



ActiveUSB SDK

User Guide

Version 2.1

Copyright © 2014-2017 by A&B Software LLC

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form by any means, electronic, mechanical, by photocopying, recording , or otherwise, without the prior written permission of A&B Software, LLC

Information furnished by A&B Software LLC is believed to be accurate and reliable; however, no responsibility is assumed by A&B Software LLC for its use; nor for any infringements of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent rights of A&B Software LLC.

Use, duplication, or disclosure by the United States Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer software clause at 48 C.F.R, 252.227-7013, or in subparagraph (c)(2) of the Commercial Computer Software - Registered Rights clause at 48 C.F.R, 52-227-19 as applicable.

ActiveUSB is a trademark of A&B Software LLC.

All other brand and product names are trademarks or registered trademarks of their respective companies.

Second Edition

May 2017

*A&B Software LLC
New London, CT 06320
USA*

www.ab-soft.com
support@ab-soft.com

Table of Contents

Part I Introduction	7
1 License Agreement	9
2 System Requirements	11
3 Installation	12
4 Driver setup	15
5 Registration	21
6 USB3 Vision configurator	22
7 UcamViewer	25
8 Distributing your application	29
Part II Getting started	29
1 Visual Basic	30
2 Visual C++	32
3 VB.NET	36
4 Visual C#	40
5 Using ActiveUSB API at runtime	44
6 Working with multiple cameras	47
Part III ActiveX Reference	47
1 Properties	48
Acquire	54
AcquisitionFrameCount	56
AcquisitionFrameRate	58
AcquisitionMode	59
Affinity	61
Alpha	62
AntiTearing	64
BackColor	66
BalanceRatio	67
BalanceRatioSelector	69
BalanceWhiteAuto	71
Bayer	73
BinningX	75
BinningY	76
BkgCorrect	77
BkgName	79
BlackLevel	80
BlackLevelAuto	82
BlackLevelSelector	84
Camera	86
ColorCorrect	88

DecimationX	89
DecimationY	90
Display	91
Edge	93
ExposureAuto	94
ExposureMode	96
ExposureTime	98
Flip	100
Font	102
Format	103
Gain	106
GainAuto	108
GainSelector	110
Gamma	112
HotPixelCorrect	114
HotPixelLevel	115
Integrate	116
IntegrateWnd	117
LensCorrect	118
LineFormat	120
LineInverter	122
LineMode	124
LineSelector	126
LineSource	128
LUTMode	130
Magnification	131
MonitorSync	133
Overlay	135
OverlayColor	136
OverlayFont	137
Palette	138
Rotate	140
ScrollBars	142
ScrollX	144
ScrollY	145
SizeX	146
SizeY	147
OffsetX	148
OffsetY	149
Timeout	150
TestImageSelector	151
Trigger	153
TriggerActivation	155
TriggerDelay	157
TriggerSelector	159
TriggerSource	161
UserOutputSelector	163
UserOutputValue	165
UserSetSelector	166
WebStream	168
2 Methods	170
CloseVideo	182
CreateSequence	183
CreateVideo	185

Draw	187
DrawAlphaClear	188
DrawAlphaEllipse	189
DrawAlphaLine	191
DrawAlphaPixel	193
DrawAlphaRectangle	194
DrawAlphaText	196
DrawEllipse	198
DrawLine	200
DrawPixel	201
DrawRectangle	202
DrawText	204
GetAcquisitionFrameRateMax	205
GetAcquisitionFrameRateMin	206
GetAudioLevel	207
GetAudioList	208
GetAudioSource	209
GetBalanceRatioMax	210
GetBalanceRatioMin	211
GetBarcode	212
GetBitsPerChannel	214
GetBlackLevelMax	216
GetBlackLevelMin	217
GetBlockId	218
GetBytesPerPixel	219
GetCameraIP	220
GetCameraList	221
GetCameraMAC	223
GetCategoryList	224
GetChunkPointer	225
GetChunkSize	227
GetCodec	229
GetCodecList	230
GetCodecProperties	231
GetComponentData	232
GetComponentLine	234
GetDIB	236
GetEnumList	238
GetExposureTimeMax	240
GetExposureTimeMin	241
GetFeature	242
GetFeature64	244
GetFeatureAccess	246
GetFeatureArray	247
GetFeatureDependents	249
GetFeatureDescription	250
GetFeatureList	251
GetFeatureIncrement	253
GetFeatureMin	255
GetFeatureMax	257
GetFeatureRepresentation	259
GetFeatureString	260
GetFeatureTip	262
GetFeatureType	263

GetFeatureVisibility	265
GetFileAccessMode	266
GetFileList	267
GetFileSize	269
GetFileTransferProgress	271
GetFormatList	273
GetFPSAcquired	275
GetFPS	276
GetGainMax	277
GetGainMin	278
GetHistogram	279
GetImageData	281
GetImageLine	283
GetImagePointer	285
GetImageStat	287
GetImageWindow	289
GetLevels	291
GetLUT	292
GetOptimalPacketSize	293
GetPicture	295
GetPixel	297
GetRawData	299
GetRGBPixel	301
GetROI	303
GetSequenceFrameCount	304
GetSequencePicture	305
GetSequencePixel	306
GetSequencePointer	308
GetSequenceRawData	310
GetSequenceTimestamp	312
GetSequenceWindow	313
GetTimestamp	315
GetTriggerDelayMax	316
GetTriggerDelayMin	317
GetVideoFPS	318
GetVideoFrameCount	319
GetVideoPosition	320
GetVideoVolume	321
Grab	322
IsFeatureAvailable	323
LoadImage	324
LoadSettings	325
LoadSequence	327
OpenVideo	328
OverlayClear	330
OverlayEllipse	331
OverlayLine	332
OverlayPixel	333
OverlayRectangle	334
OverlayText	335
PlayVideo	336
ReadBlock	338
ReadFile	340
ReadRegister	342

SaveBkg	343
SaveImage	345
SaveSettings	347
SaveSequence	349
SetAudioLevel	351
SetAudioSource	352
SetCodec	354
SetCodecProperties	356
SetColorMatrix	358
SetImageWindow	360
SetFeature	362
SetFeature64	364
SetFeatureArray	366
SetFeatureString	368
SetGains	370
SetLensDistortion	372
SetLevels	374
SetLUT	377
SetROI	379
SetVideoFPS	381
SetVideoPosition	382
SetVideoSync	384
SetVideoVolume	386
SetWebStreamer	387
ShowAudioDlg	389
ShowCodecDlg	390
ShowCompressionDlg	391
ShowProperties	393
SoftTrigger	395
StartCapture	396
StopCapture	398
StartSequenceCapture	399
StopSequenceCapture	401
StartVideoCapture	402
StopVideoCapture	404
StopVideo	405
TriggerVideo	406
WriteBlock	407
WriteFile	409
WriteRegister	411
3 Events	412
CameraPlugged	414
CameraUnplugged	415
CaptureCompleted	416
EventMessage	417
EventDataMessage	419
FormatChanged	421
FrameAcquired	422
FrameAcquiredX	423
FrameDropped	425
FrameLoaded	426
FrameRecorded	427
FrameReady	428
MouseDbClick	430

MouseDown	431
MouseDownRight	432
MouseMove	434
MouseUp	435
MouseUpRight	436
PlayCompleted	437
RawFrameAcquired	438
Scroll	439
Timeout	440
4 Property Pages	441
Source	442
Format	444
Analog	446
Inp/Out	448
GenlCam	450
Display	452
Part IV DirectShow	455
1 Quick Reference Guide	456
FilterConfig utility	458
Retrieving the Filter	459
Building the Graph	461
Displaying the Preview	462
Capturing to AVI	463
Getting the Image Data	464
Displaying Property Pages	466
2 Interfaces	467
IAMCameraControl	468
IAMVideoProcAmp	470
IAMVideoControl	472
IActiveGige	474
IAMStreamConfig	475
IAMVideoCompression	477
IAMDroppedFrames	479
ISpecifyPropertyPages	480
Part V TWAIN	481
Part VI Samples	483
Part VII Camera list	486
Part VIII Troubleshooting	487
Index	491

1 Introduction

ActiveUSB is an SDK and ActiveX control designed for rapid application development tools, such as Visual Basic, VB.NET, Visual C++, C#, Java, Delphi, Python, Matlab, etc. Provided is GigeViewer application allowing customers to operate multiple cameras and save images in a number of formats. Also included are TWAIN and DirectShow drivers for interfacing to third-party imaging and video capture software. With *ActiveUSB* your application immediately supports USB3 Vision™ compliant cameras.

In general, with *ActiveUSB* you can :

- Create 32-bit applications for the 32- and 64-bit Windows, as well as native 64-bit applications for the 64-bit Windows.
- Acquire and display live video from one or several USB3 Vision™ cameras.
- Select among multiple camera sources.
- Stream video from a single camera to several computers and applications using the Multicast mode.
- Set a desired video format and triggering mode.
- Select among several hardware and software trigger sources.
- Grab 8- and 16-bit monochrome images, or 24- and 48-bit color images.
- Perform automatic color interpolation of a monochrome video generated by Bayer cameras.
- Select the desired size and position of the scan area.
- Flip and rotate the live image.
- Adjust multiple camera features in real time.
- Activate automatic or one-push control over selected camera features, such as exposure and white balance.
- Save camera settings in the system registry and reload them on demand.
- Control non-standard camera features through direct access to camera registers.
- Transfer data to and from files hosted on the camera.
- Choose among several palettes for pseudo-color display.
- Get an instant access to pixel values and pixel arrays.
- Retrieve individual color planes from RGB images.
- Retrieve chunk data appended to each image.
- Receive message events from cameras in real time.
- Import live video to a PictureBox object.
- Perform image processing on captured frames and display processed video in real-time.
- Perform real-time histogram and statistical analysis over a selected color component.
- Implement real-time background correction over the dark and bright fields.
- Automatically identify hot-pixels and eliminate them from incoming images.
- Perform the running average and integration of incoming video frames.
- Correct barrel and pincushion lens distortion in real time.
- Apply custom LUTs (lookup tables) to incoming video frames.
- Apply 3x3 color correction matrix to incoming video frames.
- Perform manual and automatic Window/Level processing (brightness, contrast, white balance).
- Save images in RAW, BMP, TIF, JPEG and DPX formats with adjustable compression.
- Load and display images in BMP, TIF and JPEG formats.
- Decode 1D and 2D barcodes (UPC-A, UPC-E, EAN-8, Code 128, Code 39, Interleaved 2/5, QR Code, DataMatrix, PDF417).
- Perform time-lapse capture to an AVI file or series of sequentially-named images.
- Select a video compression codec for the AVI recording and adjust its settings.
- Scroll and zoom the live video with the full screen option.
- Overlay custom graphic and texts on the live video.
- Draw multi-colored graphics and texts with an adjustable transparency over the live video.

- Synchronize video rendering with the monitor refresh rate to eliminate the tearing artifact.
- Interface to third-party imaging applications using the included TWAIN driver.
- Interface to DirectShow-based applications via the included Video Capture Source filter.

With the extended *ActiveUSB DVR* version you can:

- Record multiple AVI files with the sound.
- Reserve a space for AVI files to eliminate dropped frames.
- Play back AVI files with an adjustable speed, direction and frame interval.
- Browse through the frames in an AVI file with the full access to recorded pixel values.
- Use a proprietary raw uncompressed codec to record and play back raw Bayer video with no quality degradation.
- Control the recording and play-back volume.
- Record the incoming video into a memory sequence.
- Perform a loop recording.
- Play back the recorded memory sequence with an adjustable speed, direction and frame interval.
- Get an instant access to pixel values and timestamp of each frame in the memory sequence.
- Stream incoming video to your network in H.264 RTSP format and view it on a remote desktop or mobile device.

ActiveUSB uses multiple threads to support video acquisition, therefore it does not require separate components for thread management.

This document gives a detailed description of *ActiveUSB*, its properties and methods; it also explains how to use the *ActiveUSB* to perform the most common tasks.

[License agreement](#)

[System requirements](#)

[Installation](#)

[Registration](#)

[Distributing your application](#)

1.1 License Agreement

This legal document is an agreement between you, the Licensee, and A&B Software LLC. By installing *ActiveUSB SDK* on your computer, you are agreeing to be bound by the terms of this agreement. If you do not agree to the terms of this agreement, promptly return the unopened package, together with all the other material which comprises the product, respectively delete all *ActiveUSB SDK* related files.

1. Subject of agreement

The subject of this agreement is the software *ActiveUSB*, the operating manuals, and all other accompanying material. It will be referred to henceforth as *ActiveUSB SDK*.

2. Grant of license

A&B Software LLC grants the Licensee a non-exclusive, non-transferable, personal and worldwide license to use one copy of *ActiveUSB SDK* in the development of an end-user application, as described in section 3 (below). This license is for a single developer/one computer and not for an entire company. If additional programmers wish to use *ActiveUSB SDK*, additional copies must be licensed.

3. End user application

An *end user application* is a specific application program that is licensed to a person or firm for business or personal use. The files which are not listed under section 5 must not be included with the end user application. Furthermore, the end user must not be in a position to be able to neither modify the program, nor to create *ActiveUSB SDK* based programs. Likewise, the end user must not be given the *ActiveUSB SDK* serial number.

4. Royalties

The *ActiveUSB SDK* is NOT royalty free. The cost and licensing issue breaks down into the cost of the development environment and the cost for each run-time license that must be shipped with any product that embeds *ActiveUSB SDK*.

5. Redistributable files

The redistributable components of *ActiveUSB SDK* are those files specifically designated as being distributable in the [distributing your application](#) section of *ActiveUSB User's Guide*.

6. Trial version

A&B Software LLC grants the Licensee a non-exclusive license to test *ActiveUSB SDK* for 21 days on one computer system using the Trial version. The Trial version must be used solely for the trial and evaluation of the Software. The Licensee is not permitted to use the Trial version for any other purpose, including without limitation any use of the Software for productive purposes, hardware testing or in the operation of any Licensee's business.

At the end of the evaluation period the Licensee shall promptly remove all coding and other vestiges of the Trial version from the computer system, and make no further use of it, except to the extent that may be permitted under any subsequent agreements between the Licensee and A&B Software LLC

7. Third party software

ActiveUSB SDK is distributed with the runtime version of GenICam™ group reference implementation, subject to GenICam™ licensing agreement.

Certain third party software included with the Software is subject to additional terms and conditions imposed by third party licensor(s).

8. Copyright

The Software is the property of A&B Software LLC. A&B Software LLC reserves all rights to the publishing, duplication, processing and utilization of *ActiveUSB SDK*. A single copy may be made exclusively for security and archiving purposes. Without the express written permission

of *A&B Software LLC* it is forbidden to:

- reverse engineer, emulate, decompile, decrypt, disassemble, or in any way derive source code from *ActiveUSB SDK*
- modify, translate, adapt, alter or create derivative work(s) of *ActiveUSB SDK*
- copy *ActiveUSB SDK*'s accompanying written documentation
- lend, hire out or lease *ActiveUSB SDK*.

9. Exclusion of warranties

A&B Software LLC offers and the Licensee accepts the product 'as is'. A&B Software LLC does not warrant *ActiveUSB SDK* will meet the Licensee's requirements, nor will operate uninterrupted, nor error free.

10. Liability

With the exception of damage caused by willful or gross negligence, neither A&B Software LLC nor its distributors are responsible for any damage whatsoever which is put down to the use of *ActiveUSB SDK*. This is valid without exception, including loss of profits, lost working time, lost company information or other financial losses. In any event the liability of A&B Software LLC is limited to the purchase price.

11. Duration of Agreement

This agreement is valid for an indefinite period of time. The Licensee's rights as a user automatically expire if the conditions of this agreement are in any way violated. In this event all data storage material and all copies of *ActiveUSB SDK* are to be destroyed.

1.2 System Requirements

Hardware requirements:

- 2 GHz Pentium 4 or better CPU recommended.
- 1 GB or more RAM recommended.
- A graphic card supporting 65535 colors or higher.
- One or more USB3 boards or notebook card installed on the system.
- One or more USB3 Vision™ cameras connected to the system.

Note - ActiveUSB only supports cameras that comply with USB3 Vision™ standard. Cameras that have the USB3 physical interface, but are not USB3 Vision compliant will not be recognized by ActiveUSB.

Note - While it is possible to acquire images from certain USB3 Vision camera through USB2 ports, this will only work at very slow frame rates and small resolutions. It is highly recommended that you use a USB3 Interface Controller.

Software requirements:

- Windows XP, Vista, Windows 7, Windows 8, Windows 10
 - .NET FrameWork (for .NET compatibility)
-

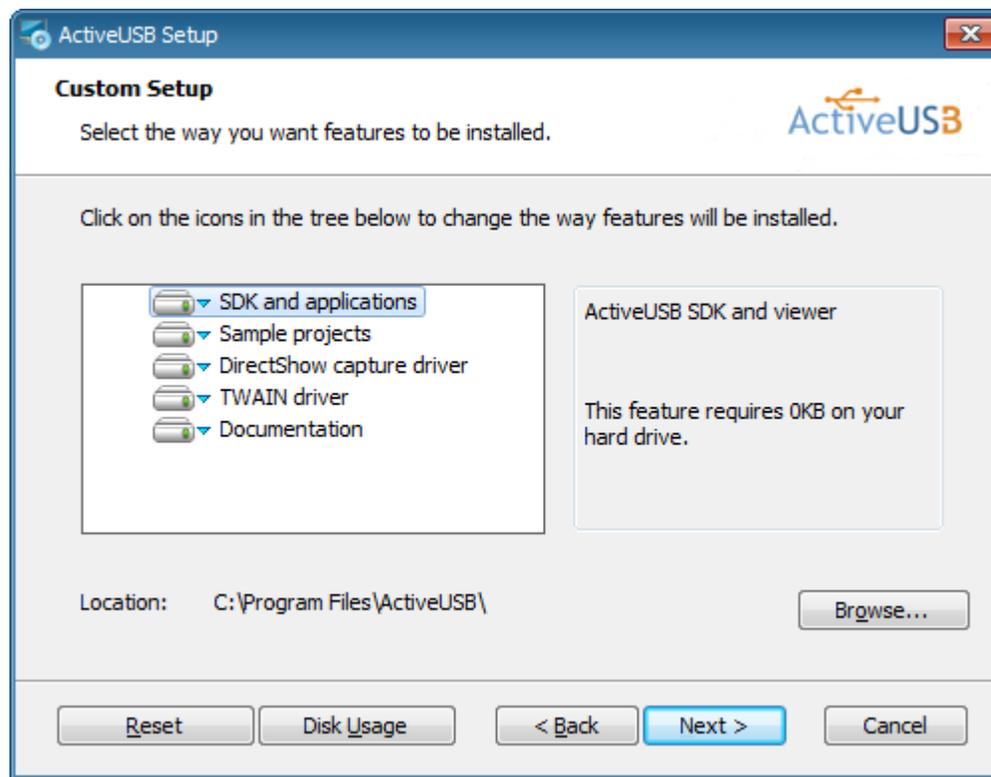
1.3 Installation

To install *ActiveUSB*, perform the following steps:

1. Save and exit out of all currently running applications.
2. Insert *ActiveUSB* CD into the CD-ROM drive. The setup program should start automatically. The **Windows Installer** box with a status bar will appear while setup prepares to start the installation process.
3. After the setup program has verified your system has the appropriate installer files, you are ready to start installing *ActiveUSB*. The **Welcome** dialog box will appear. After reading the preliminary information, click **Next**.



4. The **License Agreement** dialog box will appear. To accept the license and continue, click **I Agree**, and then click **Next**. Note that the **Next** button is not available until you click **I Agree**. If you do not accept the license, click **I Do Not Agree**, and setup will terminate.
 5. The **Choose Setup Type** dialog box will appear. Select which version of *ActiveUSB* you want to install: **Basic** or **DVR**.
 6. The **Select Installation Options** dialog box will appear, where you can modify features to be installed. The default location of *ActiveUSB* files is *C:\Program Files\ActiveUSB*. If you want to change the location, click **Browse...** and enter the path for the desired folder. Click **Next**.
-



7. The **Ready To Install** dialog box will appear. Click **Install**. *ActiveUSB* will begin installing on your system.
8. Once installation is finished, the **Complete Setup** dialog box will appear. Read the important information in the dialog and then click **Finish** to exit the installer. Note that depending on your operation system you might need to reboot your system at this point. You will be prompted if a reboot is required; if a message appears, follow the on-screen instructions.
9. Refer to the [Driver setup](#) chapter to continue with the installation.

-

1.4 Driver setup

In order for *ActiveUSB* to recognize USB3 Vision cameras that are connected to your system, *USB3 Vision Camera Driver* must be installed for each camera. If your USB3 Vision camera comes with its own system driver, do not use it, as it may not be compatible with *ActiveUSB*.

To install *USB3 Vision Camera Driver*, use the [USB3 Vision configurator](#) or select *Install Camera Driver* in the *ActiveUSB* start menu. You can also run the *DriverInstall* application located in the *Program Files\ActiveUSB\Drivers* folder.

In certain cases you may have to install the camera driver manually. In order to do this, perform the following steps:

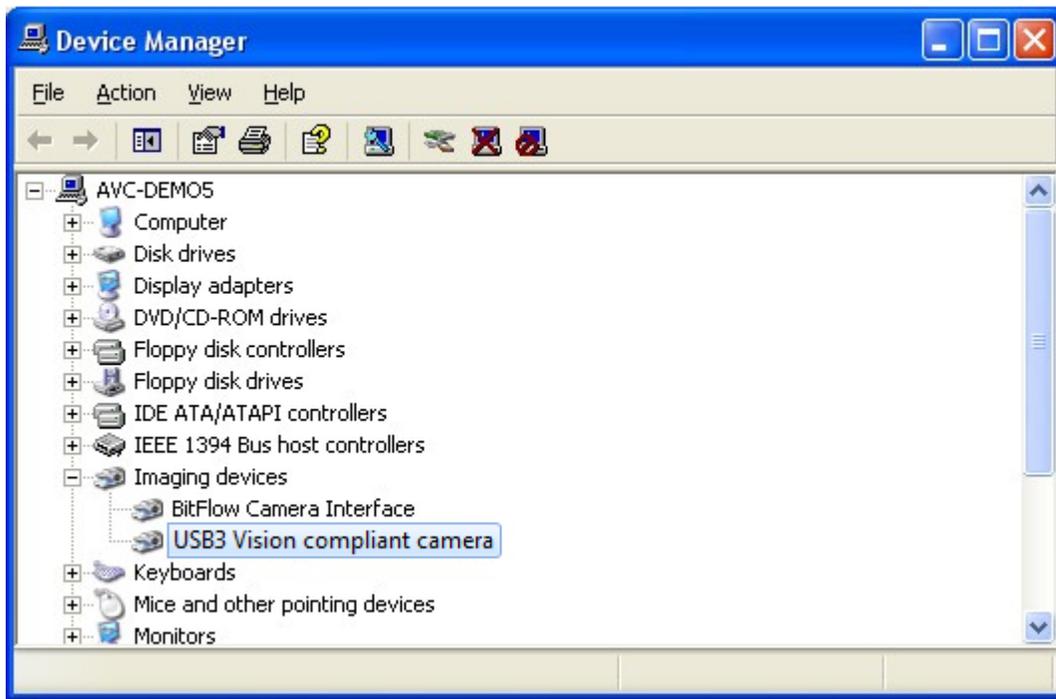
1. Make sure *ActiveUSB* is [installed](#) on your system. Connect your camera to a USB3 port. If you are using Windows XP and no driver was previously installed for the camera, the **Found New Hardware** dialog will appear as shown below.



If the dialog does not show up or if you are using Windows 7/8, proceed to section 4 .

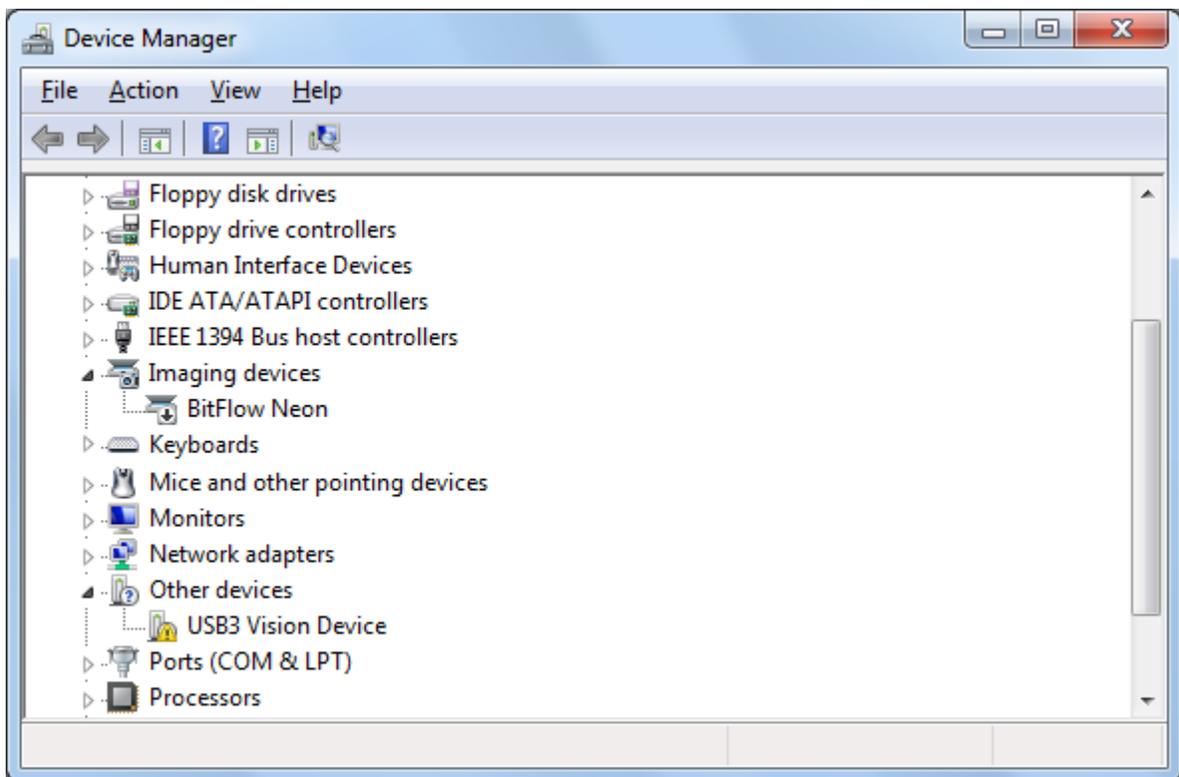
2. Choose not to connect to Windows Update and click **Next**. On the next dialog select **Install the software automatically**. Ignore the warning message that the driver is not signed up with Microsoft and click **Continue Anyway**.

3. Click **Finish** to complete the installation of the *USB3 Vision Camera Driver* for the selected camera. Verify that your camera is now listed in the Device Manager as *USB3 Vision compliant camera* :

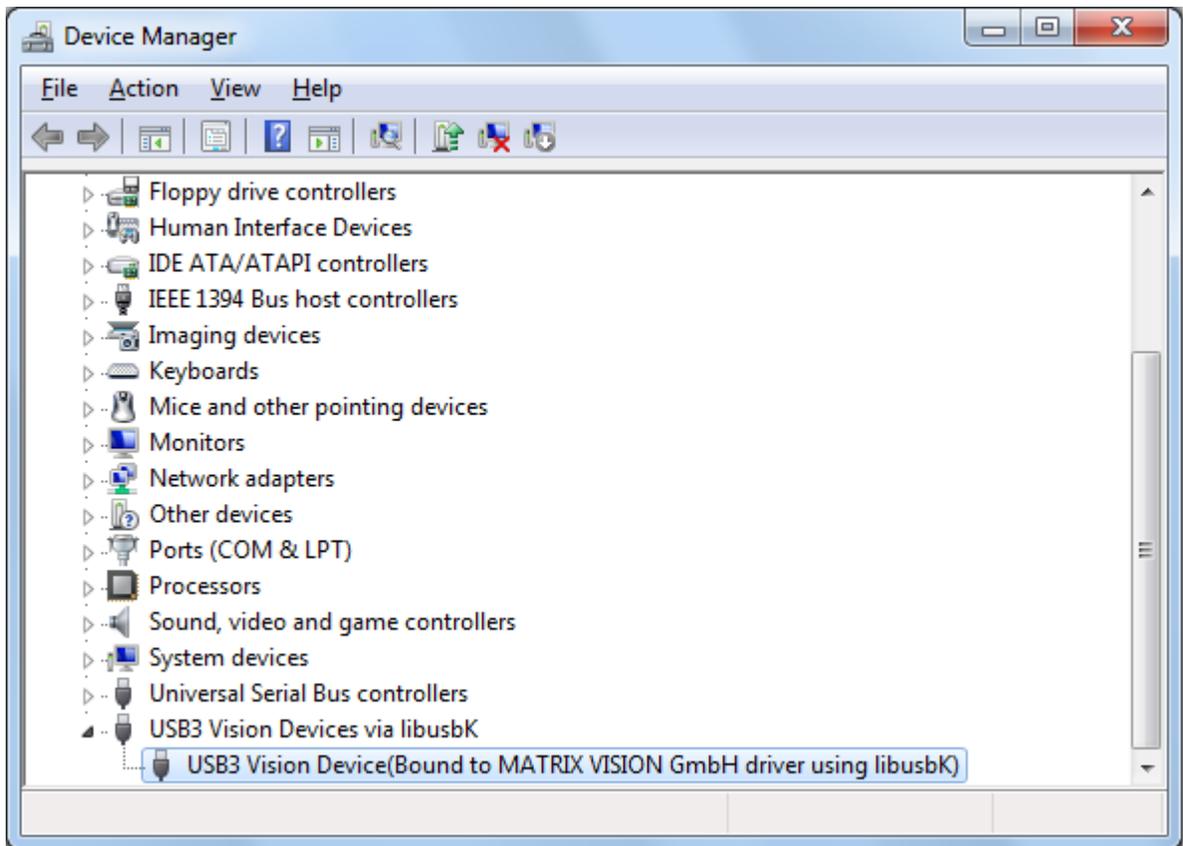


4. If the **New Hardware Found** dialog didn't appear after you connected your camera, go to the Start menu -> Programs -> ActiveUSB and select *Install Camera Driver*. This should install the driver for all USB3 Vision cameras connected to your system. Ignore the warning messages that the driver is not signed up with Microsoft.

