 属性设置 9
1. 预览图像输出格式和大小 9
2. 捕获静态图像 9
3. IAMVIDEOPROCAMP 接口 10
4. IAMCAMERACONTROL 接口 12

一. DirectShow SDK 简介

DirectShow 是 DirectX 的组件之一，DirectX 软件开发包是 Microsoft 提供的一套在 Windows平台上开发高性能图形、声音、输入、输出和网络游戏的编程接口。这其中， DirectShow 提供了应用程序从适当的硬件中捕捉和预览音、视频的能力。数据源包括: VCR、Camera、TV Tuner、Microphone 或其他的数据源。应用程序可以立刻显示捕捉的数据(预览)，或是保存到一个文件中。

DirectShow 是基于COM的，为了编写DirectShow应用程序，需要了解COM客户程序编写的基础知识。DirectShow SDK 包含在 Microsoft Windows SDK 内，本文档主要介绍与摄像头的预览、拍照、录像以及属性设置相关的变量以及函数。

DirectShow SDK 自带一个视频预览软件示例 Amcap，本文档会引用该示例中部分片段代码，该示例在 DirectShow SDK 安装路径下 **\Samples\C++\DirectShow\Capture\AMCap**

二. 开发环境的搭建

本文以最新的Visual Studio 2013为例。

先解压DirectShow SDK到任意目录，此SDK中包含了示例Amcap源码，但并未完全包含我们所需要的库，还需编译BaseClasses工程，才能生成我们需要的基本库baseclasses。

在VS2013中导入并编译 DirectShow SDK 安装路径下\ **Samples\C++\DirectShow \BaseClasses** 工程，可以编译为debug版本，也可以编译release 版本。

编译成功后在**\Samples\C++\DirectShow\BaceClasses\Debug_Unicode**目录下将产生相应的库。

将DirectShow SDK集成到我们的项目工程，以Amcap为例：

在VS2013中导入Amcap源码，右键点击工程选择 “属性-->配置属性-->VC++目录”中进行操作：

- 在包含目录中添加
  DirectShow\Include;
  DirectShow\Samples\C++\DirectShow\BaseClasses;
- 在库目录中添加
  DirectShow\Lib;
  DirectShow\Samples\C++\DirectShow\BaseClasses\Debug_Unicode;

在“配置属性-->链接器目录”中进行操作：

- 在附加库目录中添加
  DirectShow\Samples\C++\DirectShow\BaseClasses\Debug_Unicode;
  DirectShow\Lib;

点击工程“右键-->清理-->生成”，将生成exe可执行文件，即可调试。至此已成功搭建了DirectShow的开发环境。

三. 捕捉和预览图像

DirectShow 提供了 Capture Graph Builder 对象使建立视频捕捉应用程序变得更加容易，对象提供 ISampleCaptureGraphBuilder 接口，接口可以建立和控制 Capture Graph，提供的方法满足了基本的捕捉和预览功能的要求：

- FindInterFace方法：在 filter graph 中查找一个与捕捉有关的详细接口，直接访问一个详细接口的功能，不需要列举 filter graph 中的 pins 和 filters。
- ReaderStream方法：连接 Source Filter 和 Rending Filter，择添加一些中间 Filter。
- ControlStream方法：独立精确地控制 graph 的开始和结束帧。

1. 建立 GRAPH BUILDER 对象

AMCap 的 MakeBuilder 函数建立了一个 capture graph builder 对象，构造函数 ISampleCaptureGraphBuilder()通过调用 CoCreateInstance 获得了 CaptureGraphBuilder2 接口指针，并把它存储到 gcap 结构的 pBuilder 中。

```
BOOL MakeBuilder()
{
    // we have one already
    if (gcap.pBuilder)
        return TRUE;
    gcap.pBuilder = new ISampleCaptureGraphBuilder();
    if (NULL == gcap.pBuilder)
    {
        return FALSE;
    }
    return TRUE;
}
```

