VIDEOLOGY
IMAGING SOLUTIONS INC.
Original Equipment Manufacturer

.NET Wrapper SDK Descriptor

9 April 2014

Introduction

This document is the reference material for the .NET wrapper class for the Videology USB-C cameras: the 20K1xx, the 21K1xx, the 24B1.3, and the 24C1.3.

The drivers for these cameras are AVStream kernel streaming drivers, which are exposed to user-mode applications as DirectShow components, configured and managed using COM objects and methods. This wrapper provides a non-COM interface, for those users who are more comfortable working in a non-COM environment. It also provides a .NET assembly for this wrapper.

Organization

The wrapper consists of two components: VidWrapper.dll and VidWrapClr.dll.

VidWrapper.dll is an unmanaged C++ DLL that contains the basic implementation of the wrapper class. It can be used on its own, from an unmanaged C++ project. The DLL exports a single entry point (CreateCamWrapper) that returns an object that can be used to manage the cameras and streaming video. The interface is through a pure abstract base class, ICamWrapper, provided in the WrapIntf.h header file. However, it is expected that the normal use case will be to use the .NET wrapper.

VidWrapClr.dll is a managed C++/CLI assembly that wraps the unmanaged class above in a .NET-friendly module. To use this, just add a reference to VidWrapClr.dll to your project, and add a directive using the namespace Videology. VidWrapClr.dll requires that VidWrapper.dll be present in the same directory.

The kit also includes a sample C# viewer, CsTest.cs that demonstrates most of the the basic usage of the wrappers.

Functions

All of the services in the wrapper are contained within the namespace Videology. The namespace contains two enumerations, a delegate typedef, and a class containing the real functionality.

public enum class PixelFormat

This enumeration contains the pixel formats that can be requested for returning video frames. Four types are currently defined:

- **pf_UYVY** is a YUV 4:2:2 format as defined in CCIR 656, referred to in DirectShow as the four-character code UYVY. 16 bits per pixel, with the bytes in address order as U, Y0, V, Y1. The U and V components are shared between the two pixels.
- **pf_Y8** is a monochrome format containing only the Y component of the image. 8 bits per pixel. This has the fourcc Y800.
- **pf_RGB24** is the color format of a standard 24-bit DIB. 24 bits per pixel, with the bytes in address order as B, G, R, B, G, R.
- **pf_RGB32** is the color format of a standard 32-bit DIB. 32 bits per pixel, with the bytes in address order as B, G, R, 0, B, G, R, 0.

public enum class CameraMode

This enumeration defines the streaming modes that can be used.

- In **Off** mode, the camera is not streaming. No USB bandwidth is reserved.
- In **Streaming** mode, the camera is streaming data and delivering it to an image callback, but no preview window is displayed. If you wish images to be displayed, you will need to display them in your application.
- In **Live** mode, the camera is streaming data and maintaining a preview window. You may provide a window handle to be used as the containing parent for the live preview window.
- In **Snapshot** mode, the graph listens to the camera's snapshot pin, where frames are only delivered upon receiving a trigger signal, either externally or internally.
- In **Single** mode, the camera sensor is held idle until the trigger signal is received, instead of sending on its normal streaming clock. When the trigger is received, the sensor sends one frame, then goes idle again. This mode is available only for the 24B1.3 cameras.

public delegate void BufferDelegate(int, IntPtr, int, int, int);

This delegate class is used to define the callback function that is called each time a new frame arrives. The callback receives 5 parameters:

- Context – an arbitrary value that is passed through to the callback from the SetCallback function
- Data – a pointer to the bytes of the image
- Size – the number of bytes in the image data
- Width – the width, in pixels, of the image data
- Height – the height, in scanlines, of the image data

public ref struct Range

This helper class is used to return range values for those properties that accept integer values. It class contains three members:

- lower – minimum value of property
- upper – maximum value of property
- step – step between meaningful values

public ref struct RangeF

This helper class is used to return range values for those properties that accept floating point values. It class contains three members:

- lower – minimum value of property
- upper – maximum value of property
- step – step between meaningful values

public ref class Camera

The real functionality of the wrapper lies in the class Camera. The application should create an instance of this class early in its initialization and use the instance to invoke the remainder of the methods. Creating this class does not actually open an instance of the camera.