5. As an alternative to the automatic installation, you can install the driver manually. Right click on **My Computer** and click **Properties**. Select the **Hardware** tab and click **Device Manager**. If there is no compatible driver installed, your camera will appear in the device list under Other Devices as a USB3 Vision camera with a question or exclamation mark next to it:



If the system already has a compatible third party driver installed, the camera will appear in the Device Manager with the third party driver's name next to it:



In order for the camera to work properly with *ActiveUSB* software, its driver must be changed to *USB3 Vision Camera Driver*.

6. Right click on your device and select **Properties** . Choose the **Driver** tab and then click **Update Driver** .

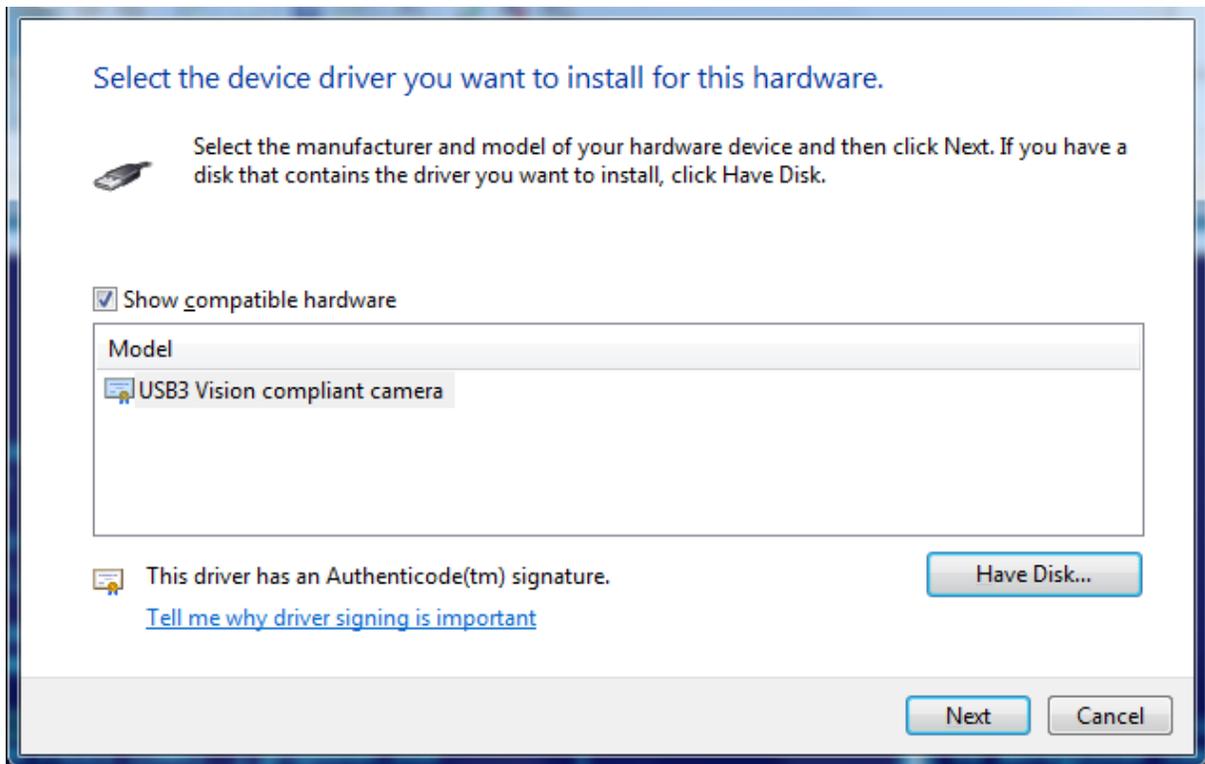
7. Select either **Install from a list or specific location** (Windows XP) or **Browse my computer for driver software** (Windows Vista/7/8/10) and click **Next** .

8. Select either **Don't search, I will choose the driver to install** (Windows XP) or **Let me pick from a list of device drivers on my computer** (Windows Vista/Windows 7) and click **Next** . On the next dialog, click **Have disk**.

9. Provide the path to *ActiveUSB* subfolder containing a camera driver for your operating system. A typical location would be:

Windows XP/Vista/Win 7/Win 8/Win 10	C:\Program Files\ActiveUSB\Driver\Win32
Windows XP/Vista/Win 7 64-bit/Win 8 64-bit/Win 10 64-bit Windows Server 64 bit	C:\Program Files\ActiveUSB\Driver\Win64

10. From the list of the drivers select *USB3 Visoin compliant camera* and click **Next** . Ignore the warning message that the driver is not signed up with Microsoft:



11. Click **Finish** to complete the installation of *USB3 Vision Camera Driver* for the selected camera. Verify that your camera is now listed in the device manager as *USB3 Vision compliant camera* under Imaging Devices.

The driver installation is now complete. You might need to restart the system for the changes to come into effect.

Note - the described procedure must be repeated for each USB3 Vision camera that you intend to use with ActiveUSB, regardless of a manufacturer or model.

Note - ActiveUSB supports only those cameras that comply with USB3 Vision™ Specifications. Cameras that have the USB3 physical interface, but are not USB3 Vision compliant will not be recognized by ActiveUSB.

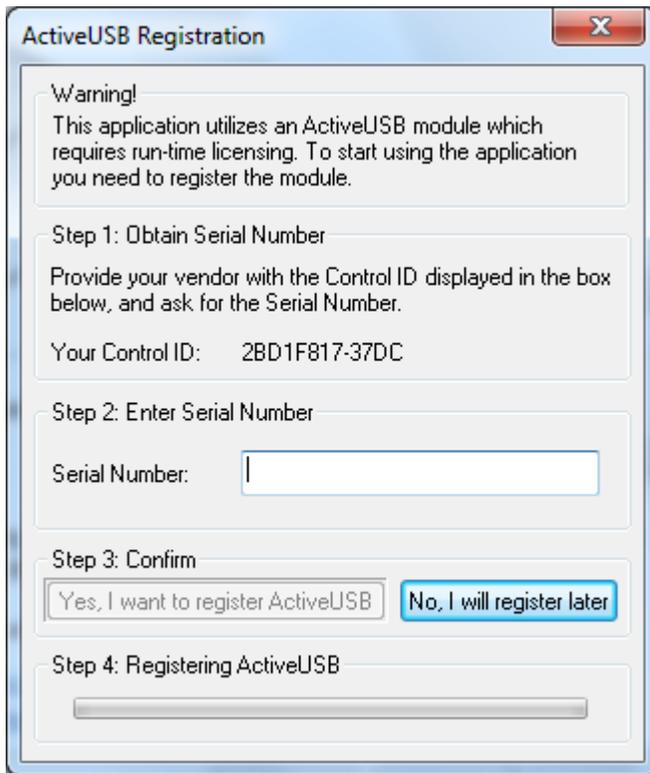
-

1.5 Registration

The *ActiveUSB* setup includes the demonstration license that allows you to use the control in both design and run-time mode for a period of 21 days.

If you wish to continue using the control, you must acquire a design-time license from your distributor. The design-time license is provided in form of a license file *ActiveUSB.lic* that must be saved in the ActiveUSB folder (typically, C:\Program Files\ActiveUSB) replacing the existing file. In addition, if you need to execute ActiveUSB based applications outside of your VB development environment, you should perform a run-time registration. This is done through the following procedure.

When you first start an executable file that was created using *ActiveUSB* control, the registration dialog box will appear.



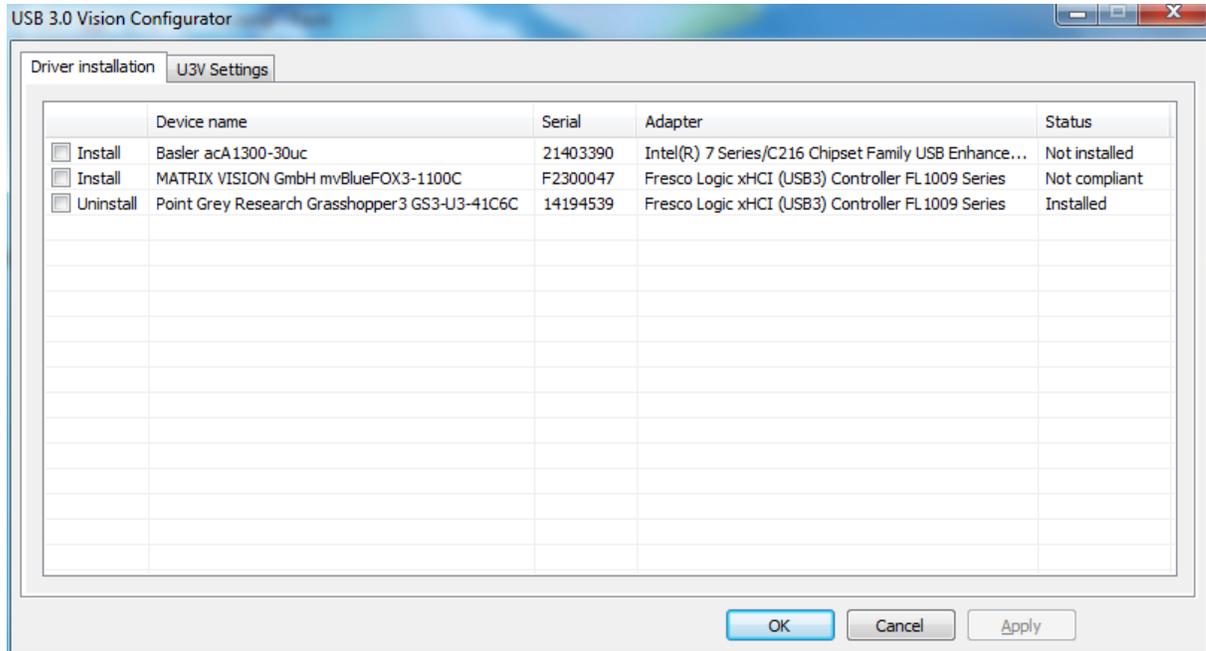
The dialog will display a Control ID uniquely generated for your system. Based on this ID, your distributor will provide you with a Serial Number that will validate a run-time permission for *ActiveUSB*. After the proper Serial Number is entered in the dialog and registration completed, all ActiveUSB based application will become unlocked.

1.6 USB3 Vision configurator

Provided with *ActiveUSB* is the USB3 Vision configurator utility which allows you to perform a driver installation for each connected camera and set up general parameters for the SDK operation.

To start the USB3 Vision configurator run "ucamconfig.exe" from *ActiveUSB* working folder. You can also execute it through the Windows Start Menu: Start -> All Programs -> ActiveUSB -> IP Configurator. The following user interface will be displayed:

Driver Installation



This panel will show all USB3 Vision cameras detected on your system, the names of USB adapters they are connected to and the status of drivers installed for each camera. The status of a driver can be one of the following:

Not installed - no driver is installed for the camera

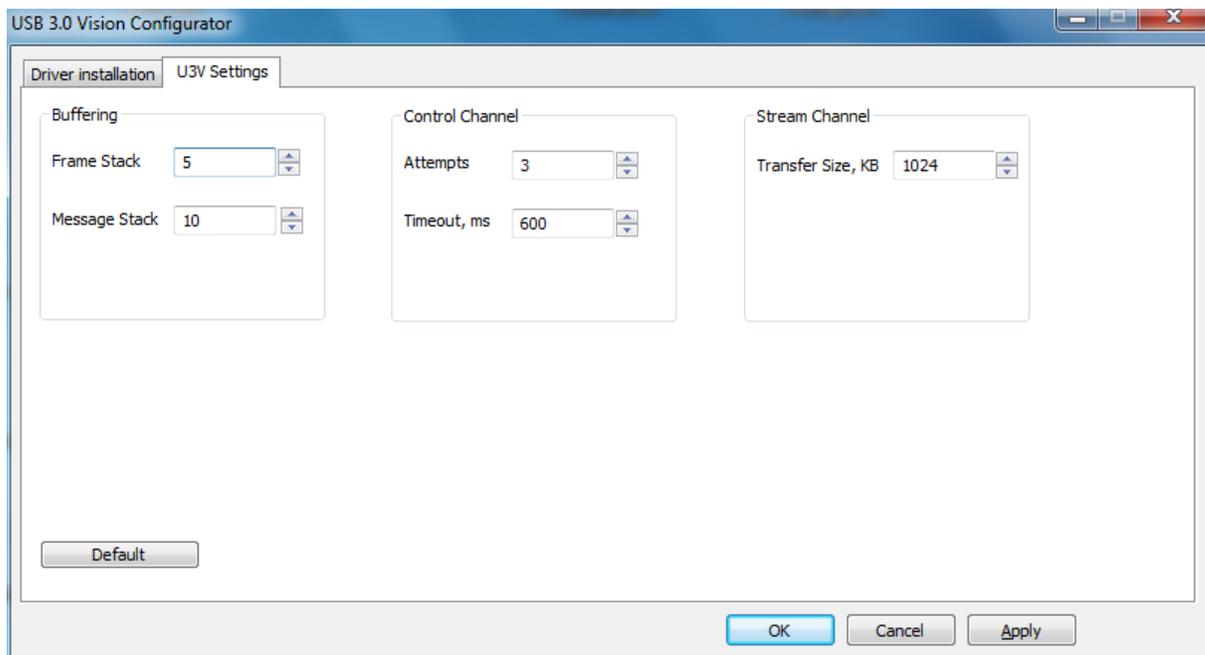
Not compliant - a third-party camera driver not compatible with *ActiveUSB* is installed

Installed - USB3 Vision compliant *ActiveUSB* camera driver is installed

If the driver is not installed or not compliant, check the *Install* box in the left column next to the camera name and click Apply. The correct *ActiveUSB* driver will be installed for the selected camera. If you want to uninstall already installed *ActiveUSB* camera driver, check the *Uninstall* box in the left column next to the camera name and click Apply.

Note - *ActiveUSB* may work with certain non-compliant drivers such as *libusbK* driver, but it may limit *ActiveUSB* functionality.

U3V Settings



The left pane displays the settings used for **Buffering** of image frames and messages. The following options are available:

Frame stack - the size of the image frame buffer. In order to process all incoming frames without missing some of them, an *ActiveUSB*-based application should be able to complete processing of a frame before the next frame arrives from the camera. In some cases an intermittent overload of the system may cause the application to pause and extend the processing of the current frame beyond the frame interval. The existence of the frame stack allows *ActiveUSB* to buffer incoming frames and delay their arrival to the application's processing queue without skipping them. The larger the frame stack is, the higher the allowance for intermittent overloads will be. A drawback of selecting a large size for the stack is a high memory consumption. The default size of this parameter is 5.

Message stack - the size of the message buffer. Used for processing of messages coming from the camera. The existence of the message stack allows *ActiveUSB* to buffer incoming messages and delay their arrival to the application's processing queue without skipping them. The default value of the message stack is 10.

The central pane displays the settings for the **Control channel** which is used to obtain and modify camera parameters. The following options are available:

Attempts - the number of times *ActiveUSB* will try to send the same command to the camera. If the camera fails to respond in time (timeout) on all of the attempts, *ActiveUSB* will raise an error. The default value for this parameter is 3.

Timeout, ms - the interval in milliseconds during which *ActiveUSB* waits for a response from the camera after sending a command. For example, when you move the Exposure slider in the [Property Pages](#), the application sends a series of commands to a corresponding camera register, and each command must be acknowledged by the camera before the application can send the next command, in order to maintain the synchronization between the application and camera. The default value for this parameter is 600 ms, but some cameras have a slower response and may require using a larger timeout.

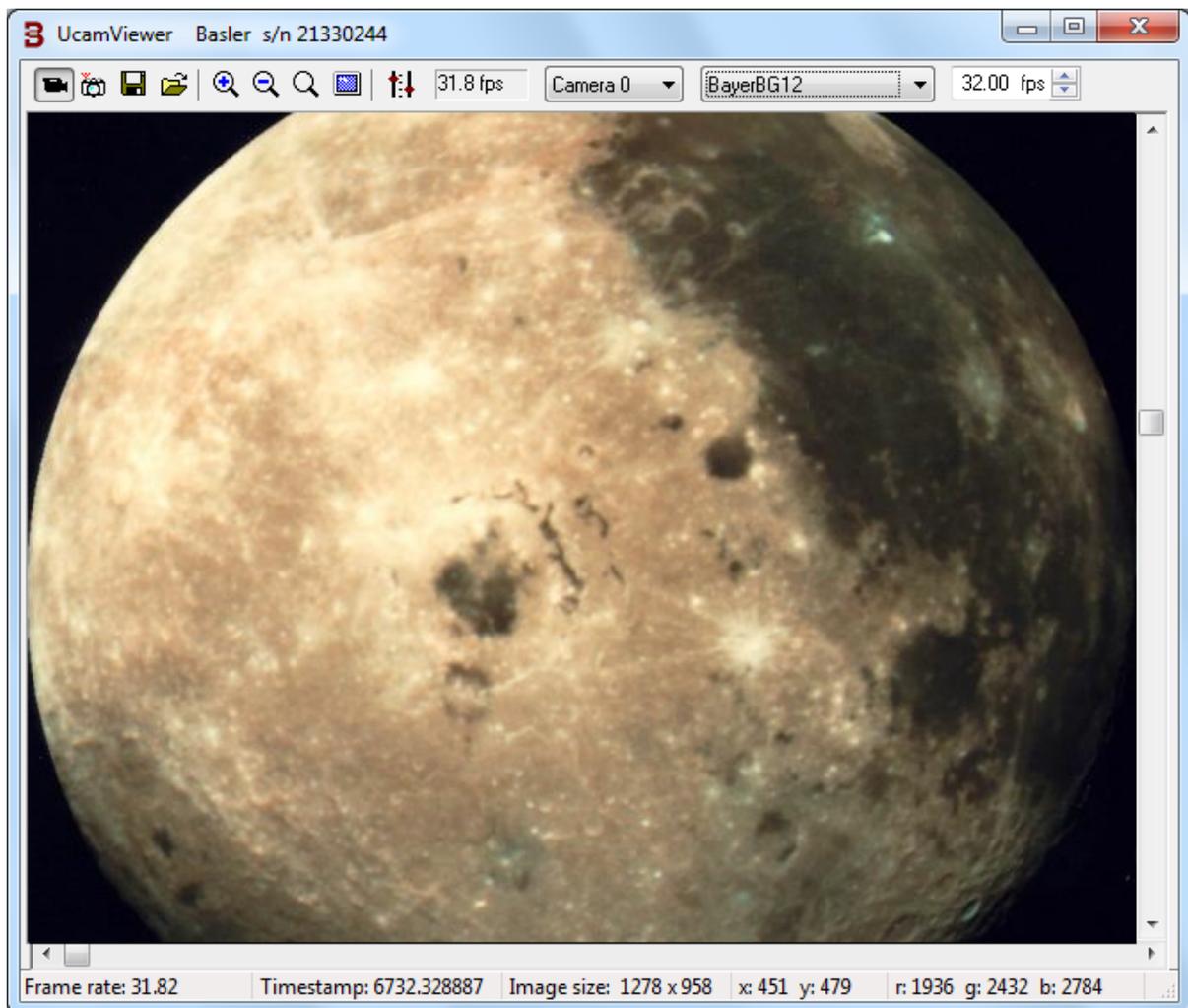
The right pane displays the settings for the **Stream channel** which is used to transmit video from cameras to *ActiveUSB*. The following options are available:

Transfer Size, KB - the size of DMA transfer blocks in kilobytes. When a camera streams images over a USB interface, it does it in a series of equally divided blocks. The larger the size of the block is, the lower the CPU overhead will be. If the transfer size is larger than the image size, it will be cut to the actual image size. The default value for this parameter is 1024, but depending on your camera and USB adapter it may need to be reduced to accommodate the maximum transfer size supported by the hardware.

1.7 UcamViewer

UcamViewer is a universal plug-and-play application for USB3 Vision™ compliant cameras which is intended for camera and network performance evaluation. It can display live video from multiple cameras, provide full control over camera parameters, analyze incoming video frames and save them to image files. UcamViewer is available only as part of *ActiveUSB* SDK and not as separate distribution.

To start UcamViewer run "UcamViewer.exe" from *ActiveUSB* working folder. You can also run it through the Windows Start Menu: Start -> All Programs -> ActiveUSB -> UcamViewer or by double-clicking on the UcamViewer icon at the desktop. The following user interface will be displayed:



When UcamViewer is launched for the first time, it will attempt to connect to the first camera in the USB3 Vision camera list, set the optimal packet size and start acquiring the live video.

Toolbar

The most often used commands to operate the viewer are provided through the toolbar at the upper part of the application window:

**Live Video**

Switches the camera into the continuous acquisition mode and initiates live preview.

**One Shot**

Initiates an acquisition of one frame.

**Save Image**

Lets you save the current frame buffer in an image file. When you click this button, the **Save As** dialog box will appear where you can select the file name and one of the image file formats: BMP, TIF and JPEG.

**Open Image**

Lets you open the static image from a file and display the image. When you click this button, the **Open** dialog box will appear where you can select the file name and one of the image file formats: BMP, TIF and JPEG.

**Zoom in**

Click this button to increase the magnification of the image.

**Zoom out**

Click this button to decrease the magnification of the image.

**Fit to screen**

Click this button to change the magnification factor of the image so that it fits to the image window.

**Full screen**

Click this button to enter the full screen mode. To exit the full screen mode, press the Esc key or double-click on the image.

**Settings**

Click this button to display *ActiveUSB* [Property Pages](#).

In addition to buttons, the toolbar also contains the following controls:

Display rate

Show the current display rate of the camera in frames per second. The display rate can be slower than the actual camera fps and may depend on the network throughput and image processing intensity.

Select Camera

Use this option to switch between several connected cameras.

Select Format

Use this option to select the desired pixel format from the list of available formats.

Select Frame Rate

Use this option to set the desired frame rate for the camera.

Status Bar

The Status Bar appears along the lower edge of UcamViewer window and contains the following information fields in the order from left to right:

FPS

Displays the actual frame rate of the camera.

Timestamp

Displays the timestamp associated with the current video frame.

Image Size

Displays the horizontal and vertical size of the video frames.

Cursor Position

Displays the X and Y coordinates of the mouse pointer. The coordinates are expressed in pixels relative to the top left corner of the image frame.

Pixel Value

Displays the value of the pixel identified by the mouse pointer. For a monochrome image, each pixel value will be represented by a single number, while for an RGB image by a triad of numbers.

Menu

Right-clicking the video window of UcamViewer will display the context menu with the following commands:

Acquire Video

Switches the camera into the continuous acquisition mode and initiates live preview. Equivalent to pressing the **Live** button.

Store camera parameters

Selecting this option will make UcamViewer store the main camera parameters upon its exit and restore them upon the next start. The parameters of each camera are stored independently in the system registry.

Settings

Displays *ActiveUSB* [Property Pages](#).

Affinity

Lets you select the number of logical CPUs that will be used for image processing. By default *ActiveUSB* splits the processing tasks among all logical CPUs (cores) available on the system. You may want to reduce the amount of utilized cores for benchmarking or to free CPU resources for other tasks.

File Transfer

Opens the File Transfer dialog allowing you to download a data file from the camera or upload it to the camera. The File Transfer dialog will list the names of all files in the camera that can be accessed via the file transfer. When a specific file name is selected, the "Download from camera"

or/and "Upload to camera" buttons will become available depending on the access mode of the file. Clicking one of these buttons will let initiate the transfer of data between a file in your system and file in the camera. The File Transfer option is typically used for updating the firmware of the camera and it is only available if the camera supports the file access functionality.

Save Image

Lets you save the current frame buffer in an image file.

Open Image

Lets you open the static image from a file.

Zoom in

Increases the magnification of the image.

Zoom out

Decreases the magnification of the image.

Fit to screen

Changes the magnification factor of the image so that it fits to the image window.

Operation

You can run several instances of UcamViewer at the same time provided each instance of the viewer is connected to a different camera.

If you want the current parameters of the camera to be memorized, activate the *Store camera parameters* option from the context menu. UcamViewer will save the parameters of the camera in the system registry upon exiting and restore it next time this camera is selected.

You are now ready to move to the next chapter and create your own USB3 Vision application!

1.8 Distributing your application

If you create an application using *ActiveUSB SDK*, your setup should include *ActiveUSB redistributable package* available from A&B Software upon request. There is no need to provide a user of your application with the license file *ActiveUSB.lic*. In addition, you must ensure that the following files exist on the end-user's system:

mfc42.dll, v.6.0	MFCDLL shared library
atl.dll	Active template library
msvcrt.dll, msvcrt40.dll	C run-time libraries
oleaut32.dll	OLE property frame
regsvr32.exe	control registration utility

These files are part of the Windows operating system and they are typically located in the Windows system directory. Please refer to Microsoft's redistribution policy if you need to redistribute them. In addition [VC++ 8.0 Redistributable Package](#) must be installed on the end-user's system.

When an *ActiveUSB* based application is executed for the first time on an end-user's system, it will display the registration dialog (see [Installation](#)). You should instruct the user to provide you or your distributor with a Control ID displayed in the dialog. After the run-time license of the user is validated, the distributor will issue a unique serial number which will unlock the control on the user's machine.

2 Getting started

This chapter describes how to create a simple application with *ActiveUSB* control using various development platforms.

[Visual Basic](#)
[Visual C++](#)
[VB.NET](#)
[Visual C#](#)

2.1 Visual Basic

This chapter shows you how to get started with *ActiveUSB* control in VB 6.0. With just a few mouse clicks and one line of code, you will be able to display a live video image in your Visual Basic program and report a value of a selected pixel in real time.

Creating the Project

Assuming that you have already run the *ActiveUSB* installation program and started Visual Basic, the next step is to create a project. Begin by selecting the *New Project* command from the file menu, and select *Standard EXE* as your project type. Then use the *Project / Components...* command and select *ActiveUSB Control* from the list. You will see *ActiveUSB* icon appear at the bottom of the toolbox:

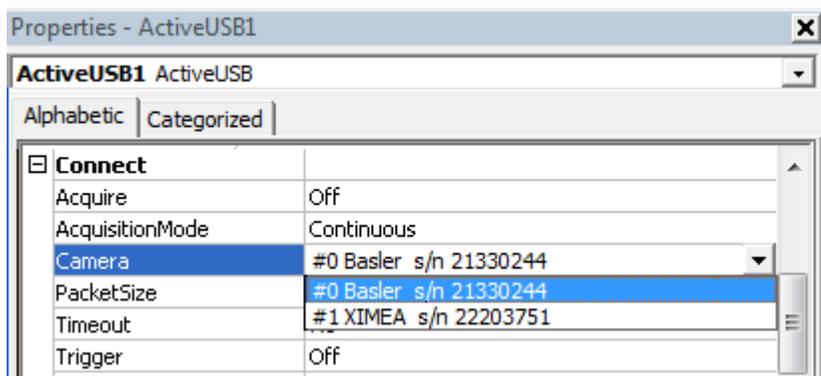


Creating the Control

Click the *ActiveUSB* icon in the *Toolbox* and draw a rectangular area on the form. A rectangle with the text "ActiveUSB Control" will appear on the form, and the *Project* window on the right will display *ActiveUSB*'s properties.

