Mightex
Simply Brighter

(In Canada)
2343 Brimley Road
Suite 868
Toronto, Ontario M1S 3L6
CANADA
Tel: 1-416-840 4991
Fax: 1-416-840 6541

(In US)
1032 Serpentine Lane
Suite 113
Pleasanton, CA 94566
USA
Tel: 1-925-218 1885
Email: sales@mightex.com

Mightex Classic USB DirectShow Filter Description

Version 1.0.3

Apr. 6, 2011

Relevant Products

Part Numbers
BCN-BG04-U, BCE-BG04-U, BTN-BG04-U, BTE-BG04-U, BCN-CG04-U, BCE-CG04-U, BCN-B013-U, BCE-B013-U, BTN-B013-U, BTE-B013-U, BCN-C030-U, BCE-C030-U,BCN-BG04-US, BCE-BG04-US, BTN-BG04-US, BTE-BG04-US, BCN-CG04-US, BCE-CG04-US, BCN-B013-US, BCE-B013-US, BTN-B013-US, BTE-B013-US, BCN-C030-US, BCE-C030-US

Revision History

Revision	Date	Author	Description
1.0.0	Dec. 08, 2007	YH Wu	Initial Revision
1.0.1	Jan. 16, 2009	YH Wu	Add SetGPIO
1.0.2	Apr. 16, 2010	JT Zheng	New Installation Description
1.0.3	Apr. 6, 2011	JT Zheng	MT_USBCamera_SDK_DS.dll File

The Mightex Classic Camera DirectShow filter enables Mightex Classic USB Camera works as standard Video Capture Device on MS Windows, this enables Mightex Classic Camera works seamlessly in most DirectShow aware applications.

The driver provides property pages and the COM interfaces for DirectShow applications to control Mightex Classic Camera. For applications to use the driver, it should add this Mightex filter to a filter graph, use the System Device Enumerator, which returns a unique moniker for the Mightex Classic Camera. The friendly name of the filter is **Mightex_ClassicSource**.

Note: while there're more than one cameras are attached, the filter is working with the first Mightex Classic camera enumerated on the USB Bus only.

Installation

The CD ROM contains directory "DirectShow" which has the following sub-directories:

\Filter
It includes the filter file (.ax file) and "RegisterServer.bat" and "unRegisterServer.bat"

\MightexClassicCameraEngine
It includes two DLL files which is Mightex Classic Camera Runtime Engine.

\Documents
It includes this document.

\Application
Under this directory, a DirectShow application which utilizes the filter (and the customized interface methods) is stored, user might use the application to preview the video and capture the video stream to AVI file. Full source code of this application is provided as an example to use the filter, user might develop his own application based on it.

Note: while capturing AVI file to disk, the actual frames might be less than the setting frame rate due to the bandwidth limitation of PC resources (for video compression and disk operation). E.g. While setting 24fps @1280x1024, it's actually not achievable on most of the PC, as PC can NOT afford enough resources to compress and store the image to Hard disk with the bandwidth at this setting.

Please follow the steps below for installing the filter:

1). Make sure the device driver is installed properly prior to using DirectShow features. For driver installation, please refer to "**Camera Quick Start Guide**".
To confirm successful driver installation**,** go to your hard disk (where you have copied the content from the CD) and find the folder named **/Application,** where you will see the file **MTApplication.** Please run the program by double clicking on it – after you have confirmed that the camera is working well, close the application window and go to step 2.

2). Now please go to your hard disk (where you have copied the content from the CD) and find the folder named **/DirectShow/MightexClassicCameraEngine**, where you will see two files named MT_*USBCamera_SDK_DS.dll* and MtUsbLib.*dl.* Please copy these two files to the directory named **WINDOWS\system32** (this is for 32bit windows, for X64 windows, user should copy the above two dlls into **WINDOWS\SysWOW64** directory).

3). Now you need to register DirectShow Filter. Please go to your hard disk (where you have copied the content from the CD) and find the folder named /**DirectShow/Filter**, where you will see: **RegisterServer.bat** , and run the **.bat** file by double clicking on it.

You should get a message with: Registration successful.

4) Confirm successful installation. Please go to your hard disk (where you have copied the content from the CD) and find the folder named /**DirectShow/Application,** where you will see **ClassicCameraApp.** Please run the program by double clicking on it, and you should now be able to control the camera using the software.