2. 设备列举和捕捉接口

通过 ICreateDevEnum::CreateClassEnumerator 方法列举捕捉系统中的设备。之后，实例化一个 DirectShow 的filter 去使用这个设备。

```
ICreateDevEnum *pCreateDevEnum = 0;
hr = CoCreateInstance(CLSID_SystemDeviceEnum, NULL,
CLSCTX_INPROC_SERVER, IID_ICreateDevEnum, (void**)&pCreateDevEnum);
IEnumMoniker *pEm = 0;

hr =
pCreateDevEnum->CreateClassEnumerator(CLSID_VideoInputDeviceCategor
y, &pEm,0);
```

通过 ISampleCaptureGraphBuilder::FindInterface 去获得与捕捉相关的接口指针 IAMDroppedFrames，IAMVideoCompression，IAMStreamConfig 以及 IAMVfwCaptureDialogs 。

```
HRESULT ISampleCaptureGraphBuilder::FindInterface(const GUID
*pCategory,
    const GUID *pType,
    IBaseFilter *pf,
    REFIID riid,
    void **ppint
    )
{
    return graphBuilder2_->FindInterface(pCategory, pType, pf, riid,
ppint);
}
```

3. 开启视频预览

AMCap 中InitCapFilters()函数建立了一个filter graph，并将视频过滤器添加到当前的filter graph。

```
hr = gcap.pFg->AddFilter(gcap.pVCap, gcap.wachFriendlyName);
```

调用ISampleCaptureGraphBuilder::RenderStream，capture filter 的preview pin 到video render。

```
hr = gcap.pBuilder->RenderStream(&PIN_CATEGORY_PREVIEW, NULL,
gcap.pVCap, NULL, NULL);
```

默认的video preview window 是一个独立的窗口。如果需要改变默认行为，则可以调用ISampleCaptureGraphBuilder::FindInterface 获得IVideoWindow 接口，第二个参数通过gcap.pVCap 指定，描述video capture filter，三个参数是想得到的接口IVideoWindow，后返回的是接口。得到IVideoWindow 接口后，可以调用IVideoWindow 的方法如put_Owner、put_WindowStyle 或者SetWindowPosition去获得video preview window的句柄，设置窗口属性，或把它放到想要的位置。

```
// This will go through a possible decoder, find the video renderer it's
// connected to, and get the IVideoWindow interface on it
hr = gcap.pBuilder->FindInterface(&PIN_CATEGORY_PREVIEW, gcap.pVCap,
IID_IVideoWindow, (void **)&gcap.pVW);
if (hr != NOERROR) {
    ErrMsg("This graph cannot preview");
} else {
    RECT rc;
```

```
    gcap.pVW->put_Owner((long)ghwndApp); // We own the window now
    gcap.pVW->put_WindowStyle(WS_CHILD); // you are now a child
    // give the preview window all our space but where the status bar
is
    GetClientRect(ghwndApp, &rc);
    cyBorder = GetSystemMetrics(SM_CYBORDER);
    cy = statusGetHeight() + cyBorder;
    rc.bottom -= cy;
    gcap.pVW->SetWindowPosition(0, 0, rc.right, rc.bottom); // be this
big
    gcap.pVW->put_Visible(OATRUE);
}
```

4. 设置捕捉文件

使用普通的OpenFile Dialog 获得捕捉文件的信息。调用AllocCaptureFile 函数为捕捉文件分配空间。因为空间比较巨大，所以这样可以提高捕捉的速度。