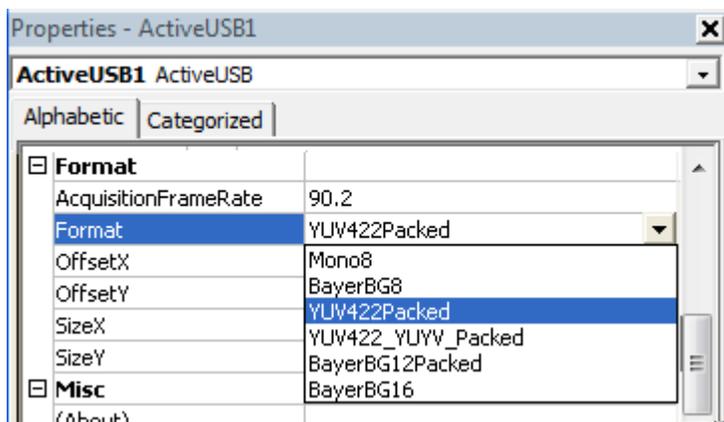
Selecting the Camera

In the properties window, click the *Camera* property. The list box will display all USB3 Vision™ cameras connected to your system. Select the one you intend to use:



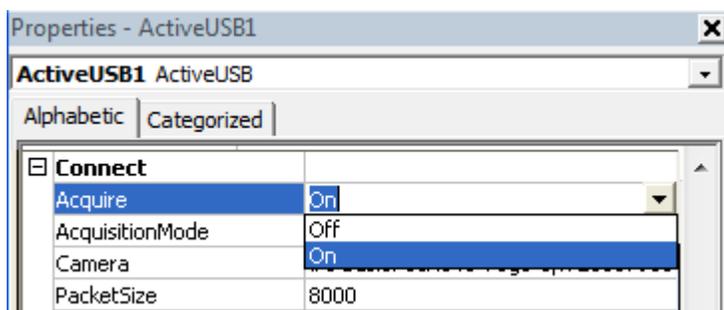
Selecting the Pixel Format

In the properties window, click the *Format* property. The list box will display all pixel formats provided by the selected camera. Choose the one you intend to use:



Activating continuous acquisition

In the properties window, click the *Acquire* property and set it to "On". The live video should appear on the form in the *ActiveUSB* control's window.



Adding a label

Click the label icon on the toolbox and draw a small rectangular area on the form outside of the *ActiveUSB* window. A label *Label 1* will appear on the form.

Adding the FrameAcquired event

Double-click in the control window. The code window will appear with the empty *FrameAcquired* subroutine in it. It is now time to enter the single line of code mentioned earlier:

```
Private Sub ActiveUSB1_FrameAcquired()  
    'the following line needs to be added to the code  
    Label1.Caption = ActiveUSB1.GetPixel (64, 32)  
End Sub
```

Running the application

You are now ready to hit F5 and watch your program display a live video and report a real-time pixel value in the coordinates $x = 64$, $y = 32$.

2.2 Visual C++

This chapter shows you how to get started with *ActiveUSB* control in Visual C++. With just a few mouse clicks and a few lines of code, you will be able to display a live video image in your C++ program and report a value of a pixel pointed by a mouse cursor in real time.

Creating the Project

VC++ 6.0

In the development environment select the *New* command from the file menu, and then select *Projects/MFC AppWizard.exe*. In the *Project name* field on the right type the name of your application, for instance *MyActiveUSB* and click *OK*. When *MFC AppWizard* appears, select *Dialog based* radio button and click *Finish*. The project will be created, and the program dialog *MyActiveUSB* will be displayed for editing.

VC++ 2005, 2008, 2010, 2012

In the .NET development environment select *New -> Project*. The *New Project* Dialog box will appear. Select *Visual C++ projects* on the left and *MFC Application* on the right. In the *Name* field below type the name of your application, for instance *MyActiveUSB* and click *OK*. When *MFC Application Wizard* appears, click *Application Type*, select *Dialog based* radio button and click *Finish*. The project will be created, and the program dialog *MyActiveUSB* will be displayed for editing.

Creating the Control

Right click in the dialog and select *Insert ActiveX control* from a shortcut menu. From the list of controls select *ActiveUSB class* and press *OK*. A white rectangle will appear on the dialog.

Generating the class for the Control

VC++ 6.0

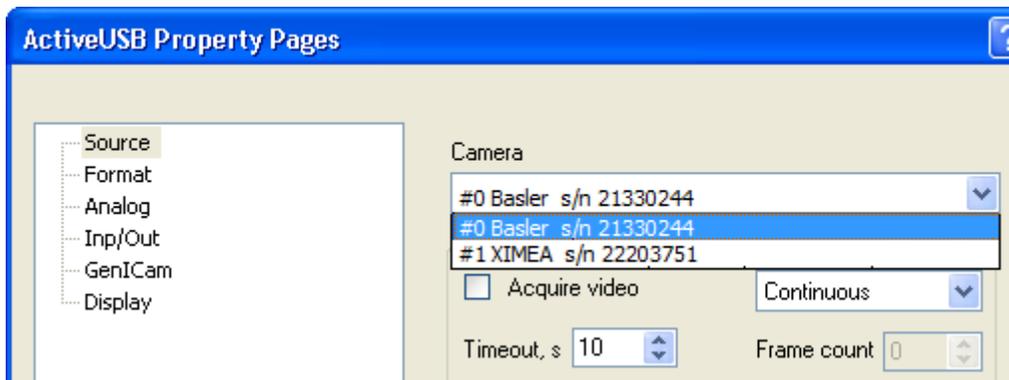
Right click in the dialog and select *Class Wizard* from the shortcut menu. When *MFC Class Wizard* appears, click the *Member Variables* tab. In the *Control ID* window click *IDC_ActiveUSB1* and then click *Add Variable*. Confirm generating the new *CActiveUSB* class. When the *Add Member Variable* dialog appears, enter the name for the *ActiveUSB* object, for instance *m_ActiveUSB*. Click *OK* to close the Wizard.

VC++ 2005, 2008, 2010, 2012

Right click on the *ActiveUSB* control in the program dialog and select *Add Variable* from the shortcut menu. *Add Member Variable Wizard* will appear. In the *Variable name* field enter the name for the *ActiveUSB* object, for instance *m_ActiveUSB*, and click *finish*. The Wizard will generate a wrapper class for *ActiveUSB* control and add a corresponding member variable to the main dialog class.

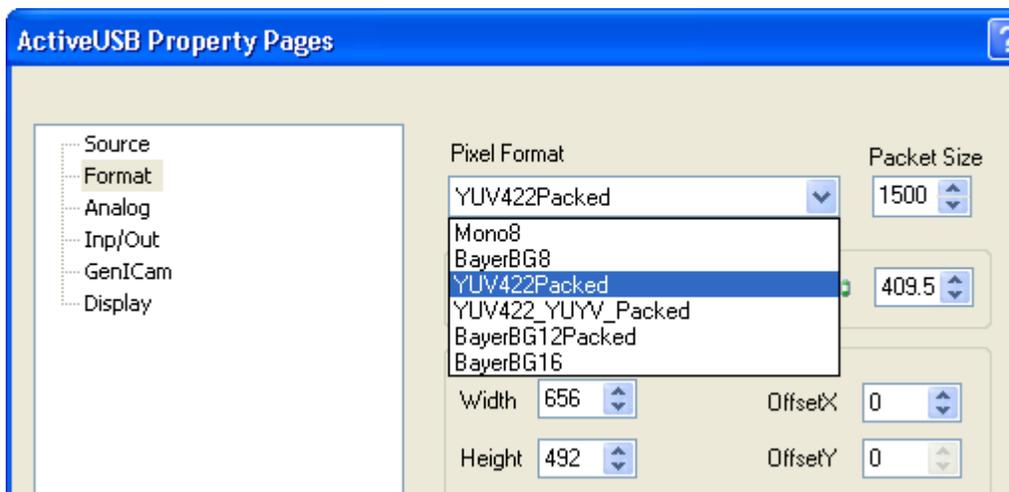
Selecting the Camera

Right click on the *ActiveUSB* control in the program dialog and select *Properties* from the shortcut menu. (In .NET select *Property pages* from the *View* menu). This will display the *ActiveUSB Class Properties* tab dialog. Click the *Source* tab. The *Camera* list box will contain the names of USB3 Vision™ cameras connected to your system. Select the one you intend to use.



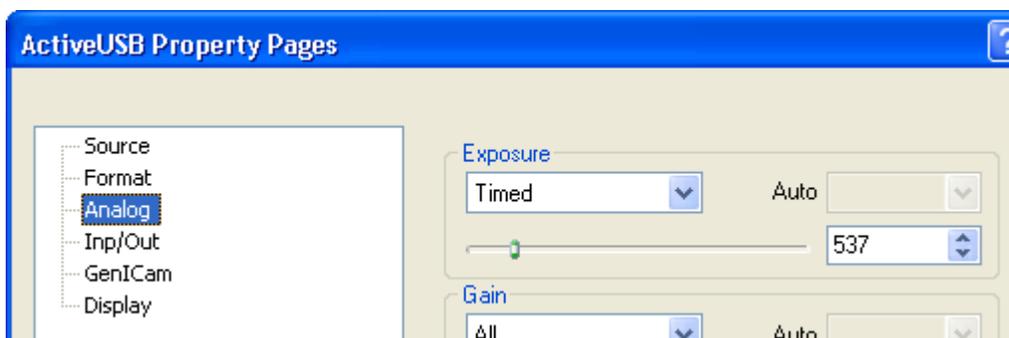
Selecting the Video Format

Switch to the *Format* tab. Click the arrow next to the *Format* box. The list box will display all the pixel formats available for the chosen camera. Select the one you intend to use. Then select the desired image size in the *Width* and *Height* boxes.



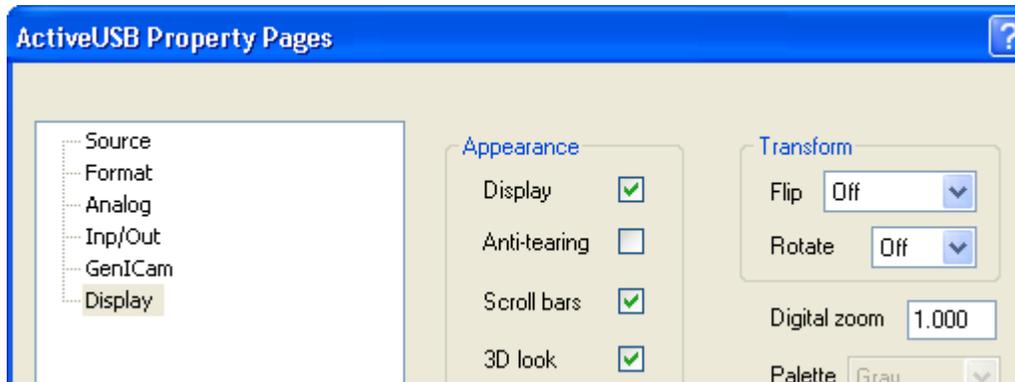
Adjusting the Analog settings

Switch to the *Analog* tab. Using the slider controls and list boxes adjust the exposure, gain, black level and white balance settings. Note that the availability of the controls depends on the selected camera.



Modifying the Control's appearance

Switch to the *Source* tab. Select *the Scroll bars* check box:



Adding the Start button

In the *Controls* toolbox click on the *Button* icon and then draw a rectangular area on the program dialog. A button "Button1" will appear. Right click on the button and select *Properties* from the shortcut menu. In the *Caption* field type "Start" and double click on the button. The *Add Member Function* dialog box will appear. After clicking *OK* the source code window will be displayed with the new member function *OnButton1* added. Insert one line to the function body:

VC++ 6.0

```
void CMyActiveUSBDlg::OnButton1()
{
    // TODO: Add your control notification handler code here
    m_ActiveUSB.SetAcquire(TRUE);
}
```

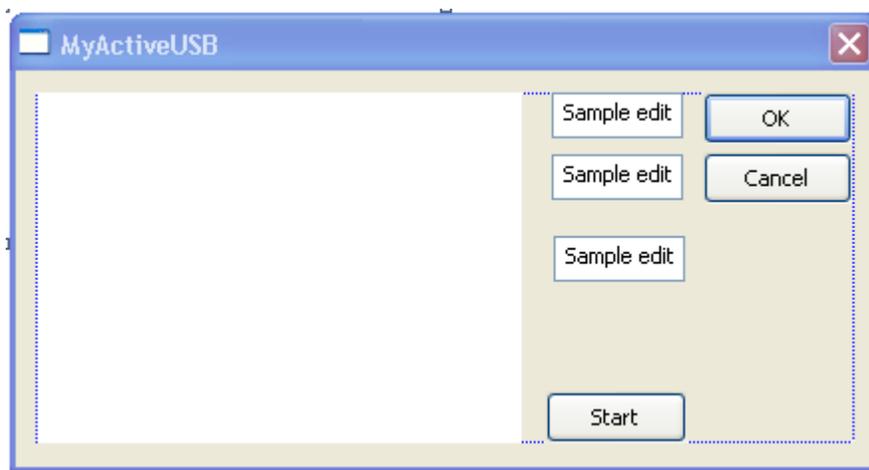
VC++ 2005, 2008, 2010

```
void CMyActiveUSBDlg::OnBnClickedButton1()
{
    // TODO: Add your control notification handler code here
    m_ActiveUSB.put_Acquire(TRUE);
}
```

This will activate continuous acquisition when the button is clicked.

Adding text boxes

In the *Controls* toolbox click on the *Edit Box* icon and then draw a small rectangular area on the program dialog. Repeat this two more times. Your final design of the program dialog will be similar to this:



Adding the MouseMove event

Right click on the ActiveUSB control in the program dialog and select *Events...* from the shortcut menu. Highlight the *MouseMove* event and click *Add and Edit*. The new event handler *OnMouseMoveActiveUSB1* will be added to the source code. Add the following lines to the body of the function:

```
void CMyActiveUSBDlg::OnMouseMoveActiveUSB1(short x, short y)
{
    // TODO: Add your control notification handler code here
    SetDlgItemInt(IDC_EDIT1,x);
    SetDlgItemInt(IDC_EDIT2,y);
    SetDlgItemInt(IDC_EDIT3,m_ActiveUSB.GetPixel(x,y));
}
```

Running the application

Close the main application form. From the *Build* menu of the development environment select *Execute MyActiveUSB.exe*. This will build and run the application. The application dialog will appear on the screen with a black image window and empty text boxes. Click the *Start* button to activate the continuous video acquisition and move the mouse cursor over the image window. You are now able to watch and scroll the live image and analyze pixel coordinates and values - all in real time!

Note - make sure to close the form containing ActiveUSB control before running your application, or otherwise the IDE will maintain the exclusive control over the camera and will not allow your application to display the video.

2.3 VB.NET

This chapter shows you how to get started with *ActiveUSB* control in VB.NET. With just a few mouse clicks and a few lines of code, you will be able to display a live video image in your VB.NET program, access the array of pixel values and display them in a table in real time.

Creating the Project

In the .NET development environment select *New -> Project*. The *New Project* Dialog box will appear. Select *Visual Basic projects* on the left and *Windows Application* on the right. In the *Name* field below type the name of your application, for instance *MyActiveUSB* and click *OK*. The project will be created, and the main application form will be displayed for editing.

Creating the Control

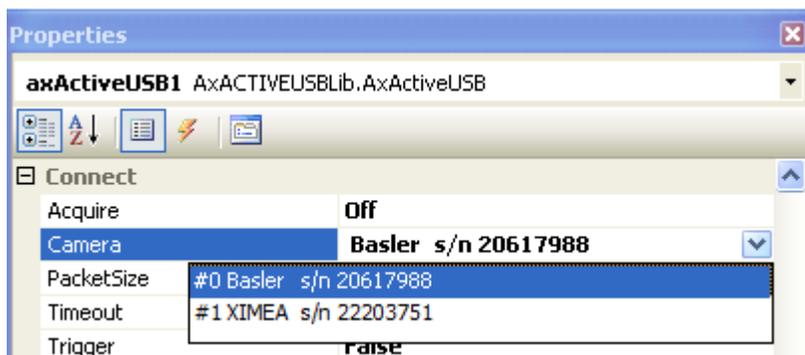
In the *Toolbox* select *Components*. From the *Tools* menu select *Choose Items... -> COM Components* and then select *ActiveUSB Class* from the list. You will see *ActiveUSB* icon appear at the bottom of the toolbox.



Click the *ActiveUSB* icon in the *Toolbox* and draw a rectangular area on the form. A rectangle with the text "ActiveUSB Control" will appear on the form, and the *Properties* window on the right will display *ActiveUSB*'s properties.

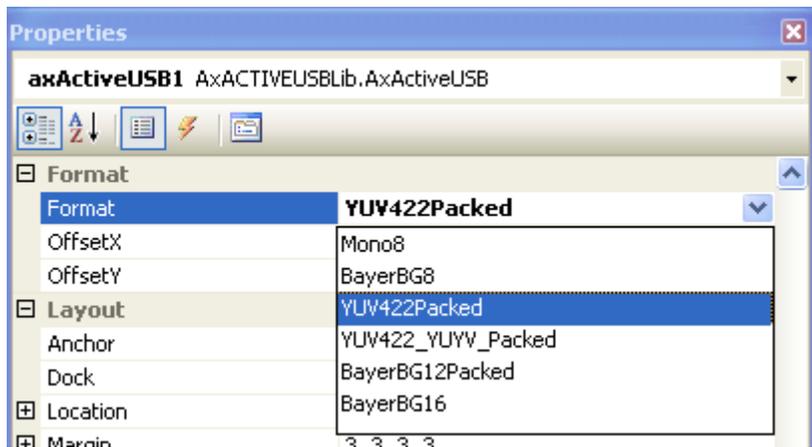
Selecting the Camera

In the properties window, click the *Camera* property. The list box will display all USB3 Vision™ compatible cameras connected to your system. Select the one you intend to use:



Selecting the Pixel Format

In the *Properties* window, click the *Format* property. The list box will display all pixel formats available for the chosen camera. Select the one you intend to use:



Adding the Start and Stop buttons

In the *Toolbox* select *Windows Form*, click on the *Button* icon and then draw a rectangular area on the program dialog. A button "Button1" will appear. Go to the *Property* window and change the *Text* property to "Start". Double click on the button. A new function *Button1_Click* will be added to the *Form1.vb* source code. Insert one line to the function body:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
Button1.Click
    AxActiveUSB1.Acquire = True
End Sub
```

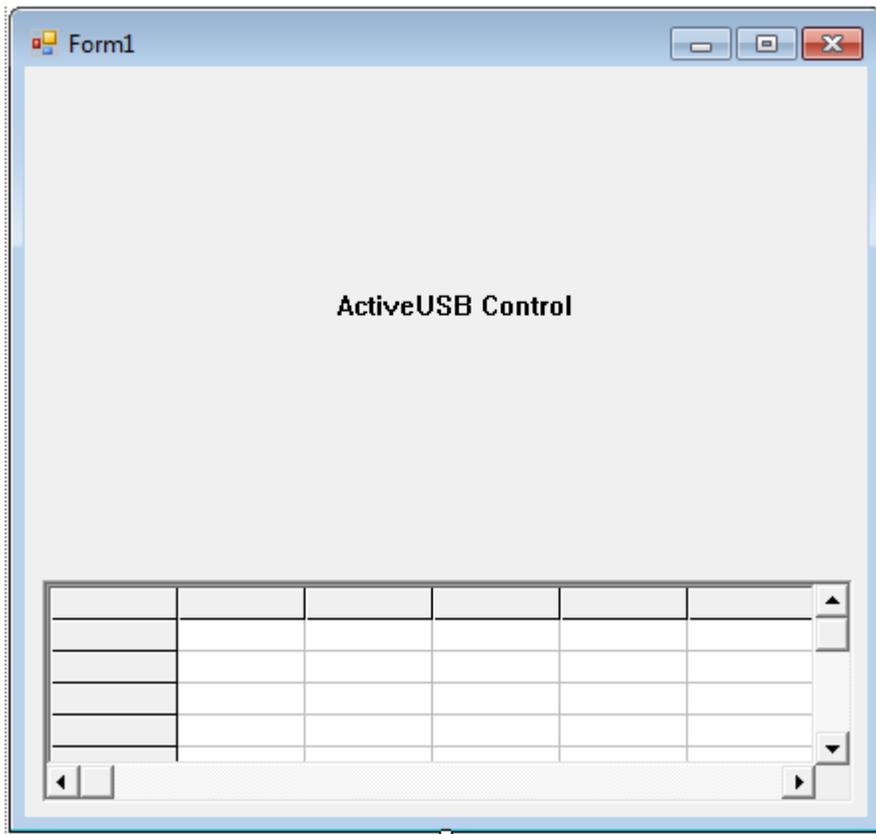
Repeat the same procedure for the *Stop* button adding the following line to the *Button2_Click* function:

```
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
Button1.Click
    AxActiveUSB1.Acquire = False
End Sub
```

Adding the Grid

From the *Tools* menu select *Customize Toolbox* and then select *Microsoft Flex Grid Control* from the list. A *FlexGrid* icon will appear at the bottom of the toolbox. Click the icon and draw a rectangular area on the form. Go to the *Property* window and set both *Cols* and *Rows* property to 5. You will see a corresponding number of rows and columns added to the grid on the main form.

Your final design of the main form will be similar to this:



Adding the FrameAcquired event

Double-click in the ActiveUSB control window. The code window will appear with an empty `AxActiveUSB1_FrameAcquired` subroutine in it. It is now time to enter the code that will retrieve an image data array and display pixel values in the grid cells:

```
Private Sub AxActiveUSB1_FrameAcquired(ByVal sender As System.Object, ByVal e As
AxActiveUSBLib._IActiveUSBEvents_FrameAcquiredEvent) Handles
AxActiveUSB1.FrameAcquired
    Dim A As Array
    Dim X As Integer
    Dim Y As Integer
    A = AxActiveUSB1.GetImageData
    For y = 1 To 4
        For x = 1 To 4
            AxMSFlexGrid1.set_TextMatrix(Y, X, A(X, Y))
        Next
    Next
End Sub
```

Running the application

Close the design window. From the *Debug* menu of the development environment select *Start* or press F5. This will build and run the application. The application dialog will appear on the screen with a black image window and empty text boxes. Click the *Start* button to activate the continuous video acquisition. The live image will appear in the control window and the real-time pixel values will be

displayed in the table.

Note - make sure to close the form containing ActiveUSB control before running your application, or otherwise the IDE will maintain the exclusive control over the camera and will not allow your application to display the video.

2.4 Visual C#

This chapter shows you how to get started with *ActiveUSB* control in Visual C#. With just a few mouse clicks and a few lines of code, you will be able to display a live video image in your C# program and report a value of a pixel pointed by a mouse cursor in real time.

Creating the Project

In the .NET development environment select *New -> Project*. The *New Project* Dialog box will appear. Select *Visual C# projects* on the left and *Windows Application* on the right. In the *Name* field below type the name of your application, for instance *MyActiveUSB* and click *OK*. The project will be created, and the main application form will be displayed for editing.

Creating the Control

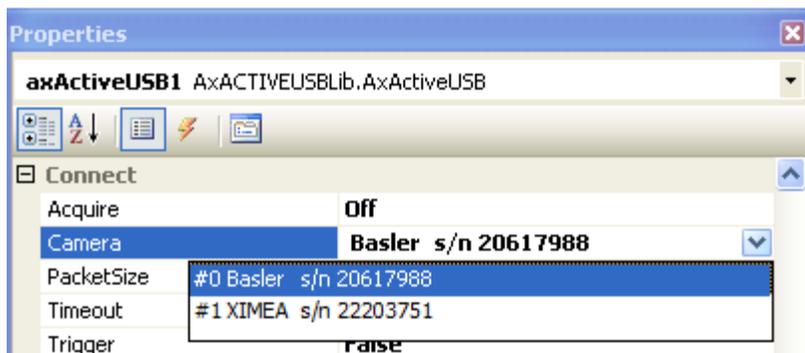
In the *Toolbox* select *Components*. From the *Tools* menu select *Choose Items...-> COM Components* and then select *ActiveUSB Class* from the list. You will see *ActiveUSB* icon appear at the bottom of the toolbox.



Click the *ActiveUSB* icon in the *Toolbox* and draw a rectangular area on the form. A rectangle with the text "ActiveUSB Control" will appear on the form, and the *Properties* window on the right will display *ActiveUSB*'s properties.

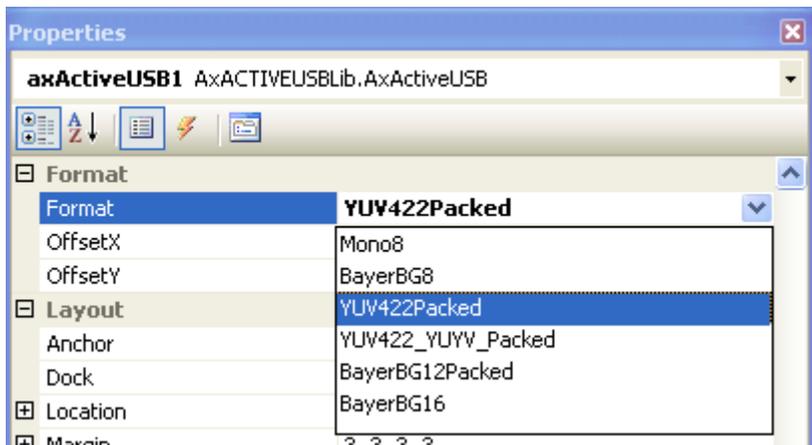
Selecting the Camera

In the properties window, click the *Camera* property. The list box will display all USB3 Vision™ compatible cameras connected to your system. Select the one you intend to use:



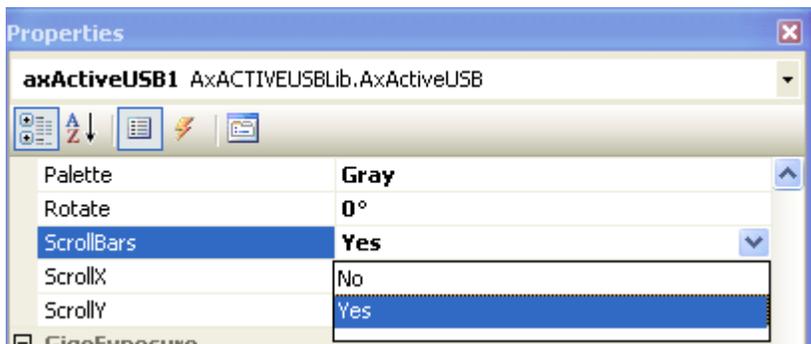
Selecting the Pixel Format

In the *Properties* window, click the *Format* property. The list box will display all pixel formats available for the chosen camera. Select the one you intend to use:



Modifying the Control's appearance

In the Properties window set the *Edge* and *ScrollBars* fields to "Yes"



Adding the Start button

In the *Toolbox* select *Windows Form*, click on the *Button* icon and then draw a rectangular area on the program dialog. A button "Button1" will appear. Go to the *Property* window and change the *Text* property to "Start". Double click on the button. A new member function *button1_Click* will be added to the *Form1* source window. Insert one line to the function body:

```
private void button1_Click(object sender, System.EventArgs e)
{
    axActiveUSB1.Acquire=true;
}
```

This will activate continuous acquisition when the button is clicked.

This will activate continuous acquisition when the button is clicked.

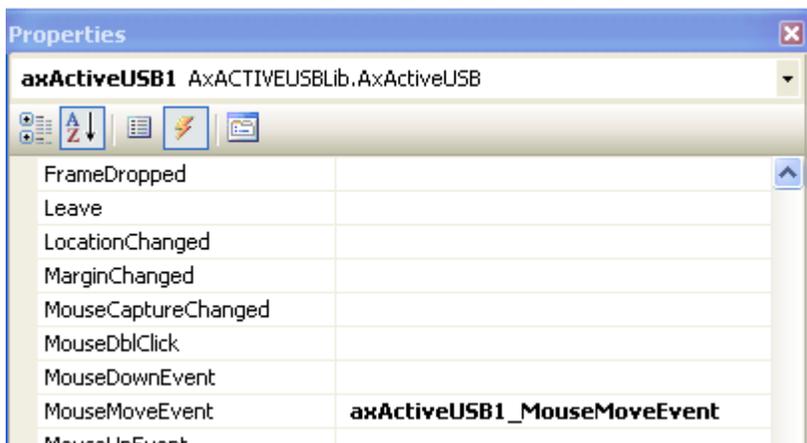
Adding text boxes

In the *Toolbox* click on the *TextBox* icon and then draw a small rectangular area on the program dialog. Repeat this two more times. Your final design of the main form will be similar to this:



Adding the MouseMove event

Click on the ActiveUSB control on the main form. In the *Property* window click the *Event* button. In the list of events double-click the *MouseMoveEvent*.



The new event handler `axActiveUSB1_MouseMoveEvent` will be added to the source code. Add the following lines to the body of the function:

