Windows10 Media Foundation

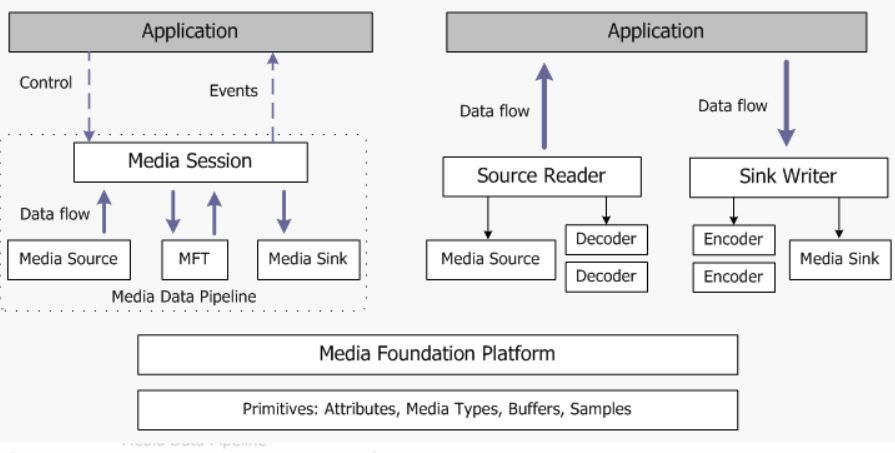
<https://docs.microsoft.com/en-us/windows/win32/medfound/whats-new-for-media-foundation>

Media Foundation

1. Better format support
2. Support capture devices and hardware codecs
3. A simplified programming model
4. Improvements to the platform

Simplified Programming Model

|  |  |
| --- | --- |
| API | Descr |
| Source Reader | Pulls raw or decoded data from a media file. |
| Sink writer | Author media file by passing in uncompressed or encoded data. |
| Transcode API | Support the most common audio/video encoding scenario |



About Media Foundation

MF uses component object model(COM) and requires C/C++.

The MF APIs are part of the windows SDK.**To develop a Media Foundation application, install the latest version of the Windows SDK.**

Audio and Video Quality

* Direct Video Acceleration (DXVA)
* Color-space information is preserved throughput the video pipeline.
* The enhanced video render(EVR) offers better timing support
* MF uses the multimedia Class Scheduler Service(MMCSS)

Content Access

Content protection will become an integral part of digital media products.

**Media Foundation Programming Guide**

Get started

Writing Media APP

Media Format support

Low-level Video APIs

Writing Media Foundation Plug-ins

Getting started

Essential Concepts

Some topics you will need to understand before writing a MF APP.

Streams

Compression

Media Container

Format

Related topics

Streams:

A stream is a sequence of media data with a uniform type

Compression :

Compression algorithms fall into two broad categories:

Lossless compression

Lossy compression

Media Containers:

Media files usually contain at least some of the following elements:

File header

An index that enables random access to the content

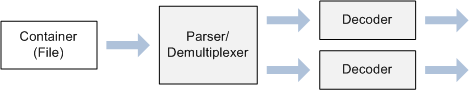
Metadata that describes the content

Packet headers, to enable network transmission or random access.

Formats

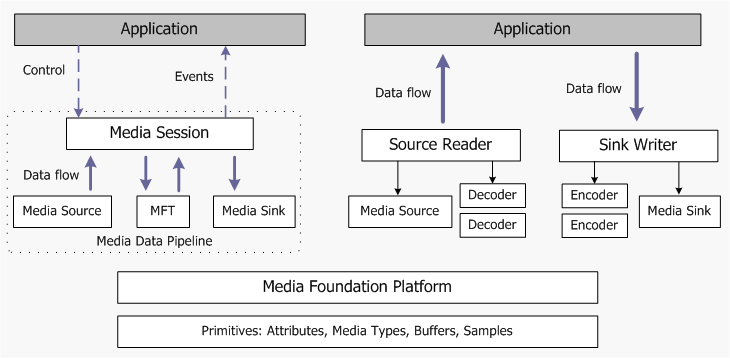
Reading a media file actually involves two stages:

1. First, the container must be parsed.
2. Next, if the streams are compressed, they must be decoded using the appropriate decoders.