NOTE:

Please remember the camera has to be connected to the PC throughout the process.″

Filter Description

The Mightex_ClassicSource Video Capture Filter for Mightex Classic Camera has the following attributes, mainly it exposes the interfaces listed in the first row of the table below. The application can use **QueryInterface** to find all of these interfaces:

Filter Interfaces	**IAMStreamConfig**, **IAMVideoProcAmp**, **IAMVideoControl**, **IMightex_ClassicSource**, **ISpecifyPropertyPages**.
Input Pin Media Types	No input pin. Only one output pin.
Input Pin Interfaces	No input pin. Only one output pin.
Output Pin Media Types	MEDIATYPE_Video (RGB24)
Output Pin Interfaces	**IKsPropertySet**, **ISpecifyPropertyPages**
Filter CLSID	CLSID_Mightex_ClassicSource
Property Page CLSID	Driver-dependent.
Plug-in Executable	Mightex_ClassicSource.ax
Merit	MERIT_DO_NOT_USE
Filter Category	CLSID_VideoInputDeviceCategory

Interfaces:

IMightex_ClassicSource

The **IMightex_ClassicSource** interface is a customized interface defined by Mightex. The interface enables user to have detailed and efficient controls on the parameters of the Mightex Classic camera.

With this interface, User might set video format properties, such as the output dimensions and frame rate. Also use this interface to set the exposure time, the X, Y Start position (at a certain resolution) and the Red, Green and Blue pixels' gains. The interface also enables an application to adjust the qualities of an incoming video signal, such as brightness, contrast, gamma, and sharpness. The interface also enables user to flip a picture horizontally and/or vertically, set the camera to "Trigger" mode which can capture image while an external trigger asserts.

In addition to the methods inherited from **IUnknown**, the **IMightex_ClassicSource** interface exposes the following methods.

Method	Description
SetStreamFrameRate	User may set the frame rate by invoking this function, this actually set previewing and capturing video frame rate precisely.
SetPhysicalCameraFrameRate	User may set the frame rate by invoking this function, this actually set camera physical frame rate.
GetPhysicalCameraFrameRate	Retrieves the actual frame rate at which the Camera is streaming. This method is used with the Camera, where the maximum frame rate can be limited by bandwidth availability. The actual frame rate is affected by some factors, such as resolution itself, exposure time, maximum vertical blanking time of a frame…etc .This is only available during video streaming.
GetModuleNoSerialNo	For a present camera device, user might get its Module Number and Serial Number by invoking this function.
SetCameraWorkMode	By default, the Camera is working in "NORMAL" mode in which camera deliver frames to Host continuously, however, in some applications, user may set it to "TRIGGER" Mode, in which the camera is waiting for an external trigger signal and capture ONE frame for each trigger signal.
SetResolution	User may set the width / height size by

	invoking this function.
SetExposureTime	User may set the exposure time by invoking this function.
SetXYStart	User may set the X, Y Start position (at a certain resolution) by invoking this function.
SetGains	User may set the Red, Green and Blue pixels' gain by invoking this function.
SetGamma	User may set the Gamma, Contrast, Brightness and Sharpness level for Camera by invoking this function.
SetBWMode	User may set the processed Bitmap data as "Black and White" mode, "Horizontal Mirror" and "Vertical Flip" for camera by invoking this function.
Snapshot	User may grab still images from the stream and store in files by invoking this function.
GetCameraControl	User may get the camera control and global control parameters by invoking this function.
GetLastBMPFrame	User may call this function to get the bitmap format frame of the last captured frame.
GetCurrentFrame	User may call this function to get a frame.
SetGPIOConifg	User may call this function to configure GPIO pins.
SetGPIOInOut	User may call this function to set GPIO output pin states and read the input pins states.