```
private void axActiveUSB1_MouseMoveEvent(object sender,
AxActiveUSBLib._IActiveUSBEvents_MouseMoveEvent e)
{
    textBox1.Text=ToString(e.x);
    textBox2.Text=e.y;
    textBox3.Text=axActiveUSB1.GetPixel(e.x,e.y);
}
```

Running the application

Close the design window. From the *Debug* menu of the development environment select *Start* or press F5. This will build and run the application. The application dialog will appear on the screen with a

black image window and empty text boxes. Click the *Start* button to activate the continuous video acquisition and move the mouse cursor over the image window. You are now able to watch and scroll the live image and analyze pixel coordinates and values - all in real time!

Note - make sure to close the design window with the form containing ActiveUSB control before running your application, or otherwise the IDE will maintain the exclusive control over the camera and will not allow your application to display the video.

2.5 Using ActiveUSB API at runtime

Most applications would use *ActiveUSB* by embedding one or several ActiveX objects into an application form at design time, as described in the previous topics. Sometimes however you may want to use *ActiveUSB* SDK dynamically at runtime. This can be useful if you want a user of your application to decide if he wants to use USB3 Vision cameras or if you are creating a dynamic link library that has no GUI. ActiveUSB easily allows you to do it via its COM programming interface.

C/C++

1. Copy ActiveUSB_i.c and ActiveUSB.h files into your project folder and include them into your project.
2. Add the following defines to a module which will be using *ActiveUSB* functions:

```
#include <comdef.h>
#include <atlbase.h>
#include <atlconv.h>
#include "ActiveUSB.h"
```

3. Initialize COM library and instantiate an ActiveUSB object:

```
IActiveUSB *pActiveUSB;
HRESULT hr = CoInitialize(0);
if (hr==S_OK)
{
    hr = CoCreateInstance(CLSID_ActiveUSB, NULL, CLSCTX_INPROC_SERVER, IID_IActiveUSB,
    (void**) &pActiveUSB);
}
```

4. Start using *ActiveUSB* properties and methods, for example:

```
hr = pActiveUSB->put_Camera(0); // selecting camera #0
hr = pActiveUSB->Grab(); // grabbing a frame
hr = pActiveUSB->SaveImage( OLESTR("frame1.jpg") ); //saving a frame as jpg file
```

Refer to *UcamConsole* and *UcamWin* sample applications for more details.

VB6

1. Add the *ActiveUSB* component to the Toolbox as described in [Getting started in VB](#).
2. Declare a global *ActiveUSB*-type object variable in the beginning of your code:

```
Dim WithEvents AU As ActiveUSB
```

3. Instantiate an *ActiveUSB* object and assign it to the variable:

```
Set AU = New ActiveUSB
```

4. Start using *ActiveUSB* properties and methods, for example:

```
CameraList=AU.GetCameraList    'retrieving the list of names of connected cameras
AU.Camera=0                    'selecting camera #0
AU.SetFeature "Gain", 10       'setting Gain value
AU.Acquire=True                 'initiating the acquisition
```

5. To add an ActiveUSB event handler to your code, select the AU variable from the Object combo box on top of your code window and then select a desired event from the Procedure box on the right, for example FrameAcquired. The following fragment will be added to your code:

```
Private Sub AU_FrameAcquired()  
End Sub
```

Refer to *UcamByRef* sample application for more details.

6. When you are done with using the ActiveUSB object, destroy it with the following command:

```
Set AU = Nothing
```

VB.NET

1. 1. Right click on your application in the Solution Explorer, select Add Reference...->COM, then select *ActiveUSB* Control from the list.

2. Declare a global *ActiveUSB*-type object variable in the beginning of your code:

```
Dim WithEvents AU As ActiveUSBLib.ActiveUSB
```

3. Instantiate an *ActiveUSB* object and assign it to the variable:

```
AU = New ActiveUSBLib.ActiveUSB
```

4. Start using *ActiveUSB* properties and methods, for example:

```
Dim CamLst As Object  
Dim i As Integer  
CamLst = AU.GetCameraList           'retrieving camera list  
CamNumber = UBound(CamLst)         'number of connected cameras found  
For i = 0 To CamNumber              'filling out combo box with the names of cameras  
    ComboBox1.Items.Add(CamLst(i))  
Next  
.....  
AU.Camera = 0  
AU.Acquire = True
```

5. To add an ActiveUSB event handler to your code, select the AU variable from the Class Name box on top of your code window and then select a desired event from the Method Name box on the right, for example FrameAcquired. The following fragment will be added to your code:

```
Private Sub AU_FrameAcquired() Handles AU.FrameAcquired  
End Sub
```

6. When you are done using the ActiveUSB object, destroy it with the following command:

```
AU = Nothing
```

C#

1. Right click on your application in the Solution Explorer, select Add Reference.../COM, then select *ActiveUSB* Control from the list.

2. Declare a global *ActiveUSB*-type object variable:

```
ActiveUSBLib.ActiveUSB AU;
```

3. Instantiate an *ActiveUSB* object and assign it to the variable:

```
AU = new ActiveUSBLib.ActiveUSB();
```

4. Start using *ActiveUSB* properties and methods, for example:

```
AU.Camera = 0  
AU.Acquire=true; 'starting automatic acquisition  
AU.ShowProperties (true, 0); 'displaying built-in property pages
```

2.6 Working with multiple cameras

In general, interfacing to multiple cameras is as easy as dropping a few *ActiveUSB* objects on the surface of your application.

1. Start by creating a new project. Depending on the development environment you are using, refer to one of the following chapters for more details:

[Visual Basic](#)

[Visual C++](#)

[VB.NET](#)

[Visual C#](#)

2. Configure each *ActiveUSB* object for a different camera by clicking a corresponding *ActiveUSB* window and modifying the *Camera* property. *Do not configure different ActiveUSB objects for the same camera. Multiple instances of ActiveUSB cannot acquire video from the same camera.* For the same reason, if you run several *ActiveUSB*-based applications, make sure that each of them connects to a different camera.

3. Add *FrameAcquired* event handlers per each *ActiveUSB* object following the procedure for your language environment. Due to the multithreading nature of *ActiveUSB*, they will not interfere with each other.

4. Add the acquisition command for each *ActiveUSB* object to initiate the video streaming. In VB.NET you might have the following lines in your code:

```
AxActiveUSB1.Acquire = True
```

```
AxActiveUSB2.Acquire = True
```

```
AxActiveUSB3.Acquire = True
```

5. If you want to process the situation when a camera is being plugged or unplugged, add *CameraPlugged* and *CameraUnplugged* events to your code.

6. For more information refer to the code of the *MultiUcam* sample.

The same general rules apply if you use *DirectShow*. You can run several instances of the *Video Capture Source Filter* in your system or in your application provided each instance is configured for a different camera. If you execute several copies of a *DirectShow*-based video capture application such as Microsoft's *Amcap*, each of them will attempt to automatically configure itself for a different camera and memorize corresponding camera settings in the system registry upon exiting. As an alternative, you can use the [FilterConfig](#) utility to register several GigE Vision *DirectShow* devices in the system, each one associated with a specific camera. For more information refer to [DirectShow Programming Reference](#).

3 ActiveX Reference

This chapter provides a complete list of properties, methods and events of the *ActiveUSB* object.

[Properties](#)

[Methods](#)

[Events](#)

[Property Pages](#)

3.1 Properties

ActiveUSB properties are divided into several categories and can be modified in a Property Window of your development environment, or in *ActiveUSB* property pages.

Source Category

<u>Camera</u>	Returns or sets the index of the currently selected camera
<u>Acquire</u>	Enables/disables the continuous acquisition mode
<u>Timeout</u>	Returns or sets the frame acquisition timeout in seconds
<u>AcquisitionMode</u>	Returns or sets the acquisition mode of the currently selected camera
<u>AcquisitionFrameCount</u>	Returns or sets the number of frames for the multi-acquisition mode
<u>TestImageSelector</u>	Returns or sets the mode of generating internal test images
<u>TriggerSelector</u>	Returns or sets the configuration of a trigger signal
<u>Trigger</u>	Enables/disables the currently selected trigger
<u>TriggerSource</u>	Returns or sets the signal source for the selected trigger
<u>TriggerActivation</u>	Returns or sets the activation mode for the selected trigger
<u>TriggerDelay</u>	Returns or sets the value of the trigger delay for the selected trigger

Format Category

Format	Returns or sets the currently selected pixel format
AcquisitionFrameRate	Returns or sets the camera's frame rate
SizeX	Returns or sets the width of the partial scan window
SizeY	Returns or sets the height of the partial scan window
OffsetX	Returns or sets the horizontal offset of the partial scan window
OffsetY	Returns or sets the vertical offset of the partial scan window
BinningX	Returns or sets the number of horizontal photo-cells to be combined together
BinningY	Returns or sets the number of vertical photo-cells to be combined together
DecimationX	Returns or sets the horizontal subsampling of the image
DecimationY	Returns or sets the vertical subsampling of the image

Analog Category

<u>ExposureMode</u>	Returns or sets the operation mode of the exposure
<u>ExposureTime</u>	Returns or sets the camera's exposure time
<u>ExposureAuto</u>	Returns or sets the automatic exposure mode
<u>GainSelector</u>	Returns or sets the channel to be configured by the gain adjustments
<u>Gain</u>	Returns or sets the camera's gain
<u>GainAuto</u>	Returns or sets the automatic gain mode
<u>BlackLevelSelector</u>	Returns or sets the channel to be configured by the black level adjustments
<u>BlackLevel</u>	Returns or sets the camera's black level
<u>BlackLevelAuto</u>	Returns or sets the automatic black level mode
<u>BalanceRatioSelector</u>	Returns or sets the channel to be configured by the balance ratio adjustment
<u>BalanceRatio</u>	Returns or sets the balance ratio of the selected color channel
<u>BalanceWhiteAuto</u>	Returns or sets the automatic white balance mode
<u>Gamma</u>	Returns or sets the camera's gamma value

Input/Output Category

<u>LineSelector</u>	Returns or sets the physical I/O line to configure
<u>LineMode</u>	Returns or sets the input or output mode of the selected line
<u>LineSource</u>	Returns or sets the signal source for the selected line
<u>LineFormat</u>	Returns or sets the electrical format for the selected line
<u>LineInverter</u>	Returns or sets the inverted state for the selected line
<u>UserOutputSelector</u>	Selects the bit of the User Output register to be set
<u>UserOutputValue</u>	Returns or sets the value of the selected bit of the User Output register
<u>UserSetSelector</u>	Returns or assigns the user set to load, save or configure

Processing Category

<u>Affinity</u>	Returns or sets the number of logical CPUs used in image processing algorithms
<u>Flip</u>	Flips the image horizontally or/and vertically
<u>Rotate</u>	Rotates the image at the specified angle
<u>Bayer</u>	Enables/disable the Bayer conversion and selects the conversion method
<u>BkgName</u>	Returns or sets the name prefix for background data
<u>BkgCorrect</u>	Returns or sets the background correction mode
<u>ColorCorrect</u>	Enables/disables the color correction mode
<u>HotPixelCorrect</u>	Enables/disables the hot pixel correction mode
<u>HotPixelLevel</u>	Returns or sets the threshold level to be used in the hot pixel correction mode
<u>Integrate</u>	Enables/disables the frame integration operation and selects the integration mode
<u>IntegrateWnd</u>	Returns or sets the number of frames to be used in the Integrate operation
<u>LUTMode</u>	Enables/disables the lookup talbe operation on the video frames
<u>LensCorrect</u>	Enables/disables the lens distortion correction

Display Category

BackColor	Returns or sets the background color of the control window
Display	Enables/disables automatic live display in the control window
AntiTearing	Enables/disables the anti-tearing feature for the live video display
MonitorSync	Enables/disables the monitor synchronization mode
Edge	Enables/disables the 3D look of the control window
ScrollBars	Enables/disables the scroll bars in the control window
ScrollX	Returns or sets the current horizontal scroll position for the video window
ScrollY	Returns or sets the current vertical scroll position for the video window
Palette	Returns or sets the number of the current live video palette
Magnification	Returns or sets the magnification factor for the live video display
Font	Returns or sets the font for drawing text in the image
Overlay	Enables/disables the overlay
OverlayColor	Returns or sets the color of the overlay graphics
OverlayFont	Returns or sets the font for drawing text in the overlay
Alpha	Shows/hides the alpha plane over the live video

3.1.1 Acquire

Description

Enables/disables the continuous acquisition mode. If the acquisition mode is enabled and the [Display](#) property is turned on, the live video will be displayed in the control window.

Syntax

[VB]
`objActiveUSB.Acquire [= Value]`

[C/C++]
`HRESULT get_Acquire (bool *pValue);`
`HRESULT put_Acquire(bool Value);`

Data Type [VB]

Boolean

Parameters [C/C++]

pValue [out,retval]
Pointer to the Boolean that is TRUE if the continuous acquisition is enable, or FALSE otherwise
Value [in]
Set to TRUE to enable the continuous acquisition, or set to FALSE otherwise

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example activates the continuous acquisition mode upon clicking the Start button:

```
Private Sub cmdStart_Click()  
On Error GoTo err_cmdStart  
    ActiveUSB1.Acquire=True  
    Exit Sub  
err_cmdStart:  
    MsgBox Err.Description  
End Sub
```

Remarks

If this property is set to TRUE, *ActiveUSB* will continuously acquire the video into the internal image memory. The [FrameAcquired](#) event will be generated upon completing each frame. To access the content of the image memory, use [GetPixel](#), [GetLine](#), [GetImageWindow](#) and [GetImageData](#) methods.

Note that this property works in both the design and run-time modes. If your development environment

does not close the design view prior to going to the run-time mode, a conflict will occur between two *ActiveUSB* objects trying to acquire the video from the same camera. In order to prevent this, it is recommended to keep the **Acquire** property set to FALSE in the design mode and turn it on by an explicit command in the program code (see the example above).

Use [AcquisitionMode](#) to set up the desired acquisition mode prior to setting **Acquire** to TRUE.

3.1.2 AcquisitionFrameCount

Description

Returns or sets the number of frames to be acquired in MultiFrame [AcquisitionMode](#)

Syntax

[VB]
`objActiveUSB.AcquisitionFrameCount [= Value]`

[C/C++]
`HRESULT get_AcquisitionFrameCount(long *pValue);`
`HRESULT put_AcquisitionFrameCount(long Value);`

Data Type [VB]

Long

Parameters [C/C++]

pValue [out,retval]
Pointer to the current frame count value
Value [in]
The frame count value to be set

Return Values

S_OK
Success
E_NOINTERFACE
The feature is not available for the selected camera
E_INVALIDARG
The value is out of range
E_FAIL
Failure to set the feature value

Example

The following VB example activates the acquisition of 100 frames:

```
ActiveUSB1.AcquisitionFrameCount=100  
ActiveUSB1.AcquisitionMode = "MultiFrame"  
ActiveUSB1.Acquire=TRUE
```

Remarks

This property is available only if the currently selected camera supports the *AcquisitionFrameCount* feature per GenICam standard.

3.1.3 AcquisitionFrameRate

Description

Returns or sets the absolute value of the camera frame rate in Hz or raw units.

Syntax

[VB]
`objActiveUSB.AcquisitionFrameRate [= Value]`

[C/C++]
`HRESULT get_AcquisitionFrameRate(float *pValue);`
`HRESULT put_AcquisitionFrameRate(float Value);`

Data Type [VB]

Single

Parameters [C/C++]

pValue [out,retval]
Pointer to the current frame rate value
Value [in]
The frame rate to be set

Return Values

S_OK
Success
E_NOINTERFACE
The feature is not available for the selected camera
E_INVALIDARG
The value is out of range
E_FAIL
Failure to set the feature value

Example

The following VB example sets the frame rate to 30:

```
ActiveUSB1.AcquisitionFrameRate = 30
```

Remarks

The feature controls the rate at which frames are captured when the frame trigger is disabled. The valid range of the frame rate values can be obtained by the [GetAcquisitionFrameRateMin](#) and [GetAcquisitionFrameRateMax](#) methods.

Note that the property is available only if the currently selected camera supports one of the following GenICam features: *AcquisitionFrameRate*, *AcquisitionFrameRateAbs*, *AcquisitionFrameRateRaw*.

3.1.4 AcquisitionMode

Description

Returns or sets the acquisition mode of the camera which defines how many frames will be captured when [Acquire](#) is set to true. Can be one of the following values:

"SingleFrame"
One frame is captured

"MultiFrame"
The number of frames to capture is specified by [AcquisitionFrameCount](#)

"Continuous"
Frames are captured continuously until Acquire is set to FALSE

Syntax

[VB]
`objActiveUSB.AcquisitionMode [= Value]`

[C/C++]
`HRESULT get_AcquisitionMode(bstr *pValue);`
`HRESULT put_AcquisitionMode(bstr Value);`

Data Type [VB]

String

Parameters [C/C++]

pValue [out, retval]
Pointer to the string specifying the current acquisition mode

Value [in]
The acquisition mode to be set

Return Values

S_OK
Success

E_NOINTERFACE
The feature is not available for the selected camera

E_INVALIDARG
The value is not part of the enumerated set

E_FAIL
Failure to set the feature value

Example

The following VB example activates the acquisition of 100 frames:

```
ActiveUSB1.AcquisitionFrameCount=100
ActiveUSB1.AcquisitionMode = "MultiFrame"
ActiveUSB1.Acquire=TRUE
```

Remarks

Depending on a camera model, other device-specific values may be available for this feature. The list of valid values can be retrieved using [GetEnumList](#).

The desired acquisition mode must be selected before setting [Acquire](#) to TRUE.

3.1.5 Affinity

Description

Returns or sets the number of logical CPUs (cores) used in image processing algorithms.

Syntax

```
[VB]  
objActiveUSB.Affinity [= Value]
```

```
[C/C++]  
HRESULT get_Affinity ( short *pAffinity );  
HRESULT put_Affinity ( short Affinity );
```

Data Type [VB]

Integer

Parameters [C/C++]

pAffinity[out,retval]
Pointer to the currently selected Affinity.
Affinity [in]
The value of affinity to set. If 0, all available logical CPUs will be used.

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB code instructs *ActiveUSB* to use 4 logical CPUs:

```
ActiveUSB1.Affinity = 4
```

Remarks

This property lets you select the number of logical CPUs that will be used for image processing, such as [Bayer Conversion](#). By default *ActiveUSB* splits the processing tasks among all logical CPUs (cores) available on the system. You may want to reduce the amount of utilized cores for benchmarking or to free CPU resources for other tasks.

The default value of this property is zero which makes *ActiveUSB* to use to all available logical CPUs.

3.1.6 Alpha

Description

Shows/hides the alpha plane over the live video.

Syntax

```
[VB]  
objActiveUSB.Alpha [= Value]
```

```
[C/C++]  
HRESULT get_Alpha ( bool *pAlpha );  
HRESULT put_Alpha( bool Alpha );
```

Data Type [VB]

Boolean

Parameters [C/C++]

pAlpha [out,retval]
Pointer to the Boolean that is TRUE if the alpha plane is enable, or FALSE otherwise

Alpha [in]
Set to TRUE to enable the alpha plane, or set to FALSE to disable it.

Return Values

S_OK
Success

E_FAIL
Failure.

Example

The following VB example moves a transparent ellipse over the live video creating an animation effect:

```
Dim x As Integer  
Dim y As Integer  
Dim sy As Integer  
Dim sx As Integer  
  
Private Sub Form_Load()  
ActiveUSB1.Acquire=True  
ActiveUSB1.Alpha=True  
End Sub  
  
Private Sub ActiveUSB1_FrameAcquired()  
ActiveUSB1.DrawAlphaEllipse x, 10, 400 + x, 200, 0, 0, 0, 0, 0  
x = x + 1  
y = y + 1  
If x = sx Then  
x = 0  
End If  
If y = sy Then  
y = 0  
End If
```

```
ActiveUSB1.DrawAlphaEllipse x, 10, 400 + x, 200, 0, 255, 255, 0, 30  
End Sub
```

Remarks

The **Alpha** feature lets you display multi-colored custom graphics and text over the live video image. Unlike the [Overlay](#) which has a solid color, objects in the alpha plane can be assigned different colors and opacity.

Setting this property to true causes ActiveUSB to show the alpha plane. Disabling this property hides the alpha plane. Note that hiding the alpha plane does not erase graphics and text from it. To clear the alpha plane, use [DrawAlphaClear](#). For more information see [DrawAlphaPixel](#), [DrawAlphaLine](#), [DrawAlphaRectangle](#), [DrawAlphaEllipse](#), [DrawAlphaText](#).

Note that using the alpha plane may increase the CPU load of your application.

3.1.7 AntiTearing

Description

Enables/disables the anti-tearing feature for the live video display in the control window.

Syntax

```
[VB]  
objActiveUSB.AntiTearing [= Value]
```

```
[C/C++]  
HRESULT get_AntiTearing ( bool *pValue );  
HRESULT put_AntiTearing( bool Value );
```

Data Type [VB]

Boolean

Parameters [C/C++]

pValue [out,retval]
Pointer to the Boolean that is TRUE if the anti-tearing is enable, or FALSE otherwise
Value [in]
Set to TRUE to enable the anti-tearing, or set to FALSE to disable it

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example enables the anti-tearing feature:

```
ActiveUSB1.AntiTearing = True  
MsgBox ActiveUSB1.AntiTearing
```

Remarks

Tearing is a display artifact caused by the difference between the frame rate of the camera and the refresh rate of the monitor. Tearing can be noticed on the live video as horizontal splits between the upper and bottom parts of frames with highly dynamic contest. If **AntiTearing** is set to TRUE, the display update will be synchronized with the vertical blank period of the monitor thus eliminating tearing artifacts.

When **AntiTearing** is enable, the effective frame rate will be limited by the monitor's refresh rate. You can also experience a performance drop resulting in an additional CPU load.

Note - some advanced graphic controllers may have the anti-tearing functionality implemented in the

*hardware. If your system have such a graphic card, do not enable the **AntiTearing** property.*

3.1.8 BackColor

Description

Returns or sets the background color of the control window.

Syntax

[VB]
`objActiveUSB.BackColor [= Color]`

[C/C++]
`HRESULT get_BackColor(OLE_COLOR& pColor);`
`HRESULT put_BackColor(OLE_COLOR Color);`

Data Type [VB]

RGB color

Parameters [C/C++]

pColor [out,retval]
Pointer to the current background color
Color [in]
The background color to be set

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example sets the background color of the control to light gray:

```
ActiveUSB1.BackColor = RGB (192, 192, 192)
```

Remarks

When the control displays a live video, its background is only visible if the size of the video display in the horizontal or vertical dimension is less than the size of the control window.

BackColor is an ambient property, therefore its default value is defined by the color of the form on which the control resides. Changing the background color of the container application will cause an equivalent change in the **BackColor** property of *ActiveUSB*.

3.1.9 BalanceRatio

Description

Returns or sets the ratio (amplification factor) of the selected color component in absolute or raw units. Used for white balancing.

Syntax

[VB]
`objActiveUSB.BalanceRatio [= Value]`

[C/C++]
`HRESULT get_BalanceRatio(float *pValue);`
`HRESULT put_BalanceRatio(float Value);`

Data Type [VB]

Single

Parameters [C/C++]

pValue [out,retval]
Pointer to the white balance ratio
Value [in]
The white balance ratio to be set

Return Values

S_OK
Success
E_NOINTERFACE
The feature is not available for the selected camera
E_INVALIDARG
The value is out of range
E_FAIL
Failure to set the feature value

Example

The following VB example adjusts white balance factors for different color channels:

```
ActiveUSB1.BalanceRatioSelector="Red"  
ActiveUSB1.BalanceRatio = 1.2  
ActiveUSB1.BalanceRatioSelector="Green"  
ActiveUSB1.BalanceRatio = 1.4  
ActiveUSB1.BalanceRatioSelector="Blue"  
ActiveUSB1.BalanceRatio = 1.
```

Remarks

This property changes the ratio of the selected color component relative to the base level of the

component. The valid range of the ratio values can be obtained by the [GetBalanceRatioMin](#) and [GetBalanceRatioMax](#) methods. Before using **BalanceRatio**, select a corresponding channel with [BalanceRatioSelector](#).

Note that the property is available only if the currently selected camera supports one of the following GenICam features: *BalanceRatio*, *BalanceRatioAbs*, *BalanceRatioRaw*.

3.1.10 BalanceRatioSelector

Description

Returns or sets the color channel to be controlled by [BalanceRatio](#). Used for white balancing and can be one of the following values:

- "Red"
Gain adjustments will be applied to the red channel.
- "Green"
Gain adjustments will be applied to the green channel.
- "Blue"
Gain adjustments will be applied to the blue channel.
- "Y"
Gain adjustments will be applied to Y channel.
- "U"
Gain adjustments will be applied to U channel.
- "V"
Gain adjustments will be applied to V channel.

Syntax

[VB]
`objActiveUSB.BalanceRatioSelector [= Value]`

[C/C++]
`HRESULT get_BalanceRatioSelector(string *pValue);`
`HRESULT put_BalanceRatioSelector(string Value);`

Data Type [VB]

String

Parameters [C/C++]

- pValue* [out, retval]
Pointer to the string specifying the current color channel for white balancing
- Value* [in]
The color channel to be set

Return Values

- S_OK
Success
- E_NOINTERFACE
The feature is not available for the selected camera
- E_INVALIDARG
The value is not part of the enumerated set
- E_FAIL
Failure to set the feature value

Example

The following VB example demonstrates the use of a combo box to switch between different white balance channels:

The following VB example adjusts white balance factors for different color channels:

```
ActiveUSB1.BalanceRatioSelector="Red"  
ActiveUSB1.BalanceRatioAbs = 1.2  
ActiveUSB1.BalanceRatioSelector="Green"  
ActiveUSB1.BalanceRatioAbs = 1.4  
ActiveUSB1.BalanceRatioSelector="Blue"  
ActiveUSB1.BalanceRatioAbs = 1.
```

Remarks

Depending on a camera model, other device-specific values may be available for this feature. The list of valid values can be retrieved using [GetEnumList](#).

Note that the property is available only if the currently selected camera supports the *BalanceRatioSelector* feature per GenICam standard.

3.1.11 BalanceWhiteAuto

Description

Returns or sets the automatic white balance control (AWB) mode. Can be one of the following values:

"Off"

White balance is manually controlled using [BalanceRatioSelector](#) and [BalanceRatio](#)

"Once"

The camera sets the optimal balance ratios for each color channel and returns to the "Off" state

"Continuous"

The camera constantly performs white balancing

Syntax

```
[VB]  
objActiveUSB.GainAuto [= Value]
```

```
[C/C++]  
HRESULT get_GainAuto( string *pValue );  
HRESULT put_GainAuto( string Value );
```

Data Type [VB]

String

Parameters [C/C++]

pValue [out, retval]
Pointer to the string specifying the white balance mode

Value [in]
The white balance mode to be set

Return Values

S_OK
Success

E_NOINTERFACE
The feature is not available for the selected camera

E_INVALIDARG
The value is not part of the enumerated set

E_FAIL
Failure to set the feature value

Example

The following VB example demonstrates the use of a button to perform a "one push" white balancing.