**Media Foundation Architecture**

Overview of the Media Foundation Architecture



Primitives and Platform

Started from bottom of the diagram, the primitives are helper objects used

Throughput the media Foundation API:

Attributes are a generic way to store information inside an object, as a list of key/value pairs.

Media types

Media buffers

Media samples

The Media Foundation Platform APIs provide some core functionality that

Is used by the Media Foundation pipeline.

Certain APPs might need to call these APIs directly; also, you will need them if you implement a custom source, transform, or sink for Media Foundation.

[Media Foundation Platform APIs](https://docs.microsoft.com/en-us/windows/win32/medfound/media-foundation-platform-apis)

Media Pipeline

The media pipeline containers three types of object that generate or process media data:

>Media Sources introduce data into pipeline. From local file/network stream; from a hardware capture device

>Media Foundation Transforms(MFTs) process data from a stream.Encoder and decoders are implemented as MFTs.

>Media Sinks consume the data; for example, by showing video on the display,or writing data to a media file.

Third parties can implement their own custom sources, sinks, and MFTs; for example to support new media file formats.

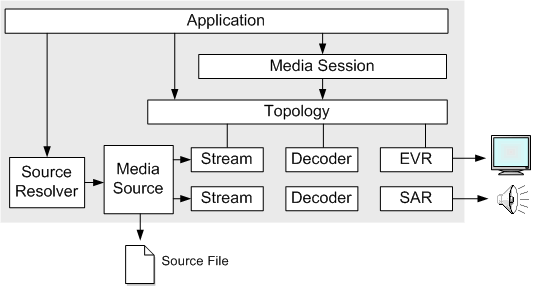
>Media session controls the flow of data through the pipeline, and handles tasks such as quality control, audio/video synchronization, and responding to format changes.

Source Reader and Writer

The source reader and Sink writer provide an alternative way to use the basic Media Foundation components(media sources, transforms, and media sinks).

The source reader hosts a media source and zero or more decoders, while the sink writer hosts a media sink and zero or more encoders.