HEADER FILE:

The “IMightex_ClassicSource.h” is as following:

```
//-------------------------------------------------------------------------
// File: IMightex_ClassicSource.h
//
// Desc: DirectShow code - custom interface allowing the user
//             to do some settings.
//
// Copyright (c) Mightex Corporation.    All rights reserved.
//-------------------------------------------------------------------------


#ifndef __H_IMightex_ClassicSource__
#define __H_IMightex_ClassicSource__

#ifdef __cplusplus
extern "C" {
#endif

      // {2A7D10A2-186F-4e34-8532-B5FBDE4739EE}
      DEFINE_GUID(CLSID_Mightex_ClassicSource,
            0x2a7d10a2, 0x186f, 0x4e34, 0x85, 0x32, 0xb5, 0xfb, 0xde, 0x47, 0x39, 0xee);
      // {5822540F-0385-461c-8CCF-792D1E8D0518}
      DEFINE_GUID(IID_IMightex_ClassicSource,
            0x5822540f, 0x385, 0x461c, 0x8c, 0xcf, 0x79, 0x2d, 0x1e, 0x8d, 0x5, 0x18);

      struct CCameraControl;
      struct CCameraGlobalControl;

      DECLARE_INTERFACE_(IMightex_ClassicSource, IUnknown)
      {
            STDMETHOD(SetStreamFrameRate) (THIS_
                  LONGLONG frameRate
                  ) PURE;
            STDMETHOD(SetPhysicalCameraFrameRate) (THIS_
                  int frameRateLevel
                  ) PURE;
            STDMETHOD(GetPhysicalCameraFrameRate) (THIS_
                  LONGLONG &actualFrameRate
                  ) PURE;
            STDMETHOD(GetModuleNoSerialNo) (THIS_
                  char *ModuleNo, char *SerialNo
                  ) PURE;
            STDMETHOD(SetCameraWorkMode) (THIS_
                  int WorkMode
                  ) PURE;
            STDMETHOD(SetResolution) (THIS_
                  int width, int height, int bin
                  ) PURE;
            STDMETHOD(SetExposureTime) (THIS_
                  int exposureTime
                  ) PURE;
            STDMETHOD(SetXYStart) (THIS_
```

```
            int xStart, int yStart
            ) PURE;
        STDMETHOD(SetGains) (THIS_
            int redGain, int greenGain, int blueGain
            ) PURE;
        STDMETHOD(SetGamma) (THIS_
            int Gamma, int Contrast, int Bright, int Sharp
            ) PURE;
        STDMETHOD(SetBWMode) (THIS_
            int BWMode, int H_Mirror, int V_Flip
            ) PURE;
        STDMETHOD(Snapshot) (THIS_
            char * TargetFile, BOOL SaveAsJPEG, BOOL AppendDataTime, int SaveFileCount
            ) PURE;
        STDMETHOD(GetCameraControl) (THIS_
            CCameraControl &cameraCtrl, CCameraGlobalControl &cameraGlobalCtl
            ) PURE;
        STDMETHOD(GetLastBMPFrame) (THIS_
            char *FileName
            ) PURE;
        STDMETHOD(GetCurrentFrame) (THIS_
            unsigned char *Buffer
            ) PURE;
        STDMETHOD(SetGPIOConifg) (THIS_
            unsigned char ConfigByte
            ) PURE;
        STDMETHOD(SetGPIOInOut) (THIS_
            unsigned char OutputByte, unsigned char *InputBytePtr
            ) PURE;
    };

#ifdef __cplusplus
}
#endif

#endif // __ICONTRAST__
```

HRESULT SetStreamFrameRate (LONGLONG frameRate);

User may set the frame rate by invoking this function, this sets previewing and capturing video frame rate precisely. User may also use **IAMStreamConfig**::**SetFormat** to do this.

Argument:

frameRate – The frame rate is expressed as frame duration in 100-nanosecond units.

Return: Always return NOERROR.

HRESULT SetPhysicalCameraFrameRate (int frameRateLevel);

User may set the physical camera frame rate by invoking this function.

Argument:

frameRateLevel –Can be from 0 – 10, while 0 means the lowest frame rate, 10 means the highest rate. In the current design, the actual frame rate is mainly depending on the PC resources. The frame rate might be different on a slow PC and a fast PC, the default setting of the camera engine is to set the Maximum frame rate, however, that might not be ideal as almost all the CPU resources will be used by camera engine, which will make other PC applications "hunger" of CPU time, user might want to reduce the frame rate a little bit to politely give other application time to run.

Return: E_FAIL: If the function fails (e.g. invalid device number)
NOERROR: if the call succedds.