ISampleCaptureGraphBuilder::AllocCapFile 执行实际的文件分配；

IFileSinkFilter::SetFileName 指示file writer filter 使用用户选择的文件名保存数据；

ISampleCaptureGraphBuilder:: SetOutputFileName 把file writer filter 加入filter graph。

```
HRESULT
ISampleCaptureGraphBuilder::AllocCapFile(LPCOLESTR lpwstr, DWORDLONG
dwlSize)
{
    return graphBuilder2_->AllocCapFile(lpwstr, dwlSize);
}
```

```
HRESULT
ISampleCaptureGraphBuilder::SetOutputFileName(const GUID *pType,
LPCOLESTR lpwstrFile,
IBaseFilter **ppf,
IFileSinkFilter **pSink)
{
    if (!pType || !lpwstrFile || !ppf || !pSink)
    {
        return E_INVALIDARG;
    }
```

```
    if (!::IsEqualGUID(*pType, MEDIASUBTYPE_Mpeg2))
    {
        return graphBuilder2_->SetOutputFileName(pType, lpwstrFile,
ppf, pSink);
    }
HRESULT hr;
    if (!graph_)
    {
        hr = GetFiltergraph(&graph_);
        if (FAILED(hr))
        {
            return hr;
        }
    }
    //
    // Configure the dump filter
    SmartPtr< IFileSinkFilter > pDump;
    hr = CoCreateInstance(CLSID_Dump, NULL, CLSCTX_INPROC_SERVER,
IID_IBaseFilter, (void**)&pDump);
    if (FAILED(hr))
    {
        return hr;
    }
    hr = pDump->SetFileName(lpwstrFile, NULL);
    if (FAILED(hr))
    {
        return hr;
    }
    hr = pDump.QueryInterface(&pMPEG2Demux_);
    if (FAILED(hr))
    {
        return hr;
    }
    hr = graph_->AddFilter(pMPEG2Demux_, L"Dump");
    if (FAILED(hr))
    {
        pMPEG2Demux_ = NULL;
        return hr;
    }
    *pSink = pDump;
    return S_OK;
}
```

5. 捕捉视频

ISampleCaptureGraphBuilder::SetOutputFileName 是一个关键方法。它把multiplexer和file writer 添加连接到filter graph 中，并设置文件名字。第一个参数MEDIASUBTYPE_Avi（DirectShow 提供的数据格式），指出capture graph builder 对象将插入一个AVI multiplexer filter，因此 file writer 将以AVI 文件格式记录捕捉的数据。二个参数(wach)是文件名。后的两个参数指出 multiplexer filter (gcap.pRender) 和file writer filter (gcap.pSink) ，这两个是通过SetOutputFileName 函数初始化的。

```
// We need a rendering section that will write the capture file out in
AVI
// file format
WCHAR wach[_MAX_PATH];
MultiByteToWideChar(CP_ACP, MB_PRECOMPOSED, gcap.szCaptureFile, -1,
wach, _MAX_PATH);
GUID guid = MEDIASUBTYPE_Avi;
hr = gcap.pBuilder->SetOutputFileName(&guid, wach, &gcap.pRender,
&gcap.pSink);
if (hr != NOERROR)
{
    ErrMsg("Error %x: Cannot set output file", hr);
    goto SetupCaptureFail;
}
```

6. 保存捕捉视频

最初分配的捕捉文件只是保存临时的数据，以便尽可能快的捕捉。当需要把捕捉的数据保存到硬盘中时，调用ISampleCaptureGraphBuilder::CopyCaptureFile。该方法从得到的捕捉文件中输出数据到选择的另一个文件中，这个新的储存文件的大小适合实际捕捉的数据匹配的。

ISampleCaptureGraphBuilder::CopyCaptureFile 方法的第一个参数是复制源，第二个是目标文件。第三个参数设为TRUE 指出用户允许用ESC 键中断复制操作。最后参数是可选的。允许提供一个进程指示器。

```
HRESULT
ISampleCaptureGraphBuilder::CopyCaptureFile(LPOLESTR lpwstrOld,
                          LPOLESTR lpwstrNew,
                          int fAllowEscAbort,
                          IAMCopyCaptureFileProgress *pCallback)
{
    return graphBuilder2_->CopyCaptureFile(lpwstrOld, lpwstrNew,
                                      fAllowEscAbort, pCallback);
};
```

四. 属性设置

DirectShow 提供的Capture Graph Builder 对象不但可以满足基本的捕捉和预览功能的要求，还在strmif.h 头文件中提供了大量控制接口和方法控制图像预览的属性。

1. 预览图像输出格式和大小

接口：IID_IAMStreamConfig 所提供的四个方法函数，提供了获取设备支持的预览格式和大小，获取设备当前预览格式和大小，设置新的预览格式和大小功能。

```
EXTERN_C const IID IID_IAMStreamConfig;
MIDL_INTERFACE("C6E13340-30AC-11d0-A18C-00A0C9118956")
IAMStreamConfig : public IUnknown
{
public:
    virtual HRESULT STDMETHODCALLTYPE SetFormat(
        /* [in] */ AM_MEDIA_TYPE *pmt) = 0;
    virtual HRESULT STDMETHODCALLTYPE GetFormat(
        /* [annotation][out] */
        __out AM_MEDIA_TYPE **ppmt) = 0;
    virtual HRESULT STDMETHODCALLTYPE GetNumberOfCapabilities(
        /* [annotation][out] */
        __out int *piCount,
        /* [annotation][out] */
        __out int *piSize) = 0;
    virtual HRESULT STDMETHODCALLTYPE GetStreamCaps(
        /* [in] */ int iIndex,
        /* [annotation][out] */
        __out AM_MEDIA_TYPE **ppmt,
        /* [annotation][out] */
        __out BYTE *pSCC) = 0;
};
```