How to play Media files with MF

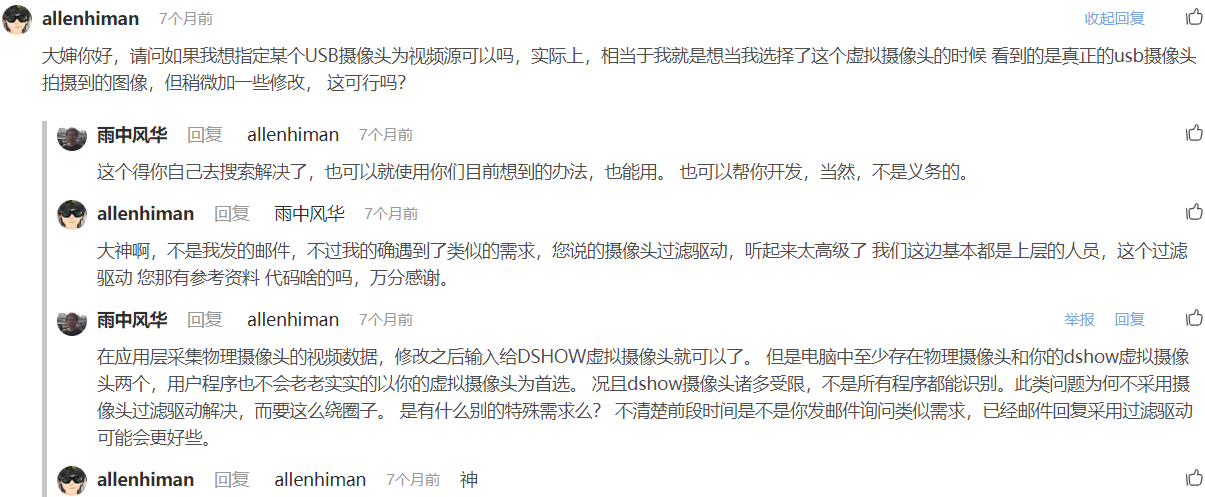


<https://blog.csdn.net/fanxiushu/article/details/8496747>

<https://blog.csdn.net/fanxiushu/article/details/104609499>

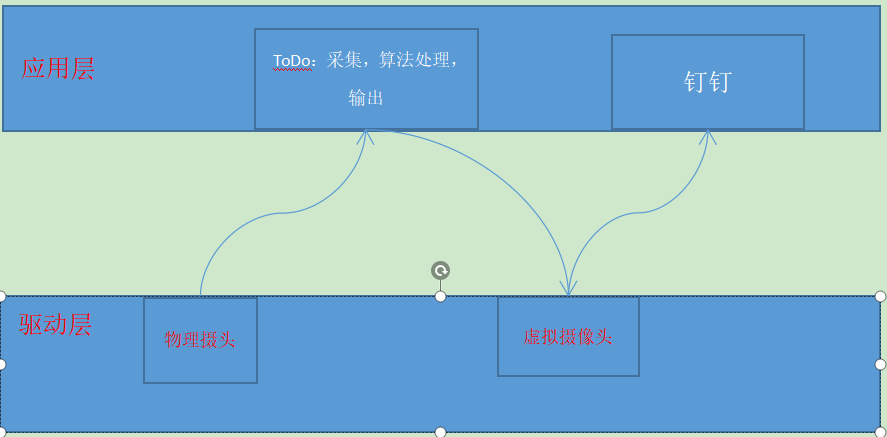
摄像头过滤驱动

<https://blog.csdn.net/fanxiushu/article/details/8842550>



**方案1**

介入方式：应用层采集物理摄像头数据，加上算法处理后，送给虚拟摄像头，应用程序选择虚拟摄像头作为视频源。



实际测试：

首先打开新浪虚拟视频软件

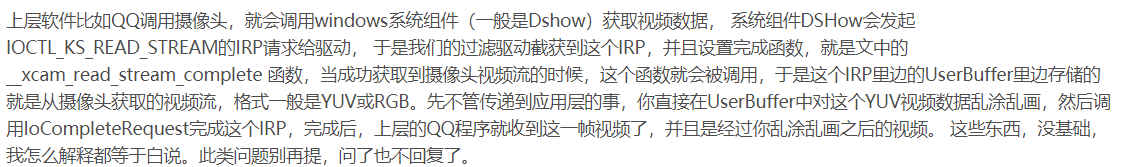
钉钉，直播效果

VipKid上测试效果



方案二



拦截 IOCTL\_KS\_READ\_STREAM 可以截获读取的数据流，  
拦截 IOCTL\_KS\_WRITE\_STREAM 可以截获写入的数据流。

回到摄像头过滤驱动来，摄像头内核流比较规律，  
拦截 IOCTL\_KS\_READ\_STREAM 就可以截获摄像头每一帧的视频数据。  
拦截 IOCTL\_KS\_PROPERTY 就可以获得视频数据的格式。

我们开发这个摄像头过滤驱动的目的，是把摄像头的视频数据发送到应用层程序，在应用层编辑处理视频数据帧，之后再回传给驱动，这样，我们就可以编辑处理视频，比如给摄像头视频加漂亮的边框，制作水印等各种特效的摄像头视频，

1. 根据上一章所说的WDM Filter框架，为了把视频数据传输到应用层，需要创建一个控制设备。

接着进入正题，如何获得摄像头的每一帧数据

系统上层会调用 DeviceIoControl 传递 IOCTL\_KS\_READ\_STREAM的 IOCTL来获得每一个视频帧的数据，跟所有内核流的IOCTL一样，他们都使用 METHOD\_NEITHER 的办法来传输数据，也就是直接使用应用层内存地址，需要在进程上下文存取数据才有效。