HRESULT GetPhysicalCameraFrameRate (LONGLONG &actualFrameRate);

Retrieves the actual camera frame rate at which the Camera is streaming. This method is used with the Camera, where the maximum frame rate can be limited by bandwidth availability. The actual frame rate is affected by some parameters, such as resolution itself, exposure time, maximum vertical blanking time of a frame…etc. This is only available during video streaming. User might also use **IAMVideoControl**::**GetCurrentActualFrameRate** for the similar purpose.

Argument:

actualFrameRate – Pointer to the frame rate in frame duration in 100-nanosecond units.

Return: Always return NOERROR.

Note:

With the customized interface, user might set the frame rate in two levels:

***.** At the physical device level, user might set the frame rate on the camera itself, so the camera will generate frames at this setting interval, please also note that the actual camera frame rate might not exactly the one user set to it, as it might also be affected by other parameters such as resolution and exposure time..etc.

***.** At Filter level, user might set a frame rate for the generating of the video stream, for example, this frame rate can be set to 30fps always, no matter what the actual camera frame rate it, the filter will do automatic frame insertion or ignorance according to user's settings.

HRESULT GetModuleNoSerialNo(char *ModuleNo, char *SerialNo);

For any present device, user might get its Module Number and Serial Number by invoking this function.

Argument:

ModuleNo – the pointer to a character Classic, the Classic should be available for at least 16 characters.

SerialNo – the pointer to a character Classic, the Classic should be available for at least 16 characters.

Return: Always return NOERROR.

HRESULT SetCameraWorkMode (int WorkMode);

By default, the Camera is working in "NORMAL" mode in which camera deliver frames to Host continuously, however, in some applications, user may set it to "TRIGGER" Mode, in which the camera is waiting for an external trigger signal and capture ONE frame for each trigger signal. User might also use **IAMVideoControl**::**SetMode** to do this.

Argument:

WorkMode – 0: NORMAL Mode, 1: TRIGGER Mode.

Return: E_FAIL: If the function fails (e.g. invalid device number)
NOERROR: if the call succeeds.

Important:

NORMAL mode and **TRIGGER** mode have the same features, but:

NORMAL mode – Camera will always grab frame as long as there's available Classic for a new frame, For example, when host (in most cases, it's a PC) is keeping to get frame from the camera, the camera is continuously grabbing frames from CMOS sensor.

TRIGGER mode – Camera will only grab a frame from CMOS sensor while there's an external trigger asserted (and there's available Classic for a new frame).

HRESULT SetResolution (int width, int height, int bin);

User may set the width / height size by invoking this function. Also use **IAMStreamConfig**::**SetFormat**.

Argument:

width– the customer defined width size.

height– the customer defined height size.

bin – 0: Normal mode, 1: 1:2 decimation mode.

Return: E_FAIL: If the function fails (e.g. invalid device number)

E_INVALIDARG: The customer defined width or height size is out of range.

The valid range for width / height sizes are:

Width: 4 – Maximum width Size (Camera dependant, refer to camera spec.)

Height: 4 – Maximum height Size (Camera dependant, refer to camera spec.)

Note: While in bin mode (bin is "1"), the minimum of width and height Size should be 8, while bin is "0", width and height can be 4.

E_PROP_ID_UNSUPPORTED: The granularity and alignment of customer defined width / height size is 4, otherwise it return –3.

NOERROR: if the call succedds.

HRESULT SetExposureTime (int exposureTime);

User may set the exposure time by invoking this function.

Argument:

exposureTime – the Exposure Time is set in 50 Microsecond UNIT, e.g. if it's 4, the exposure time of the camera will be set to 200us. The valid range of exposure time is 50us – 750ms, So the exposure time value here is from 1 – 15000.

Return: E_FAIL: If the function fails (e.g. invalid device number)

NOERROR: if the call succedds.

HRESULT SetXYStart (int xStart, int yStart);

User may set the X, Y Start position (at a certain resolution) by invoking this function.

Argument:

Xstart, YStart – the start position of the ROI (in pixel).

Return: E_FAIL: If the function fails (e.g. invalid device number)

NOERROR: if the call succedds.

Important: While the resolution is NOT set to the Maximum, the setting Region Of Interesting (ROI) is an active region of the sensor pixels, user may use this function to set the left top point of the ROI. Note that under a certain ROI, the Xstart and Ystart have a valid settable range. If any value out of this range is set to device, device firmware will
check it and set the closest valid value.