In C#:

```
using Videology;  // must have reference to VidWrapClr.dll

public class CameraTester : Form
{
   private System.ComponentModel.IContainer components = null;
   private System.Windows.Forms.Timer timer1;
   private Videology.Camera cam;
   private Size FrameSize;

   public CameraTester()
   {
      FrameSize = new System.Drawing.Size( 800, 600 );
      InitializeComponent();
           cam = new Videology.Camera();
   }
   …
}
```

void Enumerate()

The "Enumerate" method instructs the wrapper to rescan the list of hardware to count up the number of available cameras.

property int Count (get only)

The "Count" property returns the number of Videology cameras located. The camera ordinal supplied to Open should be less than or equal to this value. This value is only updated by the "Enumerate" method. The sample application starts up a timer to call

Enumerate twice a second while a camera is not opened, so it cam check whether cameras have been plugged or unplugged.

void Open(int CameraOrdinal)

The "Open" function gains access to a Videology camera. The CameraOrdinal parameter specifies which camera should be used, if multiple cameras are plugged in. The first camera is number 0. An exception is thrown if the numbered camera cannot be found. This function must be called before manipulating the camera's state with the other function. This does not start streaming.

void Close()

The "Close" function terminates access to a Videology camera. Each "Open" should be paired with a "Close". If you use a callback to receive frames, the "Close" function will block until all callbacks return.

void SetFrameSize(System::Drawing::Size FrameSize)

The "SetFrameSize" function sets the size of the image that the camera should deliver. The set of valid image sizes is different for each camera model, and in some cases depends on the type of USB communication. The 24B1.3 cameras can select from 320x240, 640x480, 800x600, 1024x768 and 1280x1024. The 20K1xx cameras are fixed at 720x480 for a bulk connection, or 640x480 for an isochronous connection. The 21K1xx cameras are fixed at 720x576 for a bulk connection, or 640x576 for an isochronous connection.

DirectShow does not allow the frame size to be changed on the fly, so if you set the frame size while you are streaming, the wrapper will automatically stop and restart the graph.

System::Drawing::Size GetFrameSize( )

The "GetFrameSize" function returns the actual image size set by the camera. If "SetFrameSize" attempts to set a size that is not supported, the camera will choose the closest supported size.

property int FrameRate

When read, the "FrameRate" property returns the actual frame rate per second as delivered by the camera driver, averaged over the last 3 seconds.

When written, the "FrameRate" property sets the desired frame rate for the video stream. The frame rate can only be set for the 24B1.3 cameras. The 20K1xx cameras are fixed at 30 fps. The 21K1xx cameras are fixed at 25 fps.

void SetCallback(PixelFormat, BufferDelegate, IntPtr)

The "SetCallback" function sets a callback routine to be called every time a new frame arrives from the camera.

Callbacks in managed code are handled by "delegates". The function to be called must be wrapped by a delegate object, and it is this delegate object that is handed to SetCallback. The PixelFormat parameter specifies the image format that the callback wishes to receive; it must be one of the pixel format enumerations above. The third parameter is a context value that is passed back to the callback function unchanged.

The callback function need not have returned by the time "Stop" is called, but all callbacks must return before "Close" will complete.

Here is a sample usage from C# code:

```
private void GoGraph()
{
   cw = new Videology.Camera();
   Videology.BufferDelegate bd = new Videology.BufferDelegate( Callback );

   try {
      cw.Open( 0 );
   }
   catch
   {
      Console.WriteLine( "No capture device." );
      return;
   }

   cw.SetFrameSize( FrameSize );
   cw.SetCallback( Videology.PixelFormat.pf_RGB24, bd, IntPtr.Zero );
   cw.Start( Videology.CameraMode.Live );
}

private void
Callback( int Context, IntPtr data, int size, int width, int height )
{
   …
}
```

property IntPtr WindowParent (set only)

This property sets the window that should be used as the parent window for the preview. The preview window is resized to match the size of the parent. If you have a Windows Forms window, you may pass its Handle property.

The top-level parent of this window will receive events from the DirectShow graph. DirectShow graph produces events as it plays, allowing applications to be notified when the graph starts and stops, or when certain errors occur. One of the more useful events is EC_DEVICE_LOST, which lets the application know that the capture source has been removed. In the case of a USB camera, this event is sent when the USB device is unplugged.