对于 IOCTL\_KS\_READ\_STREAM，IRP->UserBuffer 指向 KSSTREAM\_HEADER  结构，此结构里定义了相关参数，

以及需要把摄像头每一帧数据存放的内存地址，注意这个地址依然是应用层的内存地址，也必须在进程上下文访问才有效。

所以我们的处理办法，就是在 过滤设备的派遣函数 处理 IOCTL\_KS\_READ\_STREAM 的时候，创建MDL，并锁定到这个内存地址。

在 IOCTL\_KS\_READ\_STREAM的完成函数里，已经成功获得摄像头的数据帧了，

通过这个MDL，就能随意访问这个数据帧的内容。以下是代码片段，应该是不难理解的。

# 虚拟摄像头驱动原理及开发

<https://www.e2esoft.cn/camm/>

****新浪的虚拟视频是DirectShow应用层上的视频模拟，大概的思路就是：  
因为所有的摄像头WDM驱动，都需要通过运行在应用层的ksproxy.ax与DirectShow交互，ksproxy.ax顾名思义，是WDM驱动同DShow的代理，****

****因此在应用层模拟虚拟摄像头，可在这上面做文章（但是有个遗憾就是老的VFW不支持这种方式）。  
具体怎么实现，有兴趣的可搜索相关文档。****

****e2esoft的VCam是真正基于WDM驱动的虚拟摄像头，他有一个WDM驱动来模拟摄像头，  
在应用程序看来，跟一个真正的摄像头没有分别，所以他的适用范围很广，基本上真正摄像头能干的事情，他都能做到。****

****以上说的只是大致流程，对初学驱动的同学可能根本不理解，所以要完全理解并能自己开发一个虚拟摄像头驱动，  
还是得靠自己查阅大量的相关资料。****

**Directshow 虚拟摄像头**

使用DirectShow操作摄像头的流程是如何的

首先使用CoCreateInstance创建 IGraphBuilder接口，这个接口是管理所有接口的“总管”。

之后从IGraphBuilder查询出IMediaControl控制接口。

然后就是创建ICreateDevEnum接口，从此接口枚举出系统所有安装的摄像头，枚举的当然也包括DSHOW虚拟摄像头。

选择你感兴趣的摄像头，并且获取这个摄像头的IBaseFilter接口， 把这个接口添加到IGraphBuilder中

再然后选择其他Filter，比如压缩的Filter，Render Filter等，都加到IGraphBuilder中。

之后，查询每个Filter的PIN接口，把匹配的PIN接口连接起来。这样就构成了一个DSHOW的连接图。

要使整个“”图“” 动起来，只需要运行 IMediaControl 的Run函数，这样摄像头的数据就会流经每个Filter，  
最终达到 Render Filter 在终端展现出来。

大致伪代码如下：

hr = CoCreateInstance(CLSID\_FilterGraph, NULL, CLSCTX\_INPROC,

        IID\_IGraphBuilder, (void\*\*)&graphBuilder); ///创建 IGraphBuilder接口

hr = graphBuilder->QueryInterface(IID\_IMediaControl, (void\*\*)&control); //查询IMediaControl

CComPtr<ICreateDevEnum> DevEnum; ///创建枚举摄像头设备接口

hr = CoCreateInstance(CLSID\_SystemDeviceEnum, NULL, CLSCTX\_INPROC\_SERVER, IID\_ICreateDevEnum, (void\*\*)&DevEnum);

CComPtr<IEnumMoniker> pEM;//枚举

IMoniker\* pM; //查询到的每个设备

hr = DevEnum->CreateClassEnumerator(CLSID\_VideoInputDeviceCategory, &pEM, 0);

while (pEM->Next(1, &pM, &fetch) == S\_OK) {

     ///开始枚举每个设备，如果是我们的虚拟DSHOW摄像头，也会被枚举到

     ........

    ///选择我们感兴趣的摄像头, 获取Filter接口，比如deviceFilter名字

    pM->BindToObject(0, 0, IID\_IBaseFilter, (void\*\*)&deviceFilter);

}

/////创建其他Filter，比如renderFilter，然后全部添加到IGraphBuilder中.  
graphBuilder->AddFilter(deviceFilter, L"DeviceFilter"); //添加摄像头的filter到GraphBuilder中，

graphBuilder->AddFilter(renderFilter, L"RenderFilter"); //添加其他Filter到GraphBuilder中，

//利用 IBaseFilter 接口的EnumPins函数查找合适PIN，假设上面两个Filter的PIN分别是 devicePin， renderPin,   
对这些PIN再做些其他配置等。。。。

最后调用 IGraphBuilder的Connect函数把PIN连接起来，如下  
graphBuilder->Connect(devicePin,renderPin);