- 使用ISampleCaptureGraphBuilder::FindInterface(&PIN_CATEGORY_CAPTURE, &MEDIATYPE_Video,gcap.pVCap,IID_IAMStreamConfig,(void **)&pConfig) 方法获得IID_IAMStreamConfig 接口；
- 使用IAMStreamConfig::GetNumberOfCapabilities 以及IAMStreamConfig::GetStreamCaps 可获得设备支持的图像预览格式和大小；
- 使用IAMStreamConfig:: SetFormat 方法可设置图像预览格式和大小。

2. 捕获静态图像

从摄像头中捕获静态图像的主要方式有两种：大多数摄像头支持Still Pin Capture 方式，若设备不支持Still Pin，可以通过Preview mode 动态抓取图像数据。这两种方式均是在预览的视频流中插入一个SampleGrabberFilter，这样视频流中的每一帧数据都会经

过该Filter，只要让该Filter 通过回调机制通告上层应用程序进行对当前帧进行处理便可实现图像采集。此处需要注意的是，DirectShow 的视频采集与应用程序的工作线程是两个独立的线程。

Still Pin Capture 的工作方式类似于相机快门的拍照接口，用户发送了捕获命令后，触发摄像头进行拍摄，DirectShow 在收集完数据后再通过消息机制通告给应用程序。

- 定义一个类实现Sample Grabber 的回调接口IsampleGrabberCB；
- 调用RenderStream 依次把Still pin、Sample Grabber 和系统默认Renderer Filter 连接起来；
- 配置Sample Grabber 以捕获数据；

在摄像头不支持Still Pin的情况下，可以从视频流中捕获帧数据。摄像头每产生一帧视频都会通告给操作系统，由于预览视频流源源不断地从摄像头设备处产生，DirectShow 会通过回调机制不断地发送消息给应用程序。用户未发送拍照命令时，应用程序忽略这些消息；若用户发送拍照命令，应用程序相应消息并抓取Buffer。

两种帧数据获取均通过插入SampleGrabber Filter（对应的ID 为CLSID_SampleGrabber）并回调应用程序实现，主要在于ISampleGrabberCB 类的成员函数BufferCB。BufferCB 是一个可获取当前图像Sample 指针的回调函数：

```
HRESULT BufferCB(
    double SampleTime,
    BYTE *pBuffer,
    long BufferLen
    );
```

其中，参数SampleTime 为Sample 的开始时间、pBuffer 为指向Sample 的指针、BufferLen为Sample 的大小。若数据抓取成功，则函数返回S_OK，否则返回失败。

3. IAMVIDEOPROCAMP 接口

IAMVideoProcAmp 接口提供了亮度、对比度、色调、饱和度、清晰度、伽马值、黑白、白平衡、逆光对比、增益等属性设置。

```
typedef
enum tagVideoProcAmpProperty
{
    VideoProcAmp_Brightness = 0,
    VideoProcAmp_Contrast = (VideoProcAmp_Brightness + 1),
    VideoProcAmp_Hue = (VideoProcAmp_Contrast + 1),
    VideoProcAmp_Saturation = (VideoProcAmp_Hue + 1),
    VideoProcAmp_Sharpness = (VideoProcAmp_Saturation + 1)
```

```
    VideoProcAmp_Gamma = (VideoProcAmp_Sharpness + 1),
    VideoProcAmp_ColorEnable = (VideoProcAmp_Gamma + 1),
    VideoProcAmp_WhiteBalance = (VideoProcAmp_ColorEnable + 1),
    VideoProcAmp_BacklightCompensation = (VideoProcAmp_WhiteBalance +
1),
    VideoProcAmp_Gain = (VideoProcAmp_BacklightCompensation + 1)
} VideoProcAmpProperty;

typedef
enum tagVideoProcAmpFlags
{
    VideoProcAmp_Flags_Auto = 0x1,
    VideoProcAmp_Flags_Manual = 0x2
} VideoProcAmpFlags;
```