```
Private Sub Command1_Click()  
ActiveUSB1.BalanceWhiteAuto = "Once"  
End Sub
```

Remarks

Depending on a camera model, other device-specific values may be available for this feature. The list of valid values can be retrieved using [GetEnumList](#).

Note that the property is available only if the currently selected camera supports the *BalanceWhiteAuto* feature per GenICam standard.

3.1.12 Bayer

Description

Enables/disables the Bayer conversion mode and selects the Bayer conversion method.

Syntax

```
[VB]  
objActiveUSB.Bayer [= Value]
```

```
[C/C++]  
HRESULT get_Bayer ( short *pBayer );  
HRESULT put_Bayer( short Bayer );
```

Data Type [VB]

Integer

Parameters [C/C++]

pBayer [out,retval]

Pointer to the ordinal number of the currently selected Bayer filter, zero if Bayer conversion is disabled.

Bayer [in]

Set to zero to disable Bayer conversion, or set to values from 1 to 3 to select one of the following Bayer filters:

- 1 - Nearest Neighbour filter. Missing pixels are substituted with adjacent pixels of the same color.
- 2 - Bilinear filter. Calculates the values of missing pixels by performing bilinear interpolation of the adjacent pixels.
- 3 - High Quality Linear filter. Calculates the values of missing pixels based on the Malvar, He and Cutler algorithm.
- 4 - Chrominance filter. Interpolates the values of missing pixels based on chrominance gradients.

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example activates the bilinear Bayer filter:

```
ActiveUSB1.Bayer = 1
```

Remarks

Bayer images are usually generated by a single-chip CCD camera, which has a color filter mosaic array (CFA) installed in front of the sensor. The most frequently used Bayer pattern has the following layout:

```
G B G B R
R G R G R
G B G B G
R G R G R
```

Each pixel value in a Bayer image corresponds to the intensity of the pixel behind the corresponding color filter. The conversion from such a grayscale image to RGB image is typically done on the camera itself, however some cameras (known as Bayer cameras) output a raw Bayer image unchanged. The Bayer transforms such an image into RGB image by performing real-time Bayer color reconstruction (demaosaicing). The layout of the Bayer array is defined by the current [Format](#).

Note that the higher the ordinal number of the selected Bayer filter is, the higher the quality of the resulting image is and the higher the CPU load is. If the application speed is critical, consider using the Nearest Neighbour filter.

3.1.13 BinningX

Description

Returns or sets the number of horizontal photo-sensitive cells that must be combined together.

Syntax

```
[VB]  
objActiveUSB.BinningX [= Value]
```

```
[C/C++]  
HRESULT get_BinningX( long *pBinningX );  
HRESULT put_BinningX( long BinningX );
```

Data Type [VB]

Long

Parameters [C/C++]

pBinningX [out,retval]
Pointer to the currently set horizontal binning
BinningX [in]
The horizontal binning to be set

Return Values

S_OK
Success
E_FAIL
Failure.
E_INVALIDARG
Invalid property value.

Example

This VB example sets the horizontal binning to 2:

```
ActiveUSB1.BinningX = 2  
MsgBox ActiveUSB1.BinningX
```

Remarks

This property has a net effect of increasing the intensity (or signal to noise ratio) of the pixel value and reducing the horizontal size of the image. A value of 1 indicates that no horizontal binning is performed by the camera. Note that the property is available only if the currently selected camera supports the *BinningHorizontal* feature per GenICam standard. Any change in the **BinningX** property will cause [SizeX](#), and [OffsetX](#) to change accordingly.

3.1.14 BinningY

Description

Returns or sets the number of vertical photo-sensitive cells that must be combined together.

Syntax

[VB]
`objActiveUSB.BinningY [= Value]`

[C/C++]
`HRESULT get_BinningY(long *pBinningY);`
`HRESULT put_BinningY(long BinningY);`

Data Type [VB]

Long

Parameters [C/C++]

pBinningY [out,retval]
Pointer to the currently set vertical binning
BinningY [in]
The vertical binning to be set

Return Values

S_OK
Success
E_FAIL
Failure.
E_INVALIDARG
Invalid property value.

Example

This VB example sets the vertical binning to 2:

```
ActiveUSB1.BinningY = 2  
MsgBox ActiveUSB1.BinningY
```

Remarks

This property has a net effect of increasing the intensity (or signal to noise ratio) of the pixel value and reducing the vertical size of the image. A value of 1 indicates that no vertical binning is performed by the camera. Note that the property is available only if the currently selected camera supports the *BinningVertical* feature per GenICam standard. Any change in the **BinningY** property will cause [SizeY](#), and [OffsetY](#) to change accordingly.

3.1.15 BkgCorrect

Description

Returns or sets the mode for the background correction. The values from 0 to 2 correspond to the following modes:

0 - *None*

No background correction is performed.

1 - *Dark*

The dark field (offset) background correction is applied to each acquired frame.

2 - *Flat*

The flat field (gain) background correction is applied to each acquired frame.

Syntax

[VB]

```
objActiveUSB.BkgCorrect [ = Value ]
```

[C/C++]

```
HRESULT get_BkgCorrect( short *pCorrect );  
HRESULT put_BkgCorrect( short Correct );
```

Data Types [VB]

Integer

Parameters [C/C++]

pCorrect [out,retval]

Pointer to the current background correction mode

Correct [in]

Background correction mode to be set

Return Values

S_OK

Success

Example

The following VB example sets the dark field correction mode:

```
ActiveUSB1.BkgCorrect=1
```

Remarks

The dark field (offset) correction is used to compensate for the pixel-to-pixel difference in the dark current of the camera sensor. The correction is performed by subtracting background pixel values from corresponding pixel values of the original image. The dark field correction requires the dark field background image for the current video mode to have been stored on the hard drive.

The flat field (gain) correction is used to compensate for the variations in pixel-to-pixel sensitivity as

well as non-uniformity of the illumination. The correction algorithm is based on the following formula:

$$C_{x,y} = \frac{(I_{x,y} - B_{x,y})(W_{max} - B_{x,y})}{(W_{x,y} - B_{x,y})}$$

where

- $I_{x,y}$ is a pixel value of the original image
- $B_{x,y}$ is a pixel value of the dark field image
- $W_{x,y}$ is a pixel value of the bright field image
- W_{max} is a maximum value of the bright field image
- $C_{x,y}$ is a new pixel value of the corrected image

The flat field correction requires the bright field background image for the current video mode to have been stored on the hard drive. If the dark field image for the currently selected video mode does not exist on the hard drive, the value of 0 will be used in place of B .

If corresponding background images are not found on the hard drive, no correction will be performed. See [SaveBkg](#) for more details on preparing and saving background data.

3.1.16 BkgName

Description

Returns or sets the name prefix for the background data files.

Syntax

```
[VB]  
objActiveUSB.BkgName =[ Value ]
```

```
[C/C++]  
HRESULT get_BkgName( bstr *pName );  
HRESULT put_BkgName( bstr Name );
```

Data Types [VB]

String

Parameters [C/C++]

pName [out, retval]
Pointer to the string containing the name prefix of the background data files.
Name [in]
The background name prefix to be set.

Return Values

S_OK
Success
E_FAIL
Failure.

Example

The following VB example assigns the name for background files and stores the current image as a dark field:

```
ActiveUSB1.BkgName="MyBackground"  
ActiveUSB1.SaveBkg (1)
```

Remarks

Background data are stored as raw image files in the Bkgnd subfolder of ActiveUSB program directory (typically C:\Program Files\ActiveUSB\Bkgnd). The full name of a background file is composed of the **BkgName** prefix and substrings indicating the camera index, video mode and background type. E.g., if the background name is assigned as in the example above and the camera operates in the video mode #3, the background image file will be stored under the name "MyBackground_cam0_mode3_dark.raw"

3.1.17 BlackLevel

Description

Returns or sets the absolute or raw value of the camera's black level.

Syntax

[VB]
`objActiveUSB.BlackLevel [= Value]`

[C/C++]
`HRESULT get_BlackLevel(float *pValue);`
`HRESULT put_BlackLevel(float Value);`

Data Type [VB]

Single

Parameters [C/C++]

pValue [out,retval]
Pointer to the current black level value
Value [in]
The black level value to be set

Return Values

S_OK
Success
E_NOINTERFACE
The feature is not available for the selected camera
E_INVALIDARG
The value is out of range
E_FAIL
Failure to set the feature value

Example

The following VB example demonstrates the use of a scroll control for real-time adjustment of the black level value.

```
Private Sub Form_Load()  
HScroll1.Value = ActiveUSB1.BlackLevel  
HScroll1.Min = ActiveUSB1.GetBlackLevel  
HScroll1.Max = ActiveUSB1.GetBlackLevel  
ActiveUSB1.Acquire=True  
End Sub  
  
Private Sub HScroll1_Scroll()  
ActiveUSB1.BlackLevel = HScroll1.Value  
End Sub
```

Remarks

This property changes the black level of the video by adding a constant amount of luminance to each pixel. The valid property range can be retrieved by the [GetBlackLevelMin](#) and [GetBlackLevelMax](#) methods.

Note that the property is available only if the currently selected camera supports one of the following GenICam features: *BlackLevel*, *BlackLevelAbs*, *BlackLevelRaw*.

3.1.18 BlackLevelAuto

Description

Returns or sets the automatic black level control mode. Can be one of the following values:

"Off"

Black level is manually controlled using [BlackLevel](#)

"Once"

The camera sets the optimal black level and returns to the "Off" state

"Continuous"

The camera constantly adjusts the black level

Syntax

[VB]

```
objActiveUSB.BlackLevelControl [= Value]
```

[C/C++]

```
HRESULT get_BlackLevelAuto( bstr *pValue );  
HRESULT put_BlackLevelAuto( bstr Value );
```

Data Type [VB]

String

Parameters [C/C++]

pValue [out, retval]

Pointer to the string specifying the black level control mode

Value [in]

The black level control mode to be set

Return Values

S_OK

Success

E_NOINTERFACE

The feature is not available for the selected camera

E_INVALIDARG

The value is not part of the enumerated set

E_FAIL

Failure to set the feature value

Example

The following VB example demonstrates the use of a check box to switch between the manual and automatic black level control.

```
Private Sub Check1_Click()  
If Check1.Value = 1 Then  
ActiveUSB1.BlackLevelAuto = "On"  
Else  
ActiveUSB1.BlackLevelAuto = "Off"
```

```
End If  
End Sub
```

Remarks

Depending on a camera model, other device-specific values may be available for this feature. The list of valid values can be retrieved using [GetEnumList](#).

Note that the property is available only if the currently selected camera supports the *BlackLevelAuto* feature per GenICam standard.

3.1.19 BlackLevelSelector

Description

Returns or sets the channel to be controlled by black level adjustments. Can be one of the following values:

- "All"
Black level adjustments will be applied to all channels.
- "Red"
Black level adjustments will be applied to the red channel.
- "Green"
Black level adjustments will be applied to the green channel.
- "Blue"
Black level adjustments will be applied to the blue channel.
- "Y"
Black level adjustments will be applied to Y channel.
- "U"
Black level adjustments will be applied to U channel.
- "V"
Black level adjustments will be applied to V channel.

Syntax

[VB]
`objActiveUSB.BlackLevelSelector [= Value]`

[C/C++]
`HRESULT get_BlackLevelSelector(string *pValue);`
`HRESULT put_BlackLevelSelector(string Value);`

Data Type [VB]

String

Parameters [C/C++]

- pValue* [out, retval]
Pointer to the string specifying the black level channel setting
- Value* [in]
The black level channel to be set

Return Values

- S_OK
Success
 - E_NOINTERFACE
The feature is not available for the selected camera
 - E_INVALIDARG
The value is not part of the enumerated set
 - E_FAIL
Failure to set the feature value
-

Example

The following VB example demonstrates the use of a combo box to switch between different black level channels:

```
Private Sub Form_Load()  
Lst = ActiveUSB1.GetEnumList("BlackLevelSelector")  
For i = 0 To UBound(Lst)  
  Comb1.AddItem (Lst(i))  
Next  
Comb1.ListIndex = ActiveUSB1.BlackLevelSelector  
End Sub  
  
Private Sub Comb1_Click()  
ActiveUSB1.BlackLevelSelector = Comb1.Text  
End Sub
```

Remarks

Depending on a camera model, other device-specific values may be available for this feature. The list of valid values can be retrieved using [GetEnumList](#).

Note that the property is available only if the currently selected camera supports the *BlackLevelSelector* feature per GenICam standard.

3.1.20 Camera

Description

Returns or sets the index of the currently selected camera. If no USB3 Vision™ camera is connected to the system, the property will return -1.

Syntax

[VB]
`objActiveUSB.Camera [= Value]`

[C/C++]
`HRESULT get_Camera(long *pCamera);`
`HRESULT put_Camera(long Camera);`

Data Type [VB]

Long

Parameters [C/C++]

pCamera [out, retval]
Pointer to the camera's index
Camera [in]
The index of the camera to be selected

Return Values

S_OK
Success
E_FAIL
The selected camera doesn't comply with USB3 Vision™ specifications
E_INVALIDARG
The camera's index is out of range.

Example

This VB example initializes a combo box with camera names and uses it to switch between the cameras:

```
Private Sub Form_Load()  
CamLst = ActiveUSB1.GetCameraList  
For i = 0 To UBound(CamLst)  
Combo1.AddItem (CamLst(i))  
Next  
Combo1.ListIndex = 0  
ActiveUSB1.Acquire = True  
End Sub  
  
Private Sub Combo1_Click()  
ActiveUSB1.Camera = Combo1.ListIndex
```

End Sub

This VB example shows how to change camera assignment when using multiple *ActiveGigE* objects:

```
'Initial camera assignment
ActiveUSB1.Camera=0
ActiveUSB2.Camera=1
ActiveGige3.Camera=2
....
'Disconnect all cameras, then change camera assignment
ActiveUSB1.Camera=-1
ActiveUSB2.Camera=-1
ActiveUSB3.Camera=-1
ActiveUSB1.Camera=2
ActiveUSB2.Camera=1
ActiveUSB3.Camera=0
```

Remarks

The value of the property is a zero-based index into the list of USB3 Vision™ cameras discovered on the network. The list of camera names can be retrieved with [GetCameraList](#). If you use the property window of your development environment or *ActiveUSB*'s property pages, the **Camera** property field will be presented as a list box containing the indexes and names of all the connected cameras.

To disconnect the currently selected camera from *ActiveUSB* object, use -1 as the camera index. If your application uses multiple *ActiveUSB* objects connected to different cameras and you need to change the camera assignment, disconnect a camera from an object before assigning the camera to another object, or otherwise the new assignment will be denied.

3.1.21 ColorCorrect

Description

Enables/disables the color correction mode. Used in combination with [SetColorMatrix](#).

Syntax

[VB]
`objActiveUSB.ColorCorrect [= Value]`

[C/C++]
`HRESULT get_ColorCorrect (bool *pCorrect);`
`HRESULT put_ColorCorrect(bool Correct);`

Data Type [VB]

Boolean

Parameters [C/C++]

pCorrect [out,retval]
Pointer to the Boolean that is TRUE if the color correction is enabled, or FALSE otherwise
Correct [in]
Set to TRUE to enable the color correction, or set to FALSE to disable it

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example defines the color correction matrix and turns on the color correction:

```
ActiveUSB1.SetColorMatrix 0.85, 0.1, 0.05, 0.05, 0.98, -0.03, 0.13, -0.15,  
1.02  
ActiveUSB1.ColorCorrect=True
```

Remarks

Color correction is used to eliminate the overlap in the color channels caused by the fact that the light intended to be detected only by pixels of a certain color is partially seen by pixels of other colors. For more information see [SetColorMatrix](#).

3.1.22 DecimationX

Description

Returns or sets the horizontal subsampling of the image.

Syntax

[VB]

```
objActiveUSB.DecimationX [= Value]
```

[C/C++]

```
HRESULT get_DecimationX( long *pDecimationX );  
HRESULT put_DecimationX( long DecimationX );
```

Data Type [VB]

Long

Parameters [C/C++]

pDecimationX [out,retval]

Pointer to the currently set horizontal binning

DecimationX [in]

The horizontal decimation to be set

Return Values

S_OK

Success

E_FAIL

Failure.

E_INVALIDARG

Invalid property value.

Example

This VB example sets the horizontal decimation to 2:

```
ActiveUSB1.DecimationX = 2  
MsgBox ActiveUSB1.DecimationX
```

Remarks

Depending on the camera, the subsampling can be implemented by pixel dropping or by pixel averaging/dropping. This has a net effect of reducing the horizontal size of the image by the specified decimation factor. A value of 1 indicates that no horizontal decimation is performed by the camera. Note that the property is available only if the currently selected camera supports the *DecimationHorizontal* feature per GenICam standard. Any change in the **DecimationX** property will cause [SizeX](#), and [OffsetX](#) to change accordingly.

3.1.23 DecimationY

Description

Returns or sets the vertical subsampling of the image.

Syntax

[VB]
`objActiveUSB.DecimationY [= Value]`

[C/C++]
`HRESULT get_DecimationY(long *pDecimationY);`
`HRESULT put_DecimationY(long DecimationY);`

Data Type [VB]

Long

Parameters [C/C++]

pDecimationY [out,retval]
Pointer to the currently set vertical binning
DecimationY [in]
The vertical decimation to be set

Return Values

S_OK
Success
E_FAIL
Failure.
E_INVALIDARG
Invalid property value.

Example

This VB example sets the vertical decimation to 2:

```
ActiveUSB1.DecimationY = 2  
MsgBox ActiveUSB1.DecimationY
```

Remarks

Depending on the camera, the subsampling can be implemented by pixel dropping or by pixel averaging/dropping. This has a net effect of reducing the vertical size of the image by the specified decimation factor. A value of 1 indicates that no vertical decimation is performed by the camera. Note that the property is available only if the currently selected camera supports the *DecimationVertical* feature per GenICam standard. Any change in the **DecimationY** property will cause [SizeY](#), and [OffsetY](#) to change accordingly.

3.1.24 Display

Description

Enables/disables live display in the control window. When this property is enabled, each frame will be automatically displayed upon acquisition.

Syntax

```
[VB]  
objActiveUSB.Display [= Value]
```

```
[C/C++]  
HRESULT get_Display ( bool *pDisplay );  
HRESULT put_Display( bool Display );
```

Data Type [VB]

Boolean

Parameters [C/C++]

pDisplay [out,retval]
Pointer to the Boolean that is TRUE if the live display is enable, or FALSE otherwise

Display [in]
Set to TRUE to enable the live display, or set to FALSE to disable it

Return Values

S_OK
Success

E_FAIL
Failure.

Example

This VB example disables the live display and uses the [FrameAcquired](#) event to invert pixel value in the bottom left corner of the current frame and display the processed frame in real time.

```
Private Sub Form_Load()  
ActiveUSB1.Display = False  
ActiveUSB1.Acquire = True  
End Sub  
  
Private Sub ActiveUSB1_FrameAcquired()  
a = ActiveUSB1.GetImageData  
For x = 0 To 200  
For y = 0 To 200  
a(x, y) = 255 - a(x, y)  
Next  
Next  
ActiveUSB1.Draw  
End Sub
```

Remarks

When you create a new instance of the control, this property is enabled by default. You should disable the live display when you want to perform real time image processing and display the processed image in the control window. After the processing is done, the current frame should be displayed by calling the [Draw](#) method.

When using [DirectShow Video Capture Filter](#), this property controls the internal conversion to RGB24 format which is required by image data access and image analysis methods, such as [GetImageData](#), [GetComponentData](#), [GetImageLine](#), [GetImageWindow](#), [GetHistogram](#), [GetImageStat](#) and others. Setting **Display** to FALSE will disable the conversion and provide original video formats on the output pin.

Note that disabling the live display does not turn off the image conversion performed on each frame. In order to disable the image conversion pipeline and receive only raw image data, set the [Magnification](#) property to -2.

3.1.25 Edge

Description

Enables/disables the 3D look (sunken edge) of the control window.

Syntax

```
[VB]  
objActiveUSB.Edge [= Value]
```

```
[C/C++]  
HRESULT get_Edge ( bool *pEdge );  
HRESULT put_Edge( bool Edge );
```

Data Type [VB]

Boolean

Parameters [C/C++]

pEdge [out,retval]
Pointer to the Boolean that is TRUE if the 3D look is enable, or FALSE otherwise

Edge [in]
Set to TRUE to enable the 3D look, or set to FALSE to disable it

Return Values

S_OK
Success

E_FAIL
Failure.

Example

This VB example enables the 3D look on the control window:

```
ActiveUSB1.Edge = True  
MsgBox ActiveUSB1.Edge
```

Remarks

If this property is set to TRUE, the sunken edge will appear around the border of the *ActiveUSB* control window.

3.1.26 ExposureAuto

Description

Returns or sets the automatic exposure (AE) mode. Can be one of the following values:

"Off"

Exposure is manually controlled using [ExposureTime](#)

"Once"

The camera sets the optimal exposure level and returns to the "Off" state

"Continuous"

The camera constantly adjusts the exposure level

Syntax

[VB]

```
objActiveUSB.ExposureAuto [= Value]
```

[C/C++]

```
HRESULT get_ExposureAuto( bstr *pValue );  
HRESULT put_ExposureAuto( bstr Value );
```

Data Type [VB]

String

Parameters [C/C++]

pValue [out, retval]

Pointer to the string specifying the exposure control mode

Value [in]

The exposure control mode to be set

Return Values

S_OK

Success

E_NOINTERFACE

The feature is not available for the selected camera

E_INVALIDARG

The value is not part of the enumerated set

E_FAIL

Failure to set the feature value

Example

The following VB example demonstrates the use of a check box to switch between the manual and automatic exposure control.

```
Private Sub Check1_Click()  
If Check1.Value = 1 Then  
ActiveUSB1.ExposureAuto = "On"  
Else  
ActiveUSB1.ExposureAuto = "Off"
```

```
End If  
End Sub
```

Remarks

Depending on a camera model, other device-specific values may be available for this feature. The list of valid values can be retrieved using [GetEnumList](#).

Note that the property is available only if the currently selected camera supports the *ExposureAuto* feature per GenICam standard.

3.1.27 ExposureMode

Description

Returns or sets the operation mode of the exposure (shutter). Can be one of the following values:

"Off"

Disables the exposure and lets the shutter open

"Timed"

Exposure time is set using [ExposureTime](#) or [ExposureAuto](#)

"TriggerWidth"

The camera uses the width of the current Frame or Line trigger signal pulse to control the exposure time.

"TriggerControlled"

The camera uses one or more trigger signals to control the exposure time independently from the Frame or Line triggers. See [TriggerSelector](#) for more details

Syntax

[VB]

```
objActiveUSB.ExposureMode [= Value]
```

[C/C++]

```
HRESULT get_ExposureMode( bstr *pValue );
```

```
HRESULT put_ExposureMode( bstr Value );
```

Data Type [VB]

String

Parameters [C/C++]

pValue [out, retval]

Pointer to the string specifying the exposure operational mode

Value [in]

The exposure operational mode to be set

Return Values

S_OK

Success

E_NOINTERFACE

The feature is not available for the selected camera

E_INVALIDARG

The value is not part of the enumerated set

E_FAIL

Failure to set the feature value

Example

The following VB example demonstrates the use of a combo box to switch between exposure operational modes.

```
Private Sub Form_Load()  
Lst = ActiveUSB1.GetEnumList("ExposureMode")
```

```
For i = 0 To UBound(Lst)
  Combol.AddItem (Lst(i))
Next
Combol.ListIndex = ActiveUSB1.ExposureMode
End Sub

Private Sub Combol_Click()
  ActiveUSB1.ExposureMode = Combol.Text
End Sub
```

Remarks

The property is available only if the currently selected camera supports the *ExposureMode* feature per GenICam standard.

3.1.28 ExposureTime

Description

Returns or sets the value of the camera exposure time in microseconds or raw units.

Syntax

```
[VB]  
objActiveUSB.ExposureTime [= Value]
```

```
[C/C++]  
HRESULT get_ExposureTime( float *pValue );  
HRESULT put_ExposureTime( float Value );
```

Data Type [VB]

Single

Parameters [C/C++]

pValue [out,retval]
Pointer to the current exposure value
Value [in]
The exposure value to be set

Return Values

S_OK
Success
E_NOINTERFACE
The feature is not available for the selected camera
E_INVALIDARG
The value is out of range
E_FAIL
Failure to set the feature value

Example

The following VB example demonstrates the use of a scroll control for real-time adjustment of the exposure time.

```
Private Sub Form_Load()  
HScroll1.Value = ActiveUSB1.ExposureTime  
HScroll1.Min = ActiveUSB1.GetExposureMin  
HScroll1.Max = ActiveUSB1.GetExposureMax  
ActiveUSB1.Acquire=True  
End Sub  
  
Private Sub HScroll1_Scroll()  
ActiveUSB1.ExposureTime = HScroll1.Value  
End Sub
```

Remarks

This property changes the integration time of the camera's sensor to a sub-value of the frame period. The valid range of the exposure values can be obtained by the [GetExposureTimeMin](#) and [GetExposureTimeMax](#) methods.

Note that the property is available only if the currently selected camera supports one of the following GenICam features: *ExposureTime*, *ExposureTimeAbs*, *ExposureTimeRaw*.

3.1.29 Flip

Description

Returns or sets the horizontal and vertical flipping of the image. The values from 0 to 3 correspond to the following flipping conditions:

- 0 - *None*
No image flipping is performed.
- 1 - *Horizontal*
The image is flipped horizontally.
- 2 - *Vertical*
The image is flipped vertically.
- 3 - *Both*
The image is flipped horizontally and vertically.

Syntax

```
[VB]  
objActiveUSB.Flip [= Value]
```

```
[C/C++]  
HRESULT get_Flip( long *pFlip );  
HRESULT put_Flip( long Flip );
```

Data Type [VB]

Long

Parameters [C/C++]

- pFlip* [out,retval]
Pointer to the ordinal number of the currently selected flipping condition.
- Palette* [in]
The flipping condition to be set.

Return Values

- S_OK
Success
- E_FAIL
Failure.
- E_INVALIDARG
Invalid property value.

Example

This VB example demonstrates the use of a combo box for flipping the live video:

```
Private Sub Form_Load()  
ActiveUSB1.Acquire = True  
Combo1.AddItem ("None")  
Combo1.AddItem ("Horizontal")
```

```
Combol.AddItem ("Vertical")
Combol.AddItem ("Diagonal")
Combol.ListIndex = 0
End Sub

Private Sub Combol_Click()
ActiveUSB1.Flip = Combol.ListIndex
End Sub
```

Remarks

Flipping affects the way the video is displayed in the control window as well as actual order of pixels in the image frame. If you apply one of the flipping options, the data returned by [GetImageData](#) and other data access methods will be flipped as well.

Note that the flip operation has precedence over [Rotate](#). If both **Flip** and **Rotate** properties are non-zero, the frame will be flipped first and the resulting image rotated.

3.1.30 Font

Description

Returns or sets the font for [DrawText](#).

Syntax

```
[VB]  
objActiveUSB.Font [= Font]
```

```
[C/C++]  
HRESULT get_Font(IFontDisp* *pFont);  
HRESULT put_Font(IFontDisp* Font);
```

Data Type [VB]

StdFont

Parameters [C/C++]

pFont [out,retval]
Pointer to the *IFontDisp* interface object corresponding to the current overlay font

Font [in]
IFontDisp interface object corresponding to the overlay font to be set

Return Values

S_OK
Success

E_FAIL
Failure.

Example

The following VB example inserts two string of text in the live video:

```
Private Sub ActiveUSB1_FrameAcquired()  
Dim Font1 As New StdFont  
Font1.Name = "Arial"  
Font1.Size = 30  
Font1.Bold = True  
ActiveUSB1.Font = Font1  
ActiveUSB1.DrawText 10, 100, "ActiveUSB", 255, 0, 0  
Font2.Name = "Arial"  
Font2.Size = 20  
Font2.Italic = True  
ActiveUSB1.Font = Font2  
ActiveUSB1.DrawText 10, 200, "The most powerful USB3 Vision SDK", 0, 0, 255  
End Sub
```

Remarks

Also see [DrawText](#).

3.1.31 Format

Description

Returns or sets the index of the currently selected pixel format.

Syntax

```
[VB]  
objActiveUSB.Format [= Value]
```

```
[C/C++]  
HRESULT get_Format( long *pFormat );  
HRESULT put_Format( long Format );
```

Data Type [VB]

Long

Parameters [C/C++]

pFormat [out,retval]
Pointer to the format's index
Format [in]
The index of the format to be selected

Return Values

S_OK
Success
E_FAIL
Failure.
E_INVALIDARG
The index is out of range

Example

This VB example initializes a combo box with the descriptions of available pixel formats and uses it to select a specific format:

```
Private Sub Form_Load()  
FormatLst = ActiveUSB1.GetFormatList  
For i = 0 To UBound(FormatLst)  
Combol.AddItem (FormatLst(i))  
Next  
Combol.ListIndex = 0  
ActiveUSB1.Acquire = True  
End Sub  
  
Private Sub Combol_Click()  
ActiveUSB1.Format = Combol.ListIndex  
End Sub
```

Remarks

The value of the property is a zero-based index into the list of pixel formats supported by the currently selected [Camera](#). The list of formats can be retrieved with [GetFormatList](#). If you use the property window of your development environment or *ActiveUSB*'s property pages, the **Format** property field will be presented as a list box containing all supported video formats. The format list is built in the ascending order by browsing through all pixel formats supported by the current camera.

ActiveUSB supports the following USB3 Vision™ pixel formats:

Pixel Format	Description	Bits per pixel
Mono8	8-bit monochrome unsigned	8
Mono8Signed	8-bit monochrome signed	8
Mono10	10-bit monochrome unpacked	16
Mono10Packed	10-bit monochrome packed	12
Mono12	12-bit monochrome unpacked	16
Mono12Packed	12-bit monochrome packed	12
Mono14	14-bit monochrome unpacked	14
Mono16	16-bit monochrome	16
Bayer**8	8-bit raw Bayer	8
Bayer**10	10-bit raw Bayer unpacked	16
Bayer**12	12-bit raw Bayer unpacked	16
Bayer**10	10-bit raw Bayer packed	12
Bayer**12	12-bit raw Bayer packed	12
RGB8Packed	24-bit RGB color	24
BGR8Packed	24-bit BGR color	24
RGBA8Packed	24-bit RGB color with alpha channel	32
BGRA8Packed	24-bit BGR color with alpha channel	32
RGB10Packed	30-bit RGB color	48
BGR10Packed	30-bit BGR color	48
RGB12Packed	36-bit RGB color	48
BGR12Packed	36-bit BGR color	48
BGR10V1Packed	30-bit BGR color packed	32
BGR10V2Packed	30-bit BGR color packed	32
YUV411Packed	12-bit YUV color	12
YUV422Packed	16-bit YUV color	16
YUV444Packed	24-bit YUV color	24
RGB8Planar	24-bit RGB in form of three 8-bit planes	24
RGB10Planar	30-bit BGR in form of three 16-bit planes	48
RGB12Planar	36-bit BGR in form of three 16-bit planes	48
RGB16Planar	48-bit BGR in form of three 16-bit planes	48

Note - ** in the Bayer format names stands for GR, RG, GB or BG type of [Bayer](#) layout.

Depending on your camera, some manufacturer-specific formats can also be supported by *ActiveUSB*.

For more information refer to "*USB3 Vision Camera Interface Standard For Machine Vision*" published by the Automated Imaging Association.

3.1.32 Gain

Description

Returns or sets the value of the camera gain in absolute or raw units.

Syntax

```
[VB]  
objActiveUSB.Gain [= Value]
```

```
[C/C++]  
HRESULT get_Gain( float *pValue );  
HRESULT put_Gain( float Value );
```

Data Type [VB]

Single

Parameters [C/C++]

pValue [out,retval]
Pointer to the current exposure value
Value [in]
The exposure value to be set

Return Values

S_OK
Success
E_NOINTERFACE
The feature is not available for the selected camera
E_INVALIDARG
The value is out of range
E_FAIL
Failure to set the feature value

Example

The following VB example demonstrates the use of a scroll control for real-time adjustment of the gain value.