这样初始化完成，   
调用 control->Run ， 即可让其运行起来。

以上就是使用DSHOW操作摄像头的通用流程，要让DirectShow虚拟摄像头能被正确识别和运行，  
需要遵照下面流程，实现各种接口。

首先，要被ICreateDevEnum 接口识别到我们的虚拟摄像头，肯定得先注册我们的DSHOW摄像头。

DirectShow框架已经帮我们提供了这样的注册函数。

我们的虚拟摄像头需要实现在DLL动态库中  
（本来刚开始想实现在EXE中，想通过进程间COM方式，结果以失败告终，所以认为DirectShow框架只认识DLL方式的Filter），  
这个DLL需要具备COM接口动态库的一切基本条件，

需要有DllRegisterServer， DllUnregisterServer， DllGetClassObject，DllCanUnloadNow四个导出函数。

我们需要首先按照普通进程内COM注册方式把DLL注册进系统，   
然后就是我们为了让DSHOW框架枚举到我们的虚拟DirectShow设备，需要做的特别处理

创建 IFilterMapper2接口，调用接口函数RegisterFilter ，把我们的虚拟摄像头注册进去。  
这样ICreateDevEnum 接口就能识别到了。

大致伪代码如下：

        IFilterMapper2\* pFM = NULL;

        hr = CoCreateInstance(CLSID\_FilterMapper2, NULL, CLSCTX\_INPROC\_SERVER, IID\_IFilterMapper2, (void\*\*)&pFM);

REGPINTYPES PinTypes = {

    &MEDIATYPE\_Video,

    &MEDIASUBTYPE\_NULL

};

REGFILTERPINS VCamPins = {

    L"Pins",

    FALSE, ///

    TRUE,  /// output

    FALSE, /// can hav none

    FALSE, /// can have many

    &CLSID\_NULL, // obs

    L"PIN",

    1,

    &PinTypes

};

REGFILTER2 rf2;

rf2.dwVersion = 1;

rf2.dwMerit = MERIT\_DO\_NOT\_USE;

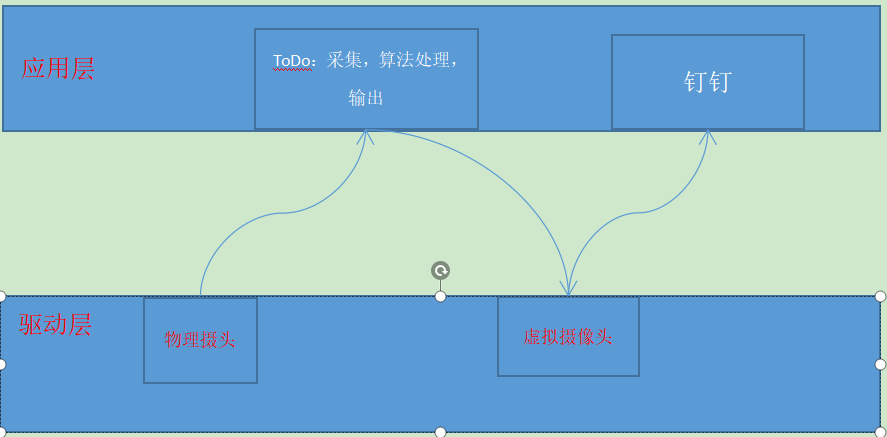
rf2.cPins = 1;

rf2.rgPins = &VCamPins;

 //根据上边提供的信息，调用RegisterFilter 注册。

pFM->RegisterFilter(CLSID\_VCamDShow, L"Fanxiushu DShow VCamera", &pMoniker, &CLSID\_VideoInputDeviceCategory, NULL, &rf2);

把以上代码添加到DllRegisterServer导出函数中，当调用DllRegisterServer注册COM组件的时候，也就把DSHOW虚拟摄像头注册进去了，同样注销也是类似处理。  
其中 CLSID\_VCamDShow 是我们自己定义的GUID，用来标志我们的虚拟摄像头接口。  
系统也会根据这个GUID来获取我们的接口进行后续的操作



驱动层

钉钉

ToDo：采集，算法处理，输出

虚拟摄像头

应用层

物理摄头