HRESULT SetGains (int redGain, int greenGain, int blueGain);

User may set the Red, Green and Blue pixels' gain by invoking this function.

Argument:

redGain, greenGain, blueGain – the gain value to be set.

Return: E_FAIL: If the function fails (e.g. invalid device number)

NOERROR: if the call succedds.

Important: The gain value should be from 1 – 64, represents 0.25x – 8x of analog amplifying Multiples. For any value of the valid range, device will handle it by setting the closet valid value. For Monochrome sensor modules and the XXX-XG04-U modules (with MT9V032 sensor), only GreenGain value will be used by the camera firmware. Note: For setting proper exposure for an image, it's recommended to adjust exposure time prior to the gain, as setting high gain will increase the noise (Gain is similar to the ISO settings in traditional consumer camera), for applications which the SNR is important, it's recommended to set Gain not more than 32 (4x).

HRESULT SetGamma (int Gamma, int Contrast, int Bright, int Sharp);

User may set the Gamma, Contrast, Brightness and Sharpness level for ALL Cameras by invoking this function. Also use **IAMVideoProcAmp**::**Set**.

Argument:

Gamma – Gamma value, valid range 1 – 20, represent 0.1 – 2.0

Contrast – Contrast value, valid range 0 – 100, represent 0% -- 100%.

Bright – Brightness value, valid range 0 – 100, represent 0% -- 100%.

Sharp – Sharp Level, valid range 0 – 3, 0: No Sharp, 1: Sharp, 2: Sharper, 3: Sharpest.

Return: E_FAIL: If the function fails,

NOERROR: if the call succedds.

Important: In most cases, the default values (Gamma = 1.0, Contrast and Brightness = 50%, Sharp = 0) are proper.

HRESULT SetBWMode (int BWMode, int H_Mirror, int V_Flip);

User may set the processed Bitmap data as "Black and White" mode, "Horizontal Mirror" and "Vertical Flip" for ALL Cameras by invoking this function. Also use **IAMVideoControl**::**SetMode**.

Argument:

BWMode : 0: Normal, 1: Black and White mode.

H_Mirror: 0: Normal, 1: Horizontal Mirror.

V_Flip: 0: Normal, 1: Vertical Flip.

Return: E_FAIL: If the function fails,
NOERROR: if the call succedds.

HRESULT Snapshot (char *TargetFile, BOOL SaveAsJPEG, BOOL AppendDataTime, int SaveFileCount);

User may grab still image and store them to files by invoking this function.

Argument:

TargetFile: The target files' path and name.
SaveAsJPEG: 0: BMP file, 1: JPG. File.
AppendDataTime: 0: Normal, 1: The files name append date and time.
SaveFileCount:The files count.

Return: Always return NOERROR.

HRESULT GetCameraControl (CCameraControl &cameraCtrl, CCameraGlobalControl &cameraGlobalCtl);

User may get the camera control and global control parameters by invoking this function.

Argument:

cameraCtrl: The Camera Control Parameters structure variable.
cameraGlobalCtl: The Camera Global Control Parameters structure variable.

Return: Always return NOERROR.

HRESULT GetLastBMPFrame(char *FileName);

User may call this function to get the bitmap format frame of the last captured frame.

Argument:

FileName: the full file name for the bitmap file. While the frame grabbing is running OR it's stopped, we can always get the last (for the time this function is invoking) frame of the Video window in Bitmap format. Note that this function may mainly be used in situation of user stop the video as the frame is exactly the user's interesting. Note that this bitmap frame is adjusted with user's setting of Gamma, contrast, bright and sharp level, if user wants to get un-adjusted image data, user might invoke **SetGamma()** function with Gamma set to 10 (mean 1.0), contrast and bright set to 50 (mean 50%) and SharpLevel set to 0 (mean Normal).

Return: E_FAIL: If the function fails,
NOERROR: if the call succedds.

HRESULT GetCurrentFrame(unsigned char *Buffer);

User may call this function to get a frame.

Argument:

Buffer: Byte buffer to hold the whole frame of image. This function can only be invoked while the frame grabbing is started, user might want to get a frame of image in memory, instead of in a file. This function will put the current frame into Buffer.

Return: E_FAIL: If the function fails,
NOERROR: if the call succedds.