```
Private Sub Form_Load()  
HScroll1.Value = ActiveUSB1.Gain  
HScroll1.Min = ActiveUSB1.GetGain  
HScroll1.Max = ActiveUSB1.GetGain  
ActiveUSB1.Acquire=True  
End Sub  
  
Private Sub HScroll1_Scroll()  
ActiveUSB1.Gain = HScroll1.Value  
End Sub
```

Remarks

This property changes the camera's video signal amplification. The valid range of the gain values can be obtained by the [GetGainMin](#) and [GetGainMax](#) methods.

Note that the property is available only if the currently selected camera supports one of the following GenICam features: *Gain*, *GainAbs*, *GainRaw*.

3.1.33 GainAuto

Description

Returns or sets the automatic gain control (AGC) mode. Can be one of the following values:

"Off"

Gain is manually controlled using [Gain](#)

"Once"

The camera sets the optimal gain level and returns to the "Off" state

"Continuous"

The camera constantly adjusts the gain level

Syntax

[VB]

```
objActiveUSB.GainAuto [= Value]
```

[C/C++]

```
HRESULT get_GainAuto( string *pValue );  
HRESULT put_GainAuto( string Value );
```

Data Type [VB]

String

Parameters [C/C++]

pValue [out, retval]

Pointer to the string specifying the gain control mode

Value [in]

The gain control mode to be set

Return Values

S_OK

Success

E_NOINTERFACE

The feature is not available for the selected camera

E_INVALIDARG

The value is not part of the enumerated set

E_FAIL

Failure to set the feature value

Example

The following VB example demonstrates the use of a check box to switch between the manual and automatic gain control.

```
Private Sub Check1_Click()  
If Check1.Value = 1 Then  
ActiveUSB1.GainAuto = "On"  
Else  
ActiveUSB1.GainAuto = "Off"
```

```
End If  
End Sub
```

Remarks

Depending on a camera model, other device-specific values may be available for this feature. The list of valid values can be retrieved using [GetEnumList](#).

Note that the property is available only if the currently selected camera supports the *GainAuto* feature per GenICam standard.

3.1.34 GainSelector

Description

Returns or sets the channel to be controlled by gain adjustments. Can be one of the following values:

- "All"
Gain adjustments will be applied to all channels.
- "Red"
Gain adjustments will be applied to the red channel.
- "Green"
Gain adjustments will be applied to the green channel.
- "Blue"
Gain adjustments will be applied to the blue channel.
- "Y"
Gain adjustments will be applied to Y channel.
- "U"
Gain adjustments will be applied to U channel.
- "V"
Gain adjustments will be applied to V channel.

Syntax

[VB]
`objActiveUSB.GainSelector [= Value]`

[C/C++]
`HRESULT get_GainSelector(string *pValue);`
`HRESULT put_GainSelector(string Value);`

Data Type [VB]

String

Parameters [C/C++]

- pValue* [out, retval]
Pointer to the string specifying the gain selector setting
- Value* [in]
The gain selection to be set

Return Values

- S_OK
Success
- E_NOINTERFACE
The feature is not available for the selected camera
- E_INVALIDARG
The value is not part of the enumerated set
- E_FAIL
Failure to set the feature value

Example

The following VB example demonstrates the use of a combo box to switch between different gain channels:

```
Private Sub Form_Load()  
Lst = ActiveUSB1.GetEnumList("GainSelector")  
For i = 0 To UBound(Lst)  
  Combol.AddItem (Lst(i))  
Next  
Combol.ListIndex = ActiveUSB1.GainSelector  
End Sub  
  
Private Sub Combol_Click()  
ActiveUSB1.GainSelector = Combol.Text  
End Sub
```

Remarks

Depending on a camera model, other device-specific values may be available for this feature. The list of valid values can be retrieved using [GetEnumList](#).

Note that the property is available only if the currently selected camera supports the *GainSelector* feature per GenICam standard.

3.1.35 Gamma

Description

Returns or sets the gamma correction factor for the image.

Syntax

[VB]
`objActiveUSB.Gamma [= Value]`

[C/C++]
`HRESULT get_Gamma(float *pValue);`
`HRESULT put_Gamma(float Value);`

Data Type [VB]

Single

Parameters [C/C++]

pValue [out,retval]
Pointer to the current gamma factor
Value [in]
The gamma factor to be set

Return Values

S_OK
Success
E_NOINTERFACE
The feature is not available for the selected camera
E_INVALIDARG
The value is out of range
E_FAIL
Failure to set the feature value

Example

The following VB example sets the Gamma factor to 1.2

```
ActiveUSB1.Gamma = 1.2
```

Remarks

This property changes the gamma correction factor of the camera's circuitry. The gamma correction modifies an image by applying standard, nonlinear gamma curves to the intensity scale. Increasing the gamma value will lighten the video and increase the contrast in its darker areas. The valid range of the gamma values can be obtained by the [GetFeatureMin](#) and [GetFeatureMax](#) methods.

Note that the property is available only if the currently selected camera supports the *Gamma* feature.

3.1.36 HotPixelCorrect

Description

Enables/disables the hot pixel correction mode. Used in combination with [HotPixelLevel](#).

Syntax

[VB]

```
objActiveUSB.Integrate [= Value]
```

[C/C++]

```
HRESULT get_HotPixelCorrect ( bool *pCorrect );  
HRESULT put_HotPixelCorrect( bool Correct );
```

Data Type [VB]

Boolean

Parameters [C/C++]

pCorrect [out,retval]

Pointer to the Boolean that is TRUE if the hot pixel correction is enabled, or FALSE otherwise

Correct [in]

Set to TRUE to enable the hot pixel correction, or set to FALSE to disable it

Return Values

S_OK

Success

E_FAIL

Failure.

Example

This VB example activates the hot pixel correction mode with a 15% threshold:

```
ActiveUSB1.HotPixelLevel = 15  
ActiveUSB1.HotPixelCorrect = True
```

Remarks

Hot pixels are individual pixels that appear much brighter than the rest of an image. They are associated with elements on a camera sensor that have higher than normal rates of charge leakage. Hot pixels are especially noticeable in a low-light situation when the shutter and/or gain are set to high values.

The hot pixel correction algorithm is based on the "top hat" technique that treats a pixel intensity as an elevation of a surface. Any pixel that protrudes through the "crown of the hat" which height is defined by a value of [HotPixelLevel](#) is considered to be noise and replaced by the mean value under the "brim of the hat". This effectively removes hot pixels from the image.

3.1.37 HotPixelLevel

Description

Returns or sets the threshold level to be used in the [hot pixel correction](#) mode, in percents.

Syntax

[VB]
`objActiveUSB.HotPixelLevel [= Value]`

[C/C++]
`HRESULT get_HotPixelLevel(long *pValue);`
`HRESULT put_HotPixelLevel(long Value);`

Data Type [VB]

Long

Parameters [C/C++]

pValue [out,retval]
Pointer to the currently selected threshold level for the hot pixel correction mode.
Value [in]
The threshold level to be set.

Return Values

S_OK
Success
E_FAIL
Failure.
E_INVALIDARG
Invalid property value.

Example

This VB example activates the hot pixel correction mode with a 15% threshold:

```
ActiveUSB1.HotPixelLevel = 15  
ActiveUSB1.HotPixelCorrect = True
```

Remarks

The hot pixel level indicates the minimum difference in the intensity between a pixel and its neighborhood in order for the pixel to be considered hot. The value of the hot pixel level is given in percents of the maximum intensity for the given type of image. If the pixel falls into a hot-pixel category, its value will be replaced by the average intensity of the adjacent pixels.

3.1.38 Integrate

Description

Enables/disables the frame integration operation and selects the integration mode. The frame integration allows you to average or add frames "on the fly" without sacrificing the frame rate. Used in combination with

Syntax

[VB]
`objActiveUSB.Integrate [= Value]`

[C/C++]
`HRESULT get_Integrate (short *pIntegrate);`
`HRESULT put_Integrate(short Integrate);`

Data Type [VB]

Integer

Parameters [C/C++]

pIntegrate [out,retval]

Pointer to the ordinal number of the currently selected Integrate filter, zero if Integrate conversion is disabled.

Integrate [in]

0 - Frame integration is disabled.

1 - Running Average mode. Each output frame is the result of averaging a selected number of previously captured frames.

2 - Running Accumulation mode. Each output frame is the sum of a selected number of previously captured frame.

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example activates the running average mode with a 16-frame window:

```
ActiveUSB1.IntegrateWnd=16  
ActiveUSB1.Integrate = 1
```

Remarks

The frame integration is especially useful for suppressing the noise in the video and increasing its contrast in a low-light situation.

3.1.39 IntegrateWnd

Description

Returns or sets the number of frames to be used in the [Integrate](#) operation.

Syntax

[VB]
`objActiveUSB.IntegrateWnd [= Value]`

[C/C++]
`HRESULT get_IntegrateWnd(long *pValue);`
`HRESULT put_IntegrateWnd(long Value);`

Data Type [VB]

Long

Parameters [C/C++]

pValue [out,retval]
Pointer to the currently selected number of frames for the frame integration
Value [in]
The number of frames to be set

Return Values

S_OK
Success
E_FAIL
Failure.
E_INVALIDARG
Invalid property value.

Example

This VB example activates the running average mode with a 16-frame window:

```
ActiveUSB1.IntegrateWnd=16  
ActiveUSB1.Integrate = 1
```

Remarks

The size of the integration windows indicates the number of consecutive frames used for the Running Average or Running Accumulation operations. Increasing the size of the integration window lowers the video noise, but increases a motion-related blurring.

3.1.40 LensCorrect

Description

Enables/disables the lens distortion correction. Used in combination with [SetLensDistortion](#).

Syntax

[VB]
`objActiveUSB.LensCorrect [= Value]`

[C/C++]
`HRESULT get_LensCorrect (bool *pCorrect);`
`HRESULT put_LensCorrect(bool Correct);`

Data Type [VB]

Boolean

Parameters [C/C++]

pCorrect [out,retval]
Pointer to the Boolean that is TRUE if the lens distortion correction is enabled, or FALSE otherwise
Correct [in]
Set to TRUE to enable the lens distortion correction, or set to FALSE to disable it

Return Values

S_OK
Success
E_FAIL
Failure.

Example

The following VB examples activates the lens distortion correction and uses two scroll bars to adjust the distortion parameters in real time :

```
Private Sub Form_Load()  
ActiveUSB1.Acquire=True  
HScroll1.Min = -1000  
HScroll1.Max = 1000  
HScroll2.Min = -1000  
HScroll2.Max = 1000  
ActiveUSB1.SetLensDistortion 0,0,False  
ActiveUSB1.LensCorrect=True  
ActiveUSB1.Acquire=True  
End Sub  
  
Private Sub HScroll1_Scroll()  
ActiveUSB1.SetLensDistortion HScroll1.Value/1000., HScroll2.Value/1000.,  
False  
End Sub  
  
Private Sub HScroll2_Scroll()
```

```
ActiveUSB1.SetLensDistortion HScroll1.Value/1000., HScroll2.Value/1000.,  
False  
End Sub
```

Remarks

The lens distortion correction is used to compensate for the barrel or pincushion image distortion caused by camera lenses. The barrel distortion makes straight lines at the edges of the image bow outwards and it is commonly seen on wide angle lenses with short focal length. The pincushion distortion makes straight lines at the edges of the image bow inwards, and it is commonly seen on telephoto lenses with long focal length. Before activating the lens distortion correction, you must set up the distortion parameters by calling [SetLensDistortion](#).

3.1.41 LineFormat

Description

Returns or sets (if possible) the current electrical format of the selected I/O line. Can be one of the following values:

"NoConnect"

The line is not connected.

"TriState"

The line is currently in the Tri-State mode (not driver).

"TTL"

The line is currently accepting or sending TTL level signals.

"LVDS"

The line is currently accepting or sending LVDS level signals.

"RS422"

The line is currently accepting or sending RS422 level signals..

"OptoCoupled",...

The line is opto-coupled.

Syntax

[VB]

```
objActiveUSB.LineFormat [= Value]
```

[C/C++]

```
HRESULT get_LineFormat( string *pValue );
```

```
HRESULT put_LineFormat( string Value );
```

Data Type [VB]

String

Parameters [C/C++]

pValue [out, retval]

Pointer to the string specifying the line format.

Value [in]

The line format to be set.

Return Values

S_OK

Success

E_NOINTERFACE

The feature is not available for the selected camera

E_INVALIDARG

The value is not part of the enumerated set

E_FAIL

Failure to set the feature value

Example

The following VB example demonstrates the use of a combo box for line format selection:

```
Private Sub Form_Load()  
Lst = ActiveUSB1.GetEnumList("LineFormat")  
For i = 0 To UBound(Lst)  
Combol.AddItem (Lst(i))  
Next  
Combol.ListIndex = ActiveUSB1.LineFormat  
End Sub  
  
Private Sub Combol_Click()  
ActiveUSB1.LineFormat = Combol.Text  
End Sub
```

Remarks

Depending on a camera model, other device-specific values may be available for this feature. The list of valid values can be retrieved using [GetEnumList](#).

Note that the property is available only if the currently selected camera supports the *LineFormat* feature per GenICam standard.

3.1.42 LineInverter

Description

Returns or sets the inverted state of the electrical input or output signal on the selected I/O line.

Syntax

```
[VB]  
objActiveUSB.LineInverter [= Value]
```

```
[C/C++]  
HRESULT get_LineInverter( VARIANT_BOOL *pValue );  
HRESULT put_LineInverter( VARIANT_BOOL Value );
```

Data Type [VB]

Boolean

Parameters [C/C++]

pValue [out, retval]
Pointer to the boolean value specifying the state of the line.
Value [in]
FALSE if the line signal is not inverted, TRUE if the line signal is inverted.

Return Values

S_OK
Success
E_NOINTERFACE
The feature is not available for the selected camera
E_INVALIDARG
The value is not part of the enumerated set
E_FAIL
Failure to set the feature value

Example

The following VB example sets an output of an inverted pulse coming from the Timer1 on the physical line 2 of the camera connector :

```
ActiveUSB1.LineSelector="Line2"  
ActiveUSB1.LineMode="Output"  
ActiveUSB1.LineInverter="True"  
ActiveUSB1.LineSource="Timer1Ouput"
```

Remarks

Depending on a camera model, other device-specific values may be available for this feature. The list of valid values can be retrieved using [GetEnumList](#).

Note that the property is available only if the currently selected camera supports the *LineInverter*

feature per GenICam standard.

3.1.43 LineMode

Description

Returns or sets the input or output mode for the currently selected I/O line. Can be one of the following values:

"Input"

The selected physical line is used to input an electrical signal.

"Output"

The selected physical line is used to output an electrical signal.

Syntax

[VB]

```
objActiveUSB.LineMode [= Value]
```

[C/C++]

```
HRESULT get_LineMode( string *pValue );  
HRESULT put_LineMode( string Value );
```

Data Type [VB]

String

Parameters [C/C++]

pValue [out, retval]

Pointer to the string specifying the line mode.

Value [in]

The line mode to be set.

Return Values

S_OK

Success

E_NOINTERFACE

The feature is not available for the selected camera

E_INVALIDARG

The value is not part of the enumerated set

E_FAIL

Failure to set the feature value

Example

The following VB example demonstrates the use of a combo box for line mode selection:

```
Private Sub Form_Load()  
Lst = ActiveUSB1.GetEnumList("LineMode")  
For i = 0 To UBound(Lst)  
Combo1.AddItem (Lst(i))  
Next  
Combo1.ListIndex = ActiveUSB1.LineMode
```

```
End Sub

Private Sub Comb1_Click()
ActiveUSB1.LineMode = Comb1.Text
End Sub
```

Remarks

When a line supports input and output modes, the default state is "Input" to avoid possible electrical contention.

Depending on a camera model, other device-specific values may be available for this feature. The list of valid values can be retrieved using [GetEnumList](#).

Note that the property is available only if the currently selected camera supports the *LineMode* feature per GenICam standard.

3.1.44 LineSelector

Description

Returns or sets the physical line (or pin) of the external device connector to configure. The values are strings specifying the index of the physical line and associated I/O control block, such as "Line0", "Line1", "Line2".

Syntax

[VB]
`objActiveUSB.LineSelector [= Value]`

[C/C++]
`HRESULT get_LineSelector(string *pValue);`
`HRESULT put_LineSelector(string Value);`

Data Type [VB]

String

Parameters [C/C++]

pValue [out, retval]
Pointer to the string specifying the physical line to configure.
Value [in]
The line selection to be set

Return Values

S_OK
Success
E_NOINTERFACE
The feature is not available for the selected camera
E_INVALIDARG
The value is not part of the enumerated set
E_FAIL
Failure to set the feature value

Example

The following VB example sets an output of an inverted pulse coming from the Timer1 on the physical line 2 of the camera connector :

```
ActiveUSB1.LineSelector="Line2"  
ActiveUSB1.LineMode="Output"  
ActiveUSB1.LineFormat="TTL"  
ActiveUSB1.LineInverter="True"  
ActiveUSB1.LineSource="Timer1Output"
```

Remarks

When a specific line is selected, all the other line features ([LineMode](#), [LineFormat](#), [LineSource](#),

[LineInverter](#)) will be applied to its associated I/O control block and will condition the resulting input or output signal.

Depending on a camera model, other device-specific values may be available for this feature. The list of valid values can be retrieved using [GetEnumList](#).

Note that the property is available only if the currently selected camera supports the *LineSelector* feature per GenICam standard.

3.1.45 LineSource

Description

Returns or sets the source of the signal to output on the selected I/O line when the line mode is Output. Can be one of the following values:

- "Off"
Line output is disabled (Tri-State).
- "AcquisitionTriggerWait"
Camera is currently waiting for a trigger to capture one frame or series of frames.
- "AcquisitionActive"
Camera is currently doing an acquisition of one frame or series of frames.
- "FrameTriggerWait"
Camera is currently waiting for a frame trigger.
- "FrameActive"
Camera is currently doing a capture of a frame.
- "Timer1Active", "Timer2Active",...
The chosen timer is in the active state.
- "Counter1Active", "Counter2Active",...
The chosen counter is in the active state.
- "UserOutput1Active", "UserOutput2Active",...
The chosen user output bit state as defined by its current [UserOutputValue](#).

Syntax

[VB]
`objActiveUSB.LineSource [= Value]`

[C/C++]
`HRESULT get_LineSource(string *pValue);`
`HRESULT put_LineSource(string Value);`

Data Type [VB]

String

Parameters [C/C++]

- pValue* [out, retval]
Pointer to the string specifying the line source.
- Value* [in]
The line source to be set

Return Values

- S_OK
Success
- E_NOINTERFACE
The feature is not available for the selected camera
- E_INVALIDARG
The value is not part of the enumerated set
- E_FAIL
Failure to set the feature value

Example

The following VB example demonstrates the use of a combo box for line source selection:

```
Private Sub Form_Load()  
Lst = ActiveUSB1.GetEnumList("LineSource")  
For i = 0 To UBound(Lst)  
  Combol.AddItem (Lst(i))  
Next  
Combol.ListIndex = ActiveUSB1.LineSource  
End Sub  
  
Private Sub Combol_Click()  
ActiveUSB1.LineSource = Combol.Text  
End Sub
```

Remarks

Depending on a camera model, other device-specific values may be available for this feature. The list of valid values can be retrieved using [GetEnumList](#).

Note that the property is available only if the currently selected camera supports the *LineSource* feature per GenICam standard.

3.1.46 LUTMode

Description

Enables/disables the lookup table operation on the video frames. The operation transforms the value of each pixel based on a corresponding factor in the LUT array.

Syntax

[VB]
`objActiveUSB.LUTMode [= Value]`

[C/C++]
`HRESULT get_LUTMode (bool *pLUT);`
`HRESULT put_LUTMode(bool LUT);`

Data Type [VB]

Boolean

Parameters [C/C++]

pLUT [out,retval]
Pointer to the Boolean that is TRUE if the lookup table operation is enable, or FALSE otherwise
LUT [in]
Set to TRUE to enable the lookup table operation, or set to FALSE to disable it.

Return Values

S_OK
Success
E_FAIL
Failure.

Example

The following VB example enables the Window/Level operation and sets the minimum and maximum levels for the histogram scaling to 20% and 70% :

```
ActiveUSB1.LUTMode = True  
ActiveUSB1.SetLevels 20, 70
```

Remarks

This property works in combination with the [SetLUT](#), [SetLevels](#) or [SetGains](#) methods.

3.1.47 Magnification

Description

Returns or sets the zoom factor for the live video display.

Syntax

```
[VB]  
objActiveUSB.Magnification [= Value]
```

```
[C/C++]  
HRESULT get_Magnification( float *pMagnification );  
HRESULT put_Magnification( float Magnification );
```

Data Type [VB]

Float

Parameters [C/C++]

pMagnification [out,retval]

Pointer to the current zoom factor

Magnification [in]

Floating point value specifying the zoom factor to be set. If 0, the image will be fit to the size of the control window. If -1, the image will be displayed in the full-screen mode. If -2, the image conversion and display will be disabled.

Return Values

S_OK

Success

E_FAIL

Failure.

E_INVALIDARG

Invalid property value.

Example

This VB example sets the zoom factor to 0.25:

```
ActiveUSB1.Magnification = 0.25  
MsgBox ActiveUSB1.Magnification
```

Remarks

This property adjusts the magnification of the live video display. It does not change the content of the image data, but only its appearance in the *ActiveUSB* control window. The valid property values are from 0 to 100. If the **Magnification** property is set to zero, the video image will be fit to the size of the control window. In this case the display might not retain the original proportions of the video frame. If the property is set to -1, the image will be displayed in the full-screen mode retaining the original proportions.

Note that the [Acquire](#) and [Display](#) properties must be set to TRUE in order for the live video to be

displayed in the control window.

Note that setting **Magnification** properties to -2 will disable an image conversion pipeline that is normally applied to each raw frame. This will significantly increase the performance of your application as long as it does not utilize *ActiveUSB*'s built-in image decoding/display and uses [GetRawData](#) to access the raw image data.

3.1.48 MonitorSync

Description

Enables/disables the monitor synchronization feature for the live video display in the control window.

Syntax

```
[VB]  
objActiveUSB.MonitorSync [= Value]
```

```
[C/C++]  
HRESULT get_MonitorSync ( bool *pValue );  
HRESULT put_MonitorSync( bool Value );
```

Data Type [VB]

Boolean

Parameters [C/C++]

pValue [out,retval]
Pointer to the Boolean that is TRUE if the monitor synchronization is enable, or FALSE otherwise

Value [in]
Set to TRUE to enable the monitor synchronization, or set to FALSE to disable it

Return Values

S_OK
Success

E_FAIL
Failure.

Example

This VB example enables the monitor synchronization:

```
ActiveUSB1.MonitorSync = True  
MsgBox ActiveUSB1.MonitorSync
```

Remarks

Enabling the monitor synchronization option causes the camera frame rate to exactly match the current refresh rate of the system monitor thus eliminating all the display artifacts related to the digital image transfer, such as the display tearing and syncopation. The monitor synchronization algorithm uses a patent-pending technique from A&B Software. For more details refer to [Software overcomes display artifacts in digital image transfer](#).

Note that this property will be unavailable if the camera does not support the software trigger.

3.1.49 Overlay

Description

Enables/disables the overlay.

Syntax

```
[VB]  
objActiveUSB.Overlay [= Value]
```

```
[C/C++]  
HRESULT get_Overlay ( bool *pOverlay );  
HRESULT put_Overlay( bool Overlay );
```

Data Type [VB]

Boolean

Parameters [C/C++]

pOverlay [out,retval]
Pointer to the Boolean that is TRUE if the overlay is enable, or FALSE otherwise

Overlay [in]
Set to TRUE to enable the overlay, or set to FALSE to disable it.

Return Values

S_OK
Success

E_FAIL
Failure.

Example

The following VB example overlays a red circle on the live video:

```
ActiveUSB1.Acquire = True  
ActiveUSB1.OverlayEllipse 100,100,200,200  
ActiveUSB1.OverlayColor=RGB(255,0,0)  
ActiveUSB1.Overlay= True
```

Remarks

Overlay feature allows you to display custom graphics and text on the live video image. Setting this property to true causes ActiveUSB to show the overlay. Disabling this property hides the overlay. Note that hiding the overlay does not erase graphics and text from it. To clear the overlay, use [OverlayClear](#). Also see [OverlayColor](#), [OverlayPixel](#), [OverlayLine](#), [OverlayRectangle](#), [OverlayEllipse](#), [OverlayText](#).

3.1.50 OverlayColor

Description

Returns or sets the color of the overlay graphics.

Syntax

[VB]
`objActiveUSB.OverlayColor [= Color]`

[C/C++]
`HRESULT get_OverlayColor(OLE_COLOR& pColor);`
`HRESULT put_OverlayColor(OLE_COLOR Color);`

Data Type [VB]

RGB color

Parameters [C/C++]

pColor [out,retval]
Pointer to the current overlay color
Color [in]
The overlay color to be set

Return Values

S_OK
Success
E_FAIL
Failure.

Example

The following VB example overlays a red circle on the live video:

```
ActiveUSB1.Acquire = True
ActiveUSB1.OverlayEllipse 100,100,200,200
ActiveUSB1.OverlayColor=RGB(255,0,0)
ActiveUSB1.Overlay= True
```

Remarks

Overlay feature allows you to display custom graphics and text on the live video image. Also see [OverlayColor](#), [OverlayPixel](#), [OverlayLine](#), [OverlayRectangle](#), [OverlayEllipse](#), [OverlayText](#).

3.1.51 OverlayFont

Description

Returns or sets the font for [OverlayText](#).

Syntax

```
[VB]  
objActiveUSB.OverlayFont [= Font]
```

```
[C/C++]  
HRESULT get_OverlayFont(IFontDisp* *pFont);  
HRESULT put_OverlayFont(IFontDisp* Font);
```

Data Type [VB]

StdFont

Parameters [C/C++]

pFont [out,retval]
Pointer to the *IFontDisp* interface object corresponding to the current overlay font

Font [in]
IFontDisp interface object corresponding to the overlay font to be set

Return Values

S_OK
Success

E_FAIL
Failure.

Example

The following VB example overlays a string of text on the live video:

```
Dim Font As New StdFont  
Font.Name = "Arial"  
Font.Size = 18  
Font.Bold = True  
ActiveUSB1.OverlayFont = Font  
ActiveUSB1.OverlayText 10, 100, "ActiveUSB rules!"  
ActiveUSB1.OverlayColor = RGB(255, 0, 0)  
ActiveUSB1.Overlay = True
```

Remarks

Also see [OverlayColor](#), [OverlayText](#).

3.1.52 Palette

Description

Returns or sets the ordinal number of the currently selected live video palette. The values from 0 to 7 correspond to the following predefined palettes:

0 - Gray

Applies the standard 256-level grayscale palette. This is a regular mode of viewing a grayscale video.

1 - Inverse

Applies the inverted 256-level grayscale palette. The video will be displayed in the negative format.

2 - Saturated

Applies the grayscale palette with colored upper entries. The saturated palette allows you to control the dynamic range of the video signal by bringing it slightly below the saturation level of the video camera or video amplifier. To achieve the maximum dynamic range, adjust the intensity of the light source and/or the gain and zero level of the video amplifier so that the red color corresponding to the brightest pixel values just barely shows up.

3 - Rainbow

Applies a color palette where the entries are evenly distributed along the Hue axis. This allows for assigning different color pigments to different levels of intensity.

4 - Spectra

Applies a color palette where the entries are distributed along the Hue and Luminance axes. That allows for assigning different color pigments to different levels of intensity while preserving the luminance scale.

5 - Isodense

Applies the 256-level grayscale palette, each 8-th entry of which is colored. The isodense palette allows you to clearly see transitions between different levels of intensities as isolines on a topographic map.

6 - Multiphase

Applies the multiphase palette. Entries in the multiphase palette are at opposite ends of the color model so even small changes in gray levels are highlighted.

7 - Random

Applies the random color palette whose entries are filled with random values each time you select it from the menu.

8 - Threshold

Applies the grayscale palette with colored luminance range of interest. Works in combination with [SetROI](#).

Syntax

[VB]