IID_IAMVideoProcAmp 接口所提供的三个方法函数，分别提供了获取范围及标志位，获取当前值，设置功能。参数long Property 为枚举tagVideoProcAmpProperty，参数long Flags 为枚举 tagVideoProcAmpFlags。

```
EXTERN_C const IID IID_IAMVideoProcAmp;
MIDL_INTERFACE("C6E13360-30AC-11d0-A18C-00A0C9118956")
IAMVideoProcAmp : public IUnknown
{
public:
    virtual HRESULT STDMETHODCALLTYPE GetRange(
        /* [in] */ long Property,
        /* [annotation][out] */
        __out long *pMin,
        /* [annotation][out] */
        __out long *pMax,
        /* [annotation][out] */
        __out long *pSteppingDelta,
        /* [annotation][out] */
        __out long *pDefault,
        /* [annotation][out] */
        __out long *pCapsFlags) = 0;
```

```
    virtual HRESULT STDMETHODCALLTYPE Set(
        /* [in] */ long Property,
        /* [in] */ long lValue,
        /* [in] */ long Flags) = 0;
    virtual HRESULT STDMETHODCALLTYPE Get(
        /* [in] */ long Property,
        /* [annotation][out] */
        __out long *lValue,
        /* [annotation][out] */
        __out long *Flags) = 0;
};
```

例如，自定义第一个参数为13 可设置电源频率：

- 使用IBaseFilter:: QueryInterface 方法可获取接口；
- 使用IAMVideoProcAmp:: GetRange 方法获取数值范围，第一个参数属性值为13；
- 使用IAMVideoProcAmp:: Set 方法设置电源频率第一个参数属性值为13，第二个参数当电源频率50Hz 时为1、60Hz 时为2，第三个参数为 VideoProcAmp_Flags_Auto。

4. IAMCAMERACONTROL 接口

IAMCameraControl 接口提供了全景、倾斜、镜像翻转、缩放、曝光、光圈、焦点等设置。

```
typedef
enum tagCameraControlProperty
{
    CameraControl_Pan = 0,
    CameraControl_Tilt = (CameraControl_Pan + 1),
    CameraControl_Roll = (CameraControl_Tilt + 1),
    CameraControl_Zoom = (CameraControl_Roll + 1),
    CameraControl_Exposure = (CameraControl_Zoom + 1),
    CameraControl_Iris = (CameraControl_Exposure + 1),
    CameraControl_Focus = (CameraControl_Iris + 1)
} CameraControlProperty;
```

```
typedef
enum tagCameraControlFlags
{
    CameraControl_Flags_Auto = 0x1,
    CameraControl_Flags_Manual = 0x2
} CameraControlFlags;
```

IAMCameraControl 接口所提供的三个函数与3.3 节中IAMVideoProcAmp 接口所提供的三个函数用法相同。

```
EXTERN_C const IID IID_IAMCameraControl;
MIDL_INTERFACE("C6E13370-30AC-11d0-A18C-00A0C9118956")
IAMCameraControl : public IUnknown
{
public:
    virtual HRESULT STDMETHODCALLTYPE GetRange(
        /* [in] */ long Property,
        /* [annotation][out] */
        __out long *pMin,
        /* [annotation][out] */
        __out long *pMax,
        /* [annotation][out] */
        __out long *pSteppingDelta,
        /* [annotation][out] */
        __out long *pDefault,
        /* [annotation][out] */
        __out long *pCapsFlags) = 0;
    virtual HRESULT STDMETHODCALLTYPE Set(
        /* [in] */ long Property,
        /* [in] */ long lValue,
        /* [in] */ long Flags) = 0;
    virtual HRESULT STDMETHODCALLTYPE Get(
        /* [in] */ long Property,
        /* [annotation][out] */
        __out long *lValue,
        /* [annotation][out] */
        __out long *Flags) = 0;
};
```