HRESULT SetGPIOConifg (unsigned char ConfigByte);

User may call this function to configure GPIO pins.

Argument:

ConfigByte – Only the 4 LSB are used, bit0 is for GPIO1, bit1 is for GPIO2 and so on. Set a certain bit to 1 configure the corresponding GPIO to output, otherwise it's input.

Return: E_FAIL: If the function fails,
NOERROR: if the call succedds.

Important: Note that This function can only be invoked if the camera is with built in GPIO (MCN, GLN and MCE).

HRESULT SetGPIOInOut (unsigned char OutputByte, unsigned char *InputBytePtr);

User may call this function to set GPIO output pin states and read the input pins states.

Argument :

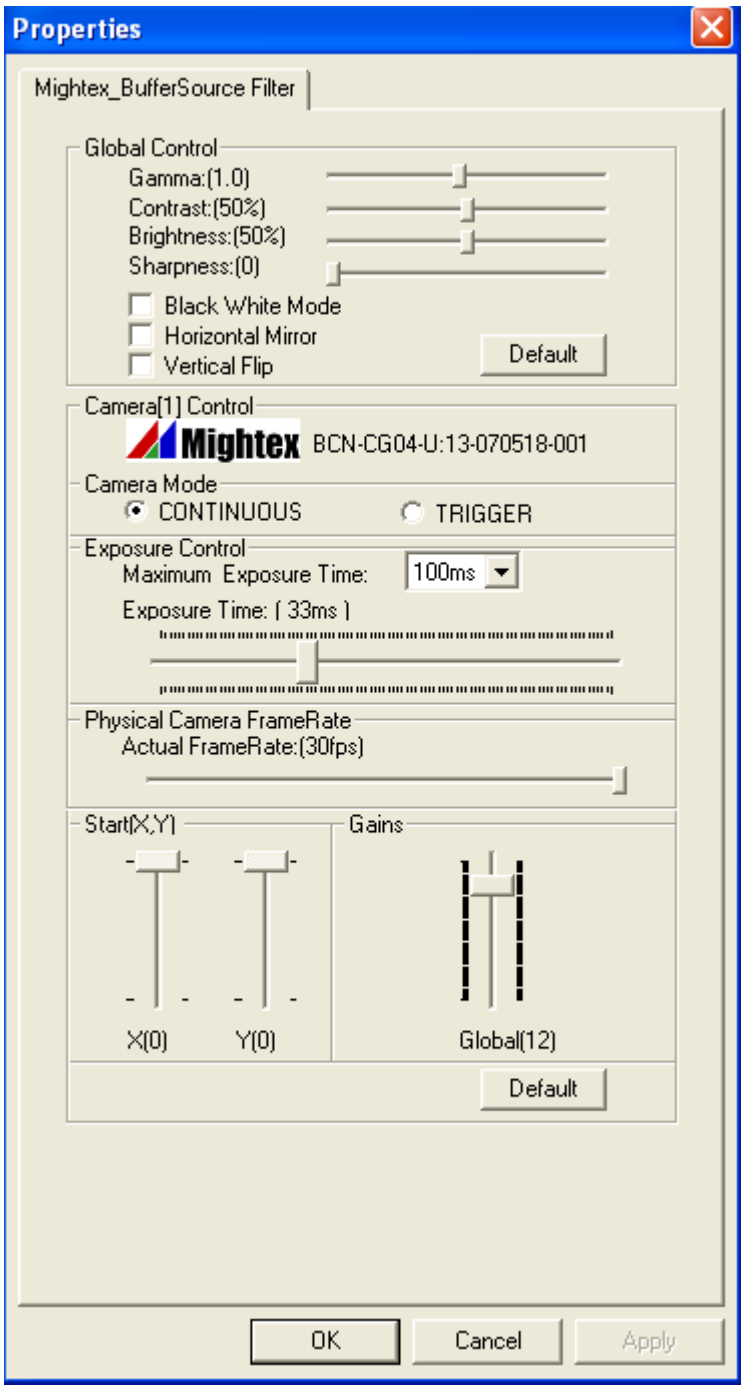
OutputByte – Only the 4 LSB are used, bit0 is for GPIO1, bit1 is for GPIO2 and so on. Set a certain bit to 1 will output High on the corresponding GPIO pin, otherwise it outputs Low. Note that it's only working for those pins are configured as "Output".