```
objActiveUSB.Palette [= Value]
```

[C/C++]

```
HRESULT get_Palette( long *pPalette );  
HRESULT put_Palette( long Palette );
```

Data Type [VB]

Long

Parameters [C/C++]

pPalette [out,retval]
Pointer to the ordinal number of the currently selected palette
Palette [in]
The number of the palette to be selected

Return Values

S_OK
Success
E_FAIL
Failure.
E_INVALIDARG
Invalid property value.

Example

This VB example demonstrates the use of a combo box for changing display palettes on the live video:

```
Private Sub Form_Load()  
ActiveUSB1.Acquire = True  
Combol.AddItem ("Gray")  
Combol.AddItem ("Inverse")  
Combol.AddItem ("Saturated")  
Combol.AddItem ("Rainbow")  
Combol.AddItem ("Spectra")  
Combol.AddItem ("Isodense")  
Combol.AddItem ("Multiphase")  
Combol.AddItem ("Random")  
Combol.ListIndex = 0  
End Sub  
  
Private Sub Combol_Click()  
ActiveUSB1.Palette = Combol.ListIndex  
End Sub
```

Remarks

The **Palette** property is used for viewing monochrome video in pseudo-colors. The property is only valid for a monochrome [Mode](#). If you use property pages in your development environment, the **Palette** property field will be presented as a list box containing the names of all available palettes.

Note that the [Acquire](#) and [Display](#) properties must be set to TRUE in order for the live video to be displayed in the control window.

3.1.53 Rotate

Description

Rotates the image at the specified angle. The following angle values are allowed:

- 0 - No image rotation is performed
- 90 - The image is rotated 90° counterclockwise
- 180 - The image is rotated 180°
- 270 - The image is rotated -90° clockwise

Syntax

[VB]

```
objActiveUSB.Rotate [= Value]
```

[C/C++]

```
HRESULT get_Rotate( long *pRotate );  
HRESULT put_Rotate( long Rotate );
```

Data Type [VB]

Long

Parameters [C/C++]

pRotate [out,retval]

Pointer to the rotational angle.

Rotate [in]

The rotational angle to be set.

Return Values

S_OK

Success

E_FAIL

Failure.

Example

This VB example demonstrates the use of a combo box for rotating the live video:

```
Private Sub Form_Load()  
ActiveUSB1.Acquire = True  
Combo1.AddItem ("0°")  
Combo1.AddItem ("90°")  
Combo1.AddItem ("180°")  
Combo1.AddItem ("270°")  
Combo1.ListIndex = 0  
End Sub  
  
Private Sub Combo1_Click()  
ActiveUSB1.Rotate = Combo1.ListIndex  
End Sub
```

Remarks

Image rotation affects the way the video is displayed in the control window as well as actual order of pixels in the image frame. If the rotation angle is different from zero, the data returned by [GetImageData](#) and other data access methods will be rotated as well.

Note that the [Flip](#) operation has precedence over rotation. If both **Flip** and **Rotate** properties are non-zero, the frame will be flipped first and the resulting image rotated.

If the value of the angle is different from 0, 90, 180 and 270, it will be substituted with the nearest allowable value.

3.1.54 ScrollBars

Description

Enables/disables the scroll bars on the control window.

Syntax

[VB]
`objActiveUSB.ScrollBars [= Value]`

[C/C++]
`HRESULT get_ScrollBars (bool *pScrollBars);`
`HRESULT put_ScrollBars(bool ScrollBars);`

Data Type [VB]

Boolean

Parameters [C/C++]

pScrollBars [out,retval]
Pointer to the Boolean that is TRUE if the scroll bars are enable, or FALSE otherwise
ScrollBars [in]
Set to TRUE to enable the scroll bars, or set to FALSE otherwise

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example enables scroll bars on the control window:

```
ActiveUSB1.ScrollBars = True  
MsgBox ActiveUSB1.ScrollBars
```

Remarks

If this property is set to TRUE and the video width or/and height (see [SizeX](#) and [SizeY](#)) exceed the size of the control window, a scroll bar(s) will be displayed on the border of the control window allowing you to pan the live video. The current position of scroll bars can be retrieved or modified via the [ScrollX](#) and [ScrollY](#) properties. When the scroll bars are moved, the [Scroll](#) event will be raised.

Note that the [Acquire](#) and [Display](#) properties must be set to TRUE in order for the live video to be displayed in the control window.

3.1.55 ScrollX

Description

Returns or sets the current horizontal scroll position for the live video display.

Syntax

[VB]
`objActiveUSB.ScrollX [= Value]`

[C/C++]
`HRESULT get_ScrollX(long *pScrollX);`
`HRESULT put_ScrollX(long ScrollX);`

Data Type [VB]

Long

Parameters [C/C++]

pScrollX [out,retval]
Pointer to the current horizontal scroll position.
ScrollX [in]
The horizontal scroll position to be set.

Return Values

S_OK
Success
E_FAIL
Failure.
E_INVALIDARG
Invalid property value.

Example

This VB example sets the current horizontal scroll position to 512:

```
ActiveUSB1.ScrollX = 512  
MsgBox ActiveUSB1.ScrollX
```

Remarks

The **ScrollX** property is only valid if the [ScrollBars](#) are present in the control window.

Note that the value returned by this property refers to the image coordinate system, not to the screen coordinates.

3.1.56 ScrollY

Description

Returns or sets the current vertical scroll position for the live video display.

Syntax

[VB]
`objActiveUSB.ScrollY [= Value]`

[C/C++]
`HRESULT get_ScrollY(long *pScrollY);`
`HRESULT put_ScrollY(long ScrollY);`

Data Type [VB]

Long

Parameters [C/C++]

pScrollY [out,retval]
Pointer to the current vertical scroll position.
ScrollY [in]
The vertical scroll position to be set.

Return Values

S_OK
Success
E_FAIL
Failure.
E_INVALIDARG
Invalid property value.

Example

This VB example sets the current vertical scroll position to 512:

```
ActiveUSB1.ScrollY = 512  
MsgBox ActiveUSB1.ScrollY
```

Remarks

The **ScrollY** property is only valid if the [ScrollBars](#) are present in the control window.

Note that the value returned by this property refers to the image coordinate system, not to the screen coordinates.

3.1.57 SizeX

Description

Returns or sets the width of the partial scan window.

Syntax

[VB]
`objActiveUSB.SizeX [= Value]`

[C/C++]
`HRESULT get_SizeX(long *pSizeX);`
`HRESULT put_SizeX(long SizeX);`

Data Type [VB]

Long

Parameters [C/C++]

pSizeX [out,retval]
Pointer to the currently selected image width
SizeX [in]
The image width to be selected

Return Values

S_OK
Success
E_FAIL
Failure.
E_INVALIDARG
Invalid property value.

Example

This VB example sets the current image width to 512:

```
ActiveUSB1.SizeX = 512  
MsgBox ActiveUSB1.SizeX
```

Remarks

This property allows you to set the horizontal size of the scan window in pixels. Only certain values of the width can be allowed for the partial scan window depending on the camera. If **SizeX** is set to a value which is not supported by the camera, it will be reset to the nearest allowable width. This property is directly associated with the *Width* feature of the camera.

3.1.58 SizeY

Description

Returns or sets the height of the partial scan window.

Syntax

```
[VB]  
objActiveUSB.SizeY [= Value]
```

```
[C/C++]  
HRESULT get_SizeY( long *pSizeY );  
HRESULT put_SizeY( long SizeY );
```

Data Type [VB]

Long

Parameters [C/C++]

pSizeY [out,retval]
Pointer to the currently selected image height
SizeY [in]
The image height to be selected

Return Values

S_OK
Success
E_FAIL
Failure.
E_INVALIDARG
Invalid property value.

Example

This VB example sets the current image height to 512:

```
ActiveUSB1.SizeY = 512  
MsgBox ActiveUSB1.SizeY
```

Remarks

This property allows you to set the vertical size of the scan window in pixels. Only certain values of the height can be allowed for the partial scan window depending on the camera. If the **SizeY** is set to a value which is not supported by the camera, it will be reset to the nearest allowable height. This property is directly associated with the *Height* feature of the camera.

3.1.59 OffsetX

Description

Returns or sets the horizontal offset of the partial scan window.

Syntax

[VB]
`objActiveUSB.OffsetX [= Value]`

[C/C++]
`HRESULT get_OffsetX(long *pOffsetX);`
`HRESULT put_OffsetX(long OffsetX);`

Data Type [VB]

Long

Parameters [C/C++]

pOffsetX [out,retval]
Pointer to the currently set horizontal offset
SizeX [in]
The horizontal offset to be set

Return Values

S_OK
Success
E_FAIL
Failure.
E_INVALIDARG
Invalid property value.

Example

This VB example sets the horizontal offset of the video window to 64:

```
ActiveUSB1.OffsetX = 64  
MsgBox ActiveUSB1.OffsetX
```

Remarks

This property allows you to set the horizontal offset of the top left corner of the scan window in pixels. Only certain values **OffsetX** can be allowed depending on the camera. If the property is set to a value which is not supported by the camera, it will be reset to the nearest allowable one.

3.1.60 OffsetY

Description

Returns or sets the vertical offset of the partial scan window.

Syntax

[VB]
`objActiveUSB.OffsetY [= Value]`

[C/C++]
`HRESULT get_OffsetY(long *pOffsetY);`
`HRESULT put_OffsetY(long OffsetY);`

Data Type [VB]

Long

Parameters [C/C++]

pOffsetY [out,retval]
Pointer to the currently set vertical offset
OffsetY [in]
The vertical offset to be set

Return Values

S_OK
Success
E_FAIL
Failure.
E_INVALIDARG
Invalid property value.

Example

This VB example sets the vertical offset of the video window to 64 pixels:

```
ActiveUSB1.OffsetY = 64  
MsgBox ActiveUSB1.OffsetY
```

Remarks

This property allows you to set the vertical offset of the top left corner of the scan window in pixels. Only certain values **OffsetY** can be allowed depending on the camera. If the property is set to a value which is not supported by the camera, it will be reset to the nearest allowable one.

3.1.61 Timeout

Description

Returns or sets the current acquisition timeout in seconds.

Syntax

[VB]
`objActiveUSB.Timeout [= Value]`

[C/C++]
`HRESULT get_Timeout(long *pTimeout);`
`HRESULT put_Timeout(long Timeout);`

Data Type [VB]

Long

Parameters [C/C++]

pTimeout [out,retval]
Pointer to the currently set timeout.
Timeout [in]
The timeout to be set.

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example sets the current timeout to 10 sec:

```
ActiveUSB1.Timeout = 10  
MsgBox ActiveUSB1.Timeout
```

Remarks

The **Timeout** property lets you set the number of seconds to wait for a frame to be acquired. Typically used to assign the timeout when the [Trigger](#) mode is active. If the timeout expires, the [Timeout](#) event will be raised.

3.1.62 TestImageSelector

Description

Returns or sets the mode of generating internal test images. Can be one of the following values:

- "Off"
Test image mode is disabled. Image is coming from the sensor.
- "Black"
Generates the darkest possible image
- "White"
Generates the brightest possible image
- "GrayHorizontalRamp"
Generates the horizontal wedge going from the darkest to the brightest possible intensity
- "GrayVerticalRamp"
Generates the vertical wedge going from the darkest to the brightest possible intensity
- "GrayHorizontalRampMoving"
Generates the horizontal wedge going from the darkest to the brightest possible intensity and moving from left to right
- "GrayVerticalRampMoving"
Generates the vertical wedge going from the darkest to the brightest possible intensity and moving from top to bottom
- "HorizontalLineMoving"
A moving horizontal line is superimposed on the live image
- "VerticalLineMoving"
A moving vertical line is superimposed on the live image
- "FrameCounter"
A frame counter is superimposed on the live image

Syntax

```
[VB]  
objActiveUSB.TestImageSelector [= Value]
```

```
[C/C++]  
HRESULT get_TestImageSelector( string *pValue );  
HRESULT put_TestImageSelector( string Value );
```

Data Type [VB]

String

Parameters [C/C++]

- pValue* [out, retval]
Pointer to the string specifying the test image selector setting
- Value* [in]
The test image selection to be set

Return Values

- S_OK
Success
- E_NOINTERFACE
The feature is not available for the selected camera
- E_INVALIDARG

The value is not part of the enumerated set
E_FAIL
Failure to set the feature value

Example

The following VB example demonstrates the use of a combo box to switch between available test images.

```
Private Sub Form_Load()  
Lst = ActiveUSB1.GetEnumList("TestImageSelector")  
For i = 0 To UBound(Lst)  
Combol.AddItem (Lst(i))  
Next  
Combol.ListIndex = ActiveUSB1.TestImageSelector  
End Sub  
  
Private Sub Combol_Click()  
ActiveUSB1.TestImageSelector = Combol.Text  
End Sub
```

Remarks

Depending on a camera model, other device-specific values may be available for this feature. The list of valid values can be retrieved using [GetEnumList](#).

Note that the property is available only if the currently selected camera supports the *TestImageSelector* feature per GenICam standard.

3.1.63 Trigger

Description

Enables/disables the selected trigger.

Syntax

[VB]

```
objActiveUSB.Trigger [= Value]
```

[C/C++]

```
HRESULT get_Trigger ( bool *pTrigger );  
HRESULT put_Trigger( bool Trigger );
```

Data Type [VB]

Boolean

Parameters [C/C++]

pTrigger [out,retval]

Pointer to the Boolean that is TRUE if the trigger mode is enable, or FALSE otherwise

Trigger [in]

Set to TRUE to enable the trigger mode, or set to FALSE otherwise

Return Values

S_OK

Success

E_NOINTERFACE

Triggering is not available for the selected camera

E_FAIL

Failure.

Example

This VB example activates the trigger mode:

The following VB example sets a hardware trigger which will start the capture of each frame on the rising edge of the signal coming from the physical input Line1:

```
ActiveUSB1.TriggerSelector = "FrameStart"  
ActiveUSB1.Trigger = True  
ActiveUSB1.TriggerActivation = "RisingEdge"  
ActiveUSB1.TriggerSource = "Line1"  
ActiveUSB1.Acquire = True
```

Remarks

This property corresponds to the *TriggerMode* feature per GenICam standard.

Before turning the trigger on, select a corresponding trigger configuration with [TriggerSelector](#).

3.1.64 TriggerActivation

Description

Returns or sets the activation mode for the selected trigger configuration. Can be one of the following values:

"RisingEdge"

Trigger will assert on the rising edge of the source signal

"FallingEdge"

Trigger will assert on the falling edge of the source signal

"AnyEdge"

Trigger will assert on both edges of the source signal

"LevelHigh"

Trigger will be valid as long as the level of the source signal is high

"LevelLow"

Trigger will be valid as long as the level of the source signal is low

Syntax

[VB]

```
objActiveUSB.TriggerActivation [= Value]
```

[C/C++]

```
HRESULT get_TriggerActivation( string *pValue );  
HRESULT put_TriggerActivation( string Value );
```

Data Type [VB]

String

Parameters [C/C++]

pValue [out, retval]

Pointer to the string specifying the trigger activation setting

Value [in]

The trigger activation mode to be set

Return Values

S_OK

Success

E_NOINTERFACE

The feature is not available for the selected camera

E_INVALIDARG

The value is not part of the enumerated set

E_FAIL

Failure to set the feature value

Example

The following VB example sets a hardware trigger which will start the capture of each frame on the rising edge of the signal coming from the physical input Line1:

```
ActiveUSB1.TriggerSelector = "FrameStart"  
ActiveUSB1.Trigger = True  
ActiveUSB1.TriggerActivation = "RisingEdge"  
ActiveUSB1.TriggerSource = "Line1"  
ActiveUSB1.Acquire = True
```

Remarks

Before setting up this property, select a corresponding trigger configuration with [TriggerSelector](#).

Depending on a camera model, other device-specific values may be available for this feature. The list of valid values can be retrieved using [GetEnumList](#).

Note that the property is available only if the currently selected camera supports the *TriggerActivation* feature per GenICam standard.

3.1.65 TriggerDelay

Description

Returns or sets the absolute value of the trigger delay in microseconds or raw units.

Syntax

```
[VB]  
objActiveUSB.TriggerDelay [= Value]
```

```
[C/C++]  
HRESULT get_TriggerDelay( float *pValue );  
HRESULT put_TriggerDelay( float Value );
```

Data Type [VB]

Single

Parameters [C/C++]

pValue [out,retval]
Pointer to the current trigger delay value
Value [in]
The trigger delay value to be set

Return Values

S_OK
Success
E_NOINTERFACE
The feature is not available for the selected camera
E_INVALIDARG
The value is out of range
E_FAIL
Failure to set the feature value

Example

The following VB example sets a hardware trigger which will start the capture of each frame on the rising edge of the signal coming from the physical input Line1:

```
ActiveUSB1.TriggerSelector = "FrameStart"  
ActiveUSB1.Trigger = True  
ActiveUSB1.TriggerActivation = "RisingEdge"  
ActiveUSB1.TriggerDelay = 10000.  
ActiveUSB1.TriggerSource = "Line1"  
ActiveUSB1.Acquire = True
```

Remarks

Trigger delay changes the time after the trigger reception before effectively activating it. Prior to setting

up this property, select a corresponding trigger configuration with [TriggerSelector](#). The valid property range can be retrieved by the [GetTriggerDelayMin](#) and [GetTriggerDelayMax](#) methods.

Note that the property is available only if the currently selected camera supports one of the following GenICam features: *TriggerDelay*, *TriggerDelayAbs*, *TriggerDelayRaw*.

3.1.66 TriggerSelector

Description

Returns or sets the configuration of a trigger signal. Can be one of the following values:

"AcquisitionStart"

Trigger will start the acquisition of one or several frames according to [AcquisitionMode](#)

"AcquisitionEnd"

Trigger will stop the acquisition of one or several frames according to [AcquisitionMode](#)

"AcquisitionActive"

Trigger will control the duration of the acquisition of one or several frames

"FrameStart"

Trigger will start the acquisition of a frame

"FrameEnd"

Trigger will end the acquisition of a frame (used in line scan cameras)

"FrameActive"

Trigger will control the duration of a frame (used in line scan cameras)

"LineStart"

Trigger will start the capture of one line of a frame (used in line scan cameras)

"ExposureStart"

Trigger will start the exposure of a frame (or line)

"ExposureEnd"

Trigger will end the exposure of a frame (or line)

"ExposureActive"

Trigger will control the duration of the exposure of a frame (or line)

Syntax

[VB]

```
objActiveUSB.TriggerSelector [= Value]
```

[C/C++]

```
HRESULT get_TriggerSelector( string *pValue );  
HRESULT put_TriggerSelector( string Value );
```

Data Type [VB]

String

Parameters [C/C++]

pValue [out, retval]

Pointer to the string specifying the trigger selector setting

Value [in]

The trigger selection to be set

Return Values

S_OK

Success

E_NOINTERFACE

The feature is not available for the selected camera

E_INVALIDARG

The value is not part of the enumerated set

E_FAIL
Failure to set the feature value

Example

The following VB example sets a hardware trigger which will start the capture of each frame on the rising edge of the signal coming from the physical input Line1:

```
ActiveUSB1.TriggerSelector = "FrameStart"  
ActiveUSB1.Trigger = True  
ActiveUSB1.TriggerActivation = "RisingEdge"  
ActiveUSB1.TriggerSource = "Line1"  
ActiveUSB1.Acquire = True
```

Remarks

Depending on a camera model, other device-specific values may be available for this feature. The list of valid values can be retrieved using [GetEnumList](#).

Note that the property is available only if the currently selected camera supports the *TriggerSelector* feature per GenICam standard.

3.1.67 TriggerSource

Description

Returns or sets the internal signal or physical input to be used as the trigger source for the selected trigger configuration. Can be one of the following values:

"Software"

Trigger signal will be generated by software using the [SoftTrigger](#) command.

"Line0", "Line1", "Line2",...

The physical line or pin to be used as external trigger source.

"Timer1Start", "Timer2Start", ..."Timer1End", "Timer2End",...

The Timer signal to be used as internal trigger source

"Counter1Start", "Counter2Start", ..."Counter1End", "Counter2End",...

The Counter signal to be used as internal trigger source

"UserInput0", "UserInput1", "UserInput2",...

The User Output bit signal to be used as internal trigger source

Syntax

[VB]

```
objActiveUSB.TriggerSource [= Value]
```

[C/C++]

```
HRESULT get_TriggerSource( string *pValue );  
HRESULT put_TriggerActivation( string Value );
```

Data Type [VB]

String

Parameters [C/C++]

pValue [out, retval]

Pointer to the string specifying the current trigger source

Value [in]

The trigger source to be set

Return Values

S_OK

Success

E_NOINTERFACE

The feature is not available for the selected camera

E_INVALIDARG

The value is not part of the enumerated set

E_FAIL

Failure to set the feature value

Example

The following VB example sets a hardware trigger which will start the capture of each frame on the rising edge of the signal coming from the physical input Line1:

```
ActiveUSB1.TriggerSelector = "FrameStart"  
ActiveUSB1.Trigger = True  
ActiveUSB1.TriggerActivation = "RisingEdge"  
ActiveUSB1.TriggerSource = "Line1"  
ActiveUSB1.Acquire = True
```

Remarks

Before setting up this property, select a corresponding trigger configuration with [TriggerSelector](#).

Depending on a camera model, other device-specific values may be available for this feature. The list of valid values can be retrieved using [GetEnumList](#).

Note that the property is available only if the currently selected camera supports the *TriggerSource* feature per GenICam standard.

3.1.68 UserOutputSelector

Description

Selects the bit of the User Output register to be set by [UserOutputValue](#). Can be one of the following values:

"UserOutput0"
Selects Bit 0 of the User Output register.
"UserOutput1"
Selects Bit 1 of the User Output register.
"UserOutput2"
Selects Bit 2 of the User Output register.
.....

Syntax

[VB]
`objActiveUSB.UserOutputSelector [= Value]`

[C/C++]
`HRESULT get_UserOutputSelector(string *pValue);`
`HRESULT put_UserOutputSelector(string Value);`

Data Type [VB]

String

Parameters [C/C++]

pValue [out, retval]
Pointer to the string specifying the bit in the User Output register.
Value [in]
The bit to be set

Return Values

S_OK
Success
E_NOINTERFACE
The feature is not available for the selected camera
E_FAIL
Failure to set the feature value

Example

The following VB example demonstrates the use of a combo box for line selection:

```
Private Sub Form_Load()  
Lst = ActiveUSB1.GetEnumList("UserOutputSelector")  
For i = 0 To UBound(Lst)  
Combo1.AddItem (Lst(i))  
Next  
Combo1.ListIndex = ActiveUSB1.UserOutputSelector
```

```
End Sub
```

```
Private Sub Combo1_Click()  
ActiveUSB1.UserOutputSelector = Combo1.Text  
End Sub
```

Remarks

This property works in combination with [UserOutputValue](#). Using a corresponding [LineSource](#), the bits from the User Output register can be directed to a physical output line.

Note that the property is available only if the currently selected camera supports the *UserOutputSelector* feature per GenICam standard.

3.1.69 UserOutputValue

Description

Returns or sets the value of the selected bit of the User Output register.

Syntax

```
[VB]  
objActiveUSB.UserOutputValue [= Value]
```

```
[C/C++]  
HRESULT get_UserOutputValue( VARIANT_BOOL *pValue );  
HRESULT put_UserOutputValue( VARIANT_BOOL Value );
```

Data Type [VB]

Boolean

Parameters [C/C++]

pValue [out, retval]
Pointer to the boolean value of the selected bit in the User Output register.
Value [in]
TRUE sets bit to High, FALSE sets bit to Low

Return Values

S_OK
Success
E_NOINTERFACE
The feature is not available for the selected camera
E_FAIL
Failure to set the feature value

Example

The following VB example demonstrates the setup of the bits in the User Output register:

```
ActiveUSB1.UserOutputSelector="UserOutput0"  
ActiveUSB1.UserOutputValue=TRUE  
ActiveUSB1.UserOutputSelector="UserOutput1"  
ActiveUSB1.UserOutputValue=FALSE
```

Remarks

This property works in combination with [UserOutputSelector](#). Using a corresponding [LineSource](#), the bits from the User Output register can be directed to a physical output line.

Note that the property is available only if the currently selected camera supports the *UserOutputValue* feature per GenICam standard.

3.1.70 UserSetSelector

Description

Returns or assigns the user set to load, save or configure. User sets allow for loading or saving factory or user-defined settings. Loading the factory default user set guarantees a state where a continuous acquisition can be started using only the mandatory features. Can be one of the following values:

"Default" (or "Factory")
 Selects the factory settings user set.
 "UserSet1"
 Selects the first user set.
 "UserSet2"
 Selects the second user set

Syntax

[VB]
`objActiveUSB.UserSetSelector [= Value]`

[C/C++]
`HRESULT get_UserSetSelector(string *pValue);`
`HRESULT put_UserSetSelector(string Value);`

Data Type [VB]

String

Parameters [C/C++]

pValue [out, retval]
 Pointer to the string specifying the user set.
Value [in]
 The user set to be selected.

Return Values

S_OK
 Success
 E_NOINTERFACE
 The feature is not available for the selected camera
 E_INVALIDARG
 The value is not part of the enumerated set
 E_FAIL
 Failure to set the feature value

Example

The following VB example demonstrates how to load the first user set.