The wrapper will forward any such messages to the top-level parent of this window, with the message code WM_FGEVENTS, which is a constant defined in the Videology.Camera class. You can intercept these messages in your application by deriving your form class from IMessageFilter, and implementing the PreFilterMessage method, as in this example:

```
public class WrapperTest : Form, IMessageFilter
{
   …
```

```
    // IMessageFilter implementation.

    public bool PreFilterMessage(ref Message aMessage)
    {
        if (aMessage.Msg == Camera.WM_FGEVENTS)
        {
            if (aMessage.WParam.ToInt32() == EC_DEVICE_LOST)
            {
                StopGraph();
                MessageBox.Show("Camera was unplugged.");
                picVideo.Refresh();
            }
        }
        return false;
    }
}
```

void Start(CameraMode mode)

The "Start" function creates the full DirectShow graph and starts streaming. The "mode" parameter sets the operational mode from the enumeration shown previously.

void Stop()

The "Stop" function halts streaming and tears down the DirectShow graph. The capture device remains attached, so that you may manipulate its state, and you may start streaming again by calling the Start function.

void Trigger(IntPtr Context)

In Single Frame mode, "Trigger" forces an image to be taken immediately. You will probably want to set up an image callback to receive the image data before calling "Trigger". You may specify a Context value to pass in to the callback function; this will override the Context specified when you called SetCallback for this one frame. If you specify IntPtr.Zero, the Context from SetCallback will be used instead

void SetROI(System::Drawing::Rectangle rct)

The "SetROI" function sets a region of interest within the capture window. After this function, the frames sent to the callback function will only include the specified subrectangle. In the preview window, the region will be stretched (using a DirectDraw overlay) to fill the full frame size. So, if you set the resolution to 1024x768 and set the region of interest to (0,0,512,384), the upper left quadrant of the sensor will be stretched to fill the 1024x768 window.

Because of the nature of the native YUV format, the Left and Right members of the Rectangle will be rounded down to the nearest even value.

property int Gain

The "Gain" property sets or returns the gain of the video signal within the camera sensor. This is handled in hardware.

property Range GainRange

The "GainRange" property returns the range of permissible values for the Gain property. Currently, the range for all cameras is from 0 to 127.

property int Contrast

The "Contrast" property sets or returns the contrast of the video signal. Contrast is adjusted in software. The range is from 0 to 127. A value of 64 leaves the contrast as delivered from the camera.

property Range ContrastRange

The "ContrastRange" property returns the range of permissible values for the Contrast property.

property int Brightness

The "Brightness" property adjusts the black level of the video. The range is from 0 to 127. This property is handled in software. A value of 64 leaves the brightness as delivered from the camera. Values greater than 64 increase the brightness, and values below 64 reduce the brightness.

property Range BrightnessRange

The "BrightnessRange" property returns the range of permissible values for the Brightness property.

property double Gamma

The "Gamma" property sets or returns the gamma adjustment level of the video frames. Gamma is used to compensate for non-linearity in display devices. The valid range is from 0.5 to 2.0. This adjustment is handled in software. A value of 1.0 leaves the incoming frames untouched.

property RangeF GammaRange

The "GammaRange" property returns the range of permissible values for the Gamma property.

property int Exposure

The "Exposure" property sets the current exposure time for the camera. The parameter is specified as the inverse of the exposure time in seconds. The valid range is from 2 (1/2 second) to 50,000 (1/50000 second).

property bool Mirror

The "Mirror" property enables or disables horizontal mirroring of the image; that is, flipping left and right.

property bool Flip

The "Flip" property enables or disables vertical flipping of the image; that is, flipping top and bottom. This property is not available on the 20K1x or 21K1x cameras.

property bool ButtonState (get only)

The "ButtonState" returns the current state of the snapshot signal.

property int USBMode

The "USBMode" function modifies the handling of the video stream on the USB bus. The USB-C cameras are all capable of streaming video over a "bulk" pipe, or an "isochronous" pipe. The bulk pipe is capable of higher throughput, but it is negatively impacted by other traffic on the bus. The isochronous pipe has a lower maximum bandwidth, but the bandwidth is reserved and cannot be reassigned to other traffic. Setting 0 selects a bulk pipe, setting 1 selects an isochronous pipe. The setting is remembered in the registry for the future.

We recommend the isochronous pipe, unless you need a frame rate that cannot be sustained with the lower bandwidth.

This property cannot be changed while streaming. You must stop and restart streaming.

unsigned short ReadI2C(char SubDevice, char Register)

The "ReadI2C" function allows access to internal structures within the camera. Please see the individual camera application nodes for further information.

A SubDevice of 0 refers to the camera sensor itself. SubDevice 0xA0, 0xA2, 0xA4, and 0xA6 refer to the four pages of EEPROM available on the 20K1x, 21K1x, and 24C1.3 cameras.

void WriteI2C(char Device, char Register, unsigned short Data)

The "WriteI2C" function allows access to internal structures within the camera. Please see the individual camera application nodes for further information.

A SubDevice of 0 refers to the camera sensor itself. SubDevice 0xA0, 0xA2, 0xA4, and 0xA6 refer to the four pages of EEPROM available on the 20K1x, 21K1x, and 24C1.3 cameras.