InputBytePtr – the Address of a byte, which will contain the current Pin States, only the 4 LSB bits are used, note that even a certain pin is configured as "output", we can still get its current state.

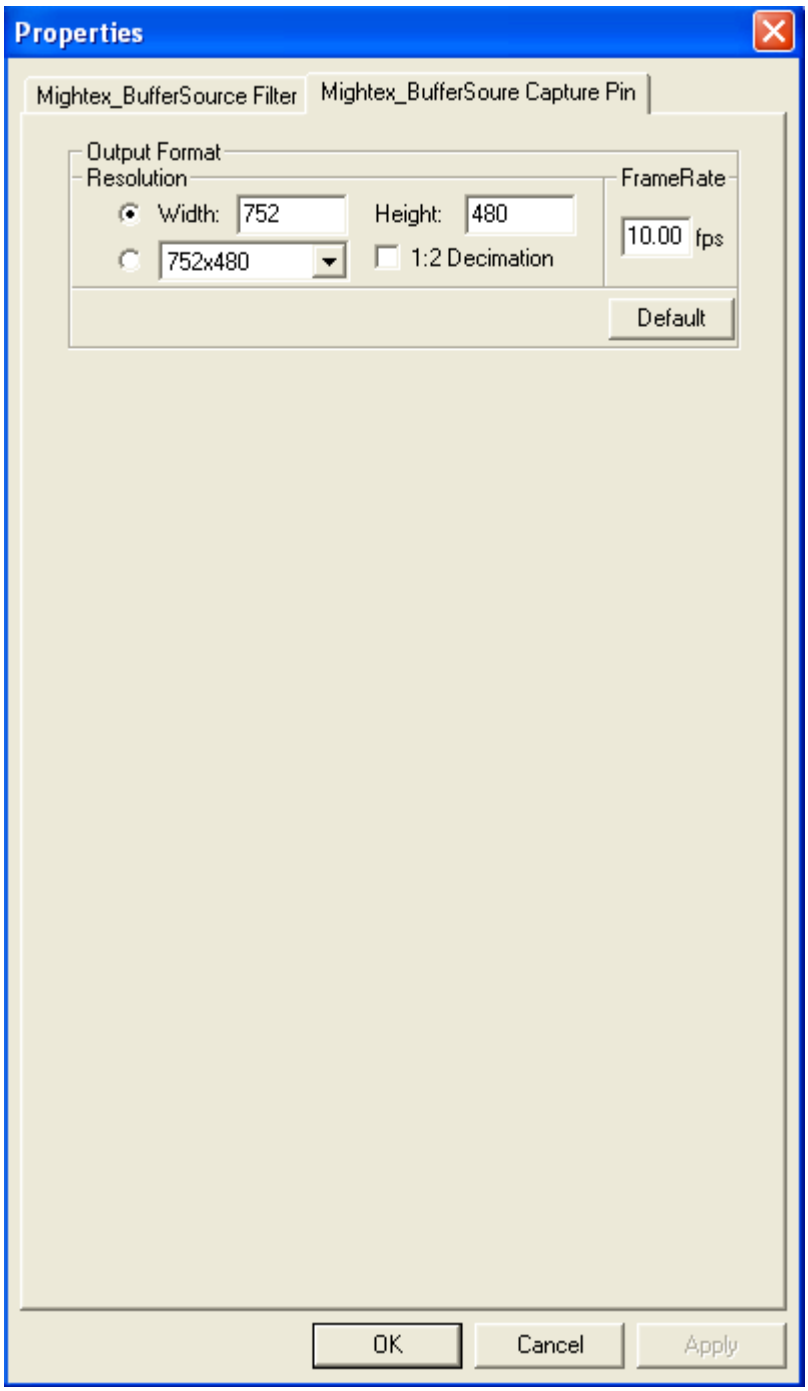
Return: E_FAIL: If the function fails,
NOERROR: if the call succedds.

Important: Note that This function can only be invoked if the camera is with built in GPIO (MCN, GLN and MCE).

The interfaces of property pages:



Note: In this example, the camera is CG04, so it only has Global Gain Control, for 3M Color camera, there're individual gain control for Red, Green and Blue.


Properties
Mightex_BufferSource Filter
Mightex_BufferSoure Capture Pin
Output Format
Resolution
Width:
752
Height:
480
752x480
1:2 Decimation
FrameRate
10.00
fps
Default
OK
Cancel
Apply

All the following interfaces are standard DirectShow interfaces for video capture device, please refer to Microsoft DirectShow documents for the detailed description of the interfaces and their methods.

IAMStreamConfig Interface

The **IAMStreamConfig** interface enables an application to set the output format on certain capture and compression filters, for both audio and video. In our case, it's a video capture filter.

Use this interface to set format properties, such as the output dimensions and frame rate (for video) or the sample rate and number of channels (for audio).

Filters expose this interface on their output pins. To use the interface, enumerate the filter's pins and query for **IAMStreamConfig**. Or, if you are using the Capture Graph Builder object to build the filter graph, you can call the **ICaptureGraphBuilder2::FindInterface** method.

In addition to the methods inherited from **IUnknown**, the **IAMStreamConfig** interface exposes the following methods.

Method	Description
GetFormat	Retrieves the current or preferred output format.
GetNumberOfCapabilities	Retrieves the number of format capabilities that this pin supports.
GetStreamCaps	Retrieves a set of format capabilities.
SetFormat	Sets the output format on the pin.

IAMVideoProcAmp Interface

The **IAMVideoProcAmp** interface enables an application to adjust the qualities of an incoming video signal, such as brightness, contrast, hue, saturation, gamma, and sharpness.

The Video Capture filter exposes this interface if the hardware supports image adjustment.

In addition to the methods inherited from **IUnknown**, the **IAMVideoProcAmp** interface exposes the following methods.

Method	Description
GetRange	Retrieves minimum, maximum, and default values for setting properties.
Set	Sets video quality for a specified property.
Get	Retrieves video quality for a specified property.

IAMVideoControl Interface

The **IAMVideoControl** interface enables you to flip a picture horizontally and/or vertically, set up a stream so it can capture from an external trigger (such as a camera button that the user pushes), simulate an external trigger in software, and list the available frame rates.

Use this interface when your application runs on hardware that is capable of flipping and external triggering. For Mightex Classic camera, it's capable to be set to Trigger mode, for details of Trigger mode, please refer to Mightex Classic Camera User manual.

In addition to the methods inherited from **IUnknown**, the **IAMVideoControl** interface exposes the following methods.

Method	Description
GetCaps	Retrieves the capabilities of the underlying hardware.
SetMode	Sets the video control mode of operation.
GetMode	Retrieves the video control mode of operation.
GetCurrentActualFrameRate	Retrieves the actual frame rate at which the device is streaming. This method is used with devices, such as the Universal Serial Bus (USB) or cameras that use the IEEE 1394 serial standard, where the maximum frame rate can be limited by bandwidth availability. This is only available during video streaming.
GetMaxAvailableFrameRate	Retrieves the maximum frame rate currently available based on bus bandwidth usage for connections such as USB and IEEE 1394 camera devices where the maximum frame rate can be limited by bandwidth availability.
GetFrameRateList	Retrieves a list of available frame rates.

Note that the last two methods are NOT implemented on Mightex filter.

IKsPropertySet Interface

The **IKsPropertySet** interface was originally designed as an efficient way to set and retrieve device properties on WDM drivers, using KSProxy to translate the user-mode COM method calls into the kernel-mode property sets used by WDM streaming class drivers. This interface is now also used to pass information strictly between software components.

In some cases, software components must implement either this interface, or else the IKsControl interface (documented in the DirectShow DDK). For example, if you are writing a software MPEG-2 decoder for use with the Microsoft® DVD Navigator, you must implement one of these interfaces and also support the DVD-related property sets that the Navigator will send to the decoder. Pins may support one of these interfaces to allow other pins or filters to set or retrieve their properties.

Note Another interface by this name exists in the dsound.h header file. The two interfaces are not compatible. The **IKsControl** interface, documented in the DirectShow DDK, is now the recommended interface for passing property sets between WDM drivers and user mode components.

Methods in Vtable Order

In addition to the methods inherited from **IUnknown**, the interface exposes the following methods.

Method	Description
Set	Sets a property identified by a property set **GUID** and a property ID.
Get	Retrieves a property identified by a property set **GUID** and a property ID.
QuerySupported	Determines whether an object supports a specified property set.