```
ActiveUSB1.UserSetSelector="UserSet1"  
ActiveUSB1.SetFeatureString("UserSetLoad", "Execute")
```

Remarks

This feature should be used in combination with "UserSetLoad" and "UserSetSave" commands. To execute a command, use [SetFeatureString](#).

Depending on a camera model, other device-specific values may be available for this feature. The list of valid values can be retrieved using [GetEnumList](#).

Note that the property is available only if the currently selected camera supports the *UserSetSelector* feature per GenICam standard.

3.1.71 WebStream

Description

Enables/disables the RTSP web streaming. Used in combination with [SetWebStreamer](#).

Syntax

```
[VB]  
objActiveGige.WebStream [= Value]
```

```
[C/C++]  
HRESULT get_WebStream ( bool *pValue);  
HRESULT put_WebStream( bool Value );
```

Data Type [VB]

Boolean

Parameters [C/C++]

pValue [out,retval]
Pointer to the Boolean that is TRUE if the web streaming is enabled, or FALSE otherwise
Value [in]
Set to TRUE to enable the web streaming, or set to FALSE to disable it

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example sets the web streamer's parameters and activates the streaming:

```
sourceAddr="192.168.0.5:8554/ActiveUsb.sdp"  
destAddr="192.168.0.10"  
streamFPS=15  
streamQuality=4  
ActiveUSB1.SetWebStreamer sourceAddr, destAddr, streamFPS, streamQuality  
ActiveUSB1.Acquire=True  
ActiveUSB1.WebStream=True
```

Remarks

The web streaming option allows you to automatically convert video outputted by the camera into the H.264 compression format and transmit it over the wireless or wired network using the RTSP protocol to a remote playback devices, such as PCs, tablets and smartphones.

Before enabling the **WebStream** property, you should configure the web streaming parameters by calling [SetWebStreamer](#).

The [Acquire](#) property must be set to TRUE prior to activating the web streaming.

To watch a transmitted video on a remote playback device, use an RTSP/RTP client such as VLC Media Player (for Windows, Linux and MAC) or Fresh Video Player (for iOS devices). For more information refer to [SetWebStreamer](#).

Note - the web streaming module utilizes Intel's H.264 encoder which takes advantage of Intel's hardware acceleration. Therefore running your ActiveUSB-based web streaming application on a PC with Intel graphics chipset will substantially improve the encoding performance.

3.2 Methods

ActiveUSB provides the following methods to a container application:

Acquisition

Grab	Grabs a single frame into the internal memory
SoftTrigger	Generates an internal trigger signal

Information

GetCameraList	Returns the list of USB3 Vision cameras connected to the system
GetFormatList	Returns the list of pixel formats supported by the currently selected camera
GetBitsPerChannel	Returns the bit depth of each component of the internal image
GetBytesPerPixel	Returns the byte depth of the internal image
GetOptimalPacketSize	Returns the optimal packet size for the current network configuration

Features

<u>GetCategoryList</u>	Returns the array of categories under which the camera features are grouped
<u>GetFeatureList</u>	Returns the array of features grouped under the specified category
<u>IsFeatureAvailable</u>	Checks availability of the specified camera feature
<u>GetFeature</u>	Returns the numerical value of the specified camera feature
<u>SetFeature</u>	Sets the numerical value of the specified camera feature
<u>GetFeature64</u>	Returns the numerical value of the specified 64-bit camera feature
<u>SetFeature64</u>	Sets the numerical value of the specified 64-bit camera feature
<u>GetFeatureString</u>	Returns the string value of the specified camera feature
<u>SetFeatureString</u>	Sets the string value of the specified camera feature
<u>GetFeatureArray</u>	Returns the array of values associated with the specified camera feature
<u>SetFeatureArray</u>	Sets the array of values associated with the specified camera feature
<u>GetFeatureMin</u>	Returns the minimum value allowed for the specified camera feature
<u>GetFeatureMax</u>	Returns the maximum value allowed for the specified camera feature
<u>GetFeatureIncrement</u>	Returns the increment value for the specified integer feature
<u>GetEnumList</u>	Returns the array of string values representing the specified enumerated feature
<u>GetFeatureAccess</u>	Returns the information on the access to the specified camera feature
<u>GetFeatureDependents</u>	Returns the names of features dependent on the specified camera feature
<u>GetFeatureTip</u>	Returns the short description of the specified camera feature
<u>GetFeatureDescription</u>	Returns the detailed description of the specified camera feature
<u>GetFeatureType</u>	Returns the type of the specified camera feature
<u>GetFeatureVisibility</u>	Returns the information on the suggested visibility for the specified camera feature
<u>GetFeatureRepresentation</u>	Returns the information on the suggested representation for the specified camera feature

GetAcquisitionFrameRateMax	Returns the maximum value allowed for the camera's frame rate
GetAcquisitionFrameRateMin	Returns the minimum value allowed for the camera's frame rate
GetBalanceRatioMax	Returns the maximum value allowed for the white balance ratio
GetBalanceRatioMin	Returns the minimum value allowed for the white balance ratio
GetBlackLevelMax	Returns the maximum value allowed for the camera's black level
GetBalanceRatioMin	Returns the minimum value allowed for the camera's black level
GetExposureTimeMax	Returns the maximum value allowed for the camera's exposure time
GetExposureTimeMin	Returns the minimum value allowed for the camera's exposure time
GetGainMax	Returns the maximum value allowed for the camera's gain
GetGainMin	Returns the minimum value allowed for the camera's gain
GetTriggerDelayMax	Returns the maximum value allowed for the camera's trigger delay
GetTriggerDelayMin	Returns the minimum value allowed for the camera's trigger delay

Image Access

<u>GetPixel</u>	Returns the pixel value at the specified coordinates
<u>GetRGBPixel</u>	Returns the array of RGB values at the specified coordinates
<u>GetImageLine</u>	Returns the array of pixel values in the specified horizontal line
<u>GetComponentLine</u>	Returns the array of pixel values of the color component in the specified horizontal line
<u>GetImageWindow</u>	Returns the 2D array of pixel values in the specified rectangular area of the current frame
<u>SetImageWindow</u>	Copies the 2D array of pixel values to the selected window of the current frame
<u>GetImageData</u>	Returns the two dimensional array of pixel values in the currently acquired frame
<u>GetComponentData</u>	Returns the two dimensional array of pixel values in the specified color component
<u>GetRawData</u>	Returns the 2D array of raw values in the currently acquired data buffer
<u>GetImagePointer</u>	Returns the memory pointer to the specified pixel
<u>GetDIB</u>	Returns the handler to a Device Independent Bitmap
<u>GetPicture</u>	Returns the Picture object corresponding to the currently acquired frame
<u>GetChunkPointer</u>	Returns the pointer to the data associated with the specified chunk feature
<u>GetChunkSize</u>	Returns the size of the data associated with the specified chunk feature

Image and Video Capture

SaveImage	Saves the current frame buffer in the specified image file
LoadImage	Loads and displays the image from the specified image file
StartCapture	Starts video capture to the specified AVI file or series of image files
StopCapture	Stops video capture
GetCodecList	Returns the names of video codecs available in the system
GetCodec	Return the name of the currently selected video codec
SetCodec	Sets the video codec to be used for the AVI capture
ShowCodecDlg	Shows the configuration dialog of the currently selected video codec
ShowCompressionDlg	Shows the video compression dialog

Advanced Video Recording (DVR version only)

<u>CreateVideo</u>	Creates an AVI file for the video recording and preallocates its size
<u>StartVideoCapture</u>	Starts time-lapse video capture to the previously created AVI file
<u>StopVideoCapture</u>	Stops video capture to the AVI file
<u>SetCodecProperties</u>	Sets the generic parameters of the currently selected compression codec
<u>GetCodecProperties</u>	Returns the generic parameters of the currently selected compression codec
<u>GetAudioList</u>	Returns the names of audio recording devices available in the system
<u>SetAudioSource</u>	Sets the index of the audio recording device to be used during the AVI capture
<u>GetAudioSource</u>	Returns the index of the currently selected audio recording device for the AVI capture
<u>ShowAudioDlg</u>	Displays the audio Input dialog for adjusting the mixer properties of the audio source
<u>SetAudioLevel</u>	Sets the recording level of the currently selected audio device
<u>GetAudioLevel</u>	Returns the recording level of the currently selected audio device
<u>SetWebStreamer</u>	Configures parameters of the RTSP web streaming

Sequence Capture (DVR version only)

<u>CreateSequence</u>	Allocates memory for capturing a sequence of frames
<u>StartSequenceCapture</u>	Starts acquiring a sequence of frames into the memory
<u>StopSequenceCapture</u>	Stops acquiring a sequence of frames into the memory
<u>GetSequenceFrameCount</u>	Returns the current number of frames in the memory sequence
<u>SaveSequence</u>	Saves the memory sequence in the specified video file
<u>LoadSequence</u>	Loads the specified video file into the memory sequence
<u>GetSequenceWindow</u>	Returns 2D-array of pixel values of the selected window in a frame in the memory sequence
<u>GetSequenceRawData</u>	Returns 2D-array of raw pixel values in a frame in the memory sequence
<u>GetSequencePixel</u>	Returns the pixel value in the selected coordinates of the frame in the memory sequence
<u>GetSequencePointer</u>	Returns the memory pointer to a selected pixel of the frame in the memory sequence
<u>GetSequencePicture</u>	Returns the Picture object corresponding to a frame in the memory sequence
<u>GetSequenceTimestamp</u>	Returns the timestamp of the selected frame in the memory sequence

Video and Sequence Playback (DVR version only)

<u>OpenVideo</u>	Opens the specified video file or memory sequence for the playback
<u>PlayVideo</u>	Plays the currently open video file or memory sequence
<u>StopVideo</u>	Stops the playback of the video file or memory sequence
<u>CloseVideo</u>	Closes the currently open video file
<u>GetVideoFrameCount</u>	Returns the number of the frame in the currently open video file or memory sequence
<u>GetVideoPosition</u>	Returns the position of the current frame in the open video file or memory sequence
<u>SetVideoPosition</u>	Seeks, extracts and display the specified frame from the open video file or sequence
<u>GetVideoFPS</u>	Returns the frame rate of the currently opened video file or memory sequence
<u>SetVideoFPS</u>	Sets the frame rate of the currently opened video file or memory sequence
<u>GetVideoVolume</u>	Returns the playback volume level of the currently open AVI file
<u>SetVideoVolume</u>	Sets the playback volume level of the currently open AVI file
<u>SetVideoSync</u>	Sets the playback synchronization mode of the currently open AVI file
<u>TriggerVideo</u>	Advances the AVI playback to the next frame

Drawing

Draw	Displays the current frame in <i>ActiveUSB</i> window.
OverlayClear	Clears graphics and text from the overlay
OverlayEllipse	Draws an empty or filled ellipse in the overlay
OverlayLine	Draws a line in the overlay
OverlayPixel	Draws a pixel in the overlay
OverlayRectangle	Draws an empty or filled rectangle in the overlay
OverlayText	Draws a string of text in the overlay
DrawPixel	Draws a pixel in the current image frame
DrawLine	Draws a line in the current image frame
DrawRectangle	Draws a rectangle in the current image frame
DrawEllipse	Draws an ellipse in the current image frame
DrawText	Draws a string of text in the current image frame
DrawAlphaClear	Clears the alpha plane
DrawAlphaPixel	Draws a pixel in the alpha plane
DrawAlphaLine	Draws a line in the alpha plane
DrawAlphaRectangle	Draws an empty or filled rectangle in the alpha plane
DrawAlphaEllipse	Draws an empty or filled ellipse in the alpha plane
DrawAlphaText	Draws a string of text in the alpha plane

Image Processing

<u>SaveBkg</u>	Stores a dark or bright background image on the hard drive
<u>SetROI</u>	Sets the rectangular region of interest and luminance range
<u>GetROI</u>	Returns the settings of the currently selected ROI
<u>GetImageStat</u>	Returns the array containing statistical data of the current image frame
<u>GetHistogram</u>	Returns the histogram of the current image frame
<u>SetLUT</u>	Assigns the array of values for the software lookup table
<u>GetLUT</u>	Returns the array of values for the software lookup table
<u>SetGains</u>	Set the levels for the software gain control
<u>SetLevels</u>	Set the minimum and maximum levels for the Window/Level operation
<u>GetLevels</u>	Returns the minimum and maximum levels for the Window/Level operation
<u>SetColorMatrix</u>	Sets the matrix coefficients for the color correction operation
<u>SetLensDistortion</u>	Sets distortion parameters for the lens distortion correction
<u>GetBarcode</u>	Returns the character string decoded from a barcode found in the current frame

File Access

<u>GetFileList</u>	Returns the array of names of all files hosted in the device
<u>GetFileSize</u>	Returns the size of the specified file in the device
<u>GetFileAccessMode</u>	Returns the access mode in which the specified file can be opened in the device
<u>ReadFile</u>	Transfers the indicated amount of bytes from the specified file in the device
<u>WriteFile</u>	Transfers the indicated amount of bytes to the specified file in the device
<u>GetFileTransferProgress</u>	Returns the progress of the current file access operation in percent

Utilities

<u>GetFPS</u>	Returns the actual frame rate of the camera
<u>GetFPSAcquired</u>	Returns the acquired (displayed) frame rate of the application
<u>GetTimestamp</u>	Returns the timestamp of the last captured frame
<u>GetBlockId</u>	Returns the block ID of the last captured frame
<u>ShowProperties</u>	Displays property pages in run-time
<u>ReadRegister</u>	Reads a 32-bit value from the specified bootstrap register of the currently selected camera
<u>WriteRegister</u>	Write a 32-bit value to the specified bootstrap register of the currently selected camera
<u>ReadBlock</u>	Reads the block of data from the camera starting from the specified bootstrap address
<u>WriteBlock</u>	Writes the block of data to the camera starting from the specified bootstrap address
<u>LoadSettings</u>	Loads previously saved camera settings from the data file
<u>SaveSettings</u>	Stores the camera settings in the data file

3.2.1 CloseVideo

Description

Closes the video file currently opened by [OpenVideo](#).

Syntax

[VB]
objActiveUSB.CloseVideo

[C/C++]
HRESULT CloseVideo();

Parameters [C/C++]

None

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example opens an AVI file, plays it and closes the file when the playback is finished using the [PlayCompleted](#) event.

```
Private Sub Play_Click()  
ActiveUSB1.OpenVideo "C:\\video1.avi"  
ActiveUSB1.PlayVideo  
End Sub
```

```
Private Sub ActiveUSB1_PlayCompleted (ByVal Frames As Long)  
ActiveUSB1.CloseVideo  
End Sub
```

Remarks

If the video file is currently being played, this method will stop the playback and close the file.

3.2.2 CreateSequence

Description

Allocates memory for capturing a sequence of frames. Used in combination with [StartSequenceCapture](#) and [StopSequenceCapture](#).

Syntax

[VB]
`objActiveUSB.CreateSequence Frames`

[C/C++]
`HRESULT CreateSequence(long Frames);`

Data Types [VB]

Frames : long

Parameters [C/C++]

Frames [in]
The maximum number of frames to be allocated for the memory sequence.

Return Values

S_OK
Success
E_FAIL
Failure.
E_OUTOFMEMORY
Not enough memory

Example

This VB example demonstrates how to allocate enough memory for the sequence capture:

```
Private Sub Form_Load()  
ActiveUSB1.CreateSequence 1000  
ActiveUSB1.Acquire = True  
End Sub
```

```
Private Sub StartButton_Click()  
ActiveUSB1.StartSequenceCapture 800  
End Sub
```

```
Private Sub StopButton_Click()  
ActiveUSB1.StopSequenceCapture  
End Sub
```

Remarks

Using this method allows you to avoid an unnecessary delay when calling [StartSequenceCapture](#). If

enough memory has been allocated with **CreateSequence**, the [StartSequenceCapture](#) will start acquiring frames into the memory immediately after being called.

The amount of memory allocated with `CreateSequence` must not exceed the limit allowed for an application in your operating system. Such an amount is typically limited to 2 GB for a 32-bit version of Windows. If you use Windows 64-bit, it is recommended to keep the amount of allocated memory below the physical size of RAM in order to avoid the use of the virtual (hard-drive) space.

If the amount of memory you try to allocate exceeds the limit allowed for your application, this method will return an error.

3.2.3 CreateVideo

Description

Creates an AVI file for the video recording and preallocates its size. Used in combination with [StartVideoCapture](#) and [StopVideoCapture](#).

Syntax

[VB]
objActiveUSB.CreateVideo File [, Size = 0]

[C/C++]
HRESULT CreateVideo(bstr File, long Size);

Data Types [VB]

File : String
Size : Long (optional)

Parameters [C/C++]

File [in]
The string containing the path to the AVI file to be created.
Size [in]
The size of the file to allocate, in bytes.

Return Values

S_OK
Success
E_FAIL
Failure
E_INVALIDARG
Invalid file name
E_OUTOFMEMORY
Not enough space to allocate

Example

This VB example demonstrates how to preallocate an AVI file with the size of 80 MBytes and perform the video capture.

```
Private Sub LoadForm()  
ActiveUSB1.CreateVideo "c:\\mycapture.avi", 80000000  
ActiveUSB1.Acquire = True  
End Sub
```

```
Private Sub StartButton_Click()  
ActiveUSB1.StartVideoCapture  
End Sub
```

```
Private Sub StopButton_Click()  
ActiveUSB1.StopVideoCapture  
End Sub
```

Remarks

Use this method to create an AVI file and perform an initialization of the currently selected video codec prior to calling [StartVideoCapture](#).

It is recommended to allocate enough space to accommodate the length of a typical video recording. If the allocated size is not sufficient, the video capture engine will append the AVI file on the fly, but this may reduce the recording throughput and result to frames being dropped.

3.2.4 Draw

Description

Displays the current frame in *ActiveUSB* window.

Syntax

```
[VB]  
objActiveUSB.Draw
```

```
[C/C++]  
HRESULT Draw();
```

Parameters

None

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example uses the [FrameAcquired](#) event to invert pixel value in the bottom left corner of the current frame and display the processed frame in real time.

```
Private Sub Form_Load()  
ActiveUSB1.Display = False  
ActiveUSB1.Acquire = True  
End Sub  
  
Private Sub ActiveUSB1_FrameAcquired()  
a = ActiveUSB1.GetImageData  
For x = 0 To 200  
For y = 0 To 200  
a(x, y) = 255 - a(x, y)  
Next  
Next  
ActiveUSB1.Draw  
End Sub
```

Remarks

Use this method when the automatic live display is disabled ([Display](#) is set to *False*). This is typically done when you want to perform real time image processing and display the processed image in the control window.

3.2.5 DrawAlphaClear

Description

Clears graphics and text from the alpha plane and resets the plane to the fully transparent state.

Syntax

[VB]
`objActiveUSB.DrawAlphaClear`

[C/C++]
`HRESULT DrawAlphaClear();`

Parameters

None

Return Values

S_OK
Success
E_FAIL
Failure.

Example

The following VB example moves a transparent red rectangle over the live image by repeatedly erasing and drawing it:

```
Dim x As Integer

Private Sub Form_Load()
    x = 0
    ActiveUSB1.Acquire = True
    ActiveUSB1.Alpha = True
End Sub

Private Sub ActiveUSB1_FrameAcquired()
    ActiveUSB1.DrawAlphaClear
    ActiveUSB1.DrawAlphaRectangle x, 10, x + 50, 80, 0, 255,0,0,50
    x = x + 2
    If x = ActiveUSB1.SizeX Then
        x = 0
    End If
End Sub
```

Remarks

To create animation effects, use this method in combination with [DrawAlphaPixel](#), [DrawAlphaLine](#), [DrawAlphaRectangle](#), [DrawAlphaEllipse](#), [DrawAlphaText](#). For more information on the alpha-plane drawing refer to [Alpha](#).

3.2.6 DrawAlphaEllipse

Description

Draws an empty or filled ellipse in the alpha plane over the video window.

Syntax

[VB]

```
objActiveUSB.DrawAlphaElliplse X1, Y1, X2, Y2, Width, Red , Green, Blue [, Opacity]
```

[C/C++]

```
HRESULT DrawAlphaEllipse( short X1, short Y2, short X2, short Y2, short Width, long Red , long Green, long Blue [, short Opacity ] );
```

Data Types [VB]

X1, Y1, X2, Y2 : Integer

Width: Integer

Red, Green, Blue: Long

Opacity: Integer

Parameters [C/C++]

X1 [in], *Y1* [in]

Coordinates of the top left corner of the ellipse's bounding rectangle relative to the image origin.

X2 [in], *Y2* [in]

Coordinates of the bottom right corner of the ellipse's bounding rectangle relative to the image origin.

Width [in]

Width of the line in pixels. If zero, a filled ellipse will be drawn.

Red [in], *Green* [in], *Blue* [in]

Values specifying the color or intensity of the ellipse.

Opacity [in]

Value in the range 1-100 specifying the percentage of opacity at which the ellipse will be blended with the underlying pixels in the image.

If 0, the pixels of the ellipse will be reset to the fully transparent state.

If omitted, default value of 100 is used.

Return Values

S_OK

Success

E_FAIL

Failure.

Example

The following VB example draws a semi-transparent filled red ellipse in the alpha-plane over the live video:

```
ActiveUSB1.Acquire = True  
ActiveUSB1.DrawAlphaEllipse 100,100,200,200,0,255,0,0,50
```

```
ActiveUSB1.Alpha = True
```

Remarks

To draw multiple ellipses, call this method several times. For more information on the alpha-plane drawing refer to [Alpha](#).

3.2.7 DrawAlphaLine

Description

Draws a line in the alpha plane over the video window.

Syntax

[VB]

```
objActiveUSB.DrawAlphaLine X1, Y1, X2, Y2, Width, Red , Green, Blue [, Opacity]
```

[C/C++]

```
HRESULT DrawAlphaLine( short X1, short Y1, short X2, short Y2, short Width, long Red , long Green, long Blue [, short Opacity ]);
```

Data Types [VB]

X1, Y1, X2, Y2 : Integer

Width: Integer

Red, Green, Blue: Long

Opacity: Integer

Parameters [C/C++]

X1 [in], *Y1* [in], *X2* [in], *Y2* [in]

Coordinates of the starting and ending point of the line relative to the image origin.

Width [in]

Width of the line in pixels.

Red [in], *Green* [in], *Blue* [in]

Values specifying the color or intensity of the line.

Opacity [in]

Value in the range 1-100 specifying the percentage of opacity at which the line will be blended with the underlying pixels in the image.

If 0, the pixels of the line will be reset to the fully transparent state.

If omitted, default value of 100 is used.

Return Values

S_OK

Success

E_FAIL

Failure.

Example

The following VB example draws a solid green line of the width of 3 in the alpha-plane over the live video:

```
ActiveUSB1.Acquire = True  
ActiveUSB1.DrawAlphaLine 100,100,200,200,3,0,255,0  
ActiveUSB1.Alpha = True
```

Remarks

To draw multiple lines, call this method several times. For more information on the alpha-plane drawing refer to [Alpha](#).

3.2.8 DrawAlphaPixel

Description

Draws a pixel in the alpha plane over the live video.

Syntax

[VB]

```
objActiveUSB.DrawAlphaPixel X, Y, Red, Green, Blue [,Opacity ]
```

[C/C++]

```
HRESULT DrawAlphaPixel( short X, short Y, long Red,long Green, long Blue  
[,short Opacity ]);
```

Data Types [VB]

X, Y: Integer

Red, Green, Blue: Long

Opacity: Integer

Parameters [C/C++]

X [in], *Y* [in]

Coordinates of the pixel relative to the image origin.

Red [in], *Green* [in], *Blue* [in]

Values specifying the color or intensity of the pixel.

Opacity [in]

Value in the range 1-100 specifies the percentage of opacity at which the pixel will be blended with the underlying pixels in the image.

If 0, the pixel will be reset to the fully transparent state.

If omitted, default value of 100 is used.

Return Values

S_OK

Success

E_FAIL

Failure.

Example

The following VB example draws a blue pixel with the 80% opacity in the alpha-plane:

```
ActiveUSB1.Acquire = True  
ActiveUSB1.DrawAlphaPixel 100,100,200,200,0,0,255,80  
ActiveUSB1.Alpha = True
```

Remarks

To draw multiple pixels, call this method repeatedly. For more information on the alpha-plane drawing refer to [Alpha](#).

3.2.9 DrawAlphaRectangle

Description

Draws an empty or filled rectangle in the alpha plane over the video window.

Syntax

[VB]

```
objActiveUSB.DrawAlphaRectangle X1, Y1, X2, Y2, Width, Red , Green, Blue [, Opacity]
```

[C/C++]

```
HRESULT DrawAlphaRectangle( short X1, short Y1, short X2, short Y2, short Width, long Red , long Green, long Blue [, short Opacity ]);
```

Data Types [VB]

X1, Y1, X2, Y2 : Integer

Width: Integer

Red, Green, Blue: Long

Opacity: Integer

Parameters [C/C++]

X1 [in], *Y1* [in]

Coordinates of the top left corner of the rectangle's bounding rectangle relative to the image origin.

X2 [in], *Y2* [in]

Coordinates of the bottom right corner of the rectangle's bounding rectangle relative to the image origin.

Width [in]

Width of the line in pixels. If zero, a filled rectangle will be drawn.

Red [in], *Green* [in], *Blue* [in]

Values specifying the color or intensity of the rectangle.

Opacity [in]

Value in the range 1-100 specifying the percentage of opacity at which the ellipse will be blended with the underlying pixels in the image.

If 0, the pixels of the rectangle will be reset to the fully transparent state.

If omitted, default value of 100 is used.

Return Values

S_OK

Success

E_FAIL

Failure.

Example

The following VB example draws a semi-transparent filled red rectangle in the alpha-plane over the live video:

```
ActiveUSB1.Acquire = True  
ActiveUSB1.DrawAlphaRectangle 100,100,200,200,0,255,0,0,50
```

```
ActiveUSB1.Alpha = True
```

Remarks

To draw multiple rectangles, call this method several times. For more information on the alpha-plane drawing refer to [Alpha](#).

3.2.10 DrawAlphaText

Description

Embeds a string of text in the alpha plane over the live video.

Syntax

[VB]

```
objActiveUSB.DrawAlphaText X, Y, Text, Red, Green, Blue [,Opacity ]
```

[C/C++]

```
HRESULT DrawAlphaText( short X, short Y, bstr Text, long Red, long Green,  
long Blue [,short Opacity ]);
```

Data Types [VB]

X, Y: Integer

Text: String

Red, Green, Blue: Long

Opacity: Integer

Parameters [C/C++]

X [in], *Y* [in]

Coordinates of the beginning of the text relative to the image origin.

Text [in]

The string containing the text to be drawn.

Red [in], *Green* [in], *Blue* [in]

Values specifying the color or intensity of the text.

Opacity [in]

Value in the range 1-100 specifies the percentage of opacity at which the ellipse will be blended with the underlying pixels in the image.

If 0, the pixels of the text area will be reset to the fully transparent state.

If omitted, default value of 100 is used.

Return Values

S_OK

Success

E_FAIL

Failure.

Example

The following VB example draws a semi-transparent yellow text in the alpha plane:

```
ActiveUSB1.Acquire = True  
ActiveUSB1.DrawAlphaText 50,100, "ActiveUSB SDK for USB3 cameras", 255,255,0,50  
ActiveUSB1.Alpha = True
```

Remarks

To select the font for drawing text strings in the alpha plane, use the [OverlayFont](#) property. To draw

multiple strings, call this method several times. For more information on the alpha-plane drawing refer to [Alpha](#).

3.2.11 DrawEllipse

Description

Draws an empty or filled ellipse in the current image frame.

Syntax

[VB]

```
objActiveUSB.DrawElliplse X1, Y1, X2, Y2, Width, Red [,Green, Blue, Opacity  
]
```

[C/C++]

```
HRESULT DrawEllipse( short X1, short Y1, short X2, short Y2, short Width,  
long Red [,long Green, long Blue, short Opacity ]);
```

Data Types [VB]

X1, Y1, X2, Y2 : Integer

Width: Integer

Red, Green, Blue: Long

Opacity: Integer

Parameters [C/C++]

X1 [in], *Y1* [in]

Coordinates of the top left corner of the ellipse's bounding rectangle relative to the image origin.

X2 [in], *Y2* [in]

Coordinates of the bottom right corner of the ellipse's bounding rectangle relative to the image origin.

Width [in]

Width of the line in pixels. If zero, a filled ellipse will be drawn.

Red [in], *Green* [in], *Blue* [in]

Values specifying the color or intensity of the ellipse. If *Green* and *Blue* arguments are omitted, they will be assigned the value of *Red*.

Opacity [in]

An optional integer between 0 and 100 specifying the percentage of opacity at which the ellipse will be blended with the underlying pixels in the image. If omitted, default value of 100 is used.

Return Values

S_OK

Success

E_FAIL

Failure.

Example

The following VB example embeds a semi-transparent filled ellipse into the live video:

```
Private Sub ActiveUSB1_FrameAcquired()  
    ActiveUSB1.DrawEllipse 50,100, 300,200, 0, 0, 0, 255, 50  
End Sub
```

Remarks

To draw multiple ellipses, call this method several times.

3.2.12 DrawLine

Description

Draws a line in the current image frame.

Syntax

[VB]

```
objActiveUSB.DrawLine X1, Y1, X2, Y2, Width, Red [,Green, Blue, Opacity ]
```

[C/C++]

```
HRESULT DrawLine( short X1, short Y1, short X2, short Y2, short Width, long Red [,long Green, long Blue, short Opacity ]);
```

Data Types [VB]

X1, Y1, X2, Y2 : Integer

Width: Integer

Red, Green, Blue: Long

Opacity: Integer

Parameters [C/C++]

X1 [in], *Y1* [in], *X2* [in], *Y2* [in]

Coordinates of the starting and ending point of the line relative to the image origin.

Width [in]

Width of the line in pixels.

Red [in], *Green* [in], *Blue* [in]

Values specifying the color or intensity of the line. If *Green* and *Blue* arguments are omitted, they will be assigned the value of *Red*.

Opacity [in]

An optional integer between 0 and 100 specifying the percentage of opacity at which the line will be blended with the underlying pixels in the image. If omitted, default value of 100 is used.

Return Values

S_OK

Success

E_FAIL

Failure.

Example

The following VB example embeds a line into the live video:

```
Private Sub ActiveUSB1_FrameAcquired()  
    ActiveUSB1.DrawLine 50,100, 300,200, 3, 255  
End Sub
```

Remarks

To draw multiple lines, call this method several times.

3.2.13 DrawPixel

Description

Draws a pixel in the current image frame.

Syntax

[VB]

```
objActiveUSB.DrawPixel X, Y, Red [,Green, Blue, Opacity ]
```

[C/C++]

```
HRESULT DrawPixel( short X, short Y, long Red [,long Green, long Blue,  
short Opacity ]);
```

Data Types [VB]

X, Y: Integer

Red, Green, Blue: Long

Opacity: Integer

Parameters [C/C++]

X [in], *Y* [in]

Coordinates of the pixel relative to the image origin.

Red [in], *Green* [in], *Blue* [in]

Values specifying the color or intensity of the pixel. If *Green* and *Blue* arguments are omitted, they will be assigned the value of *Red*.

Opacity [in]

An optional integer between 0 and 100 specifying the percentage of opacity at which the pixel will be blended with the underlying pixel in the image. If omitted, default value of 100 is used.

Return Values

S_OK

Success

E_FAIL

Failure.

Example

The following VB example embeds a pixel into the live video:

```
Private Sub ActiveUSB1_FrameAcquired()  
    ActiveUSB1.DrawPixel 50,100, 255  
End Sub
```

Remarks

To draw multiple pixels, call this method several times.

3.2.14 DrawRectangle

Description

Draws an empty or filled rectangle in the current image frame.

Syntax

[VB]

```
objActiveUSB.DrawRectangle X1, Y1, X2, Y2, Width, Red [,Green, Blue,  
Opacity ]
```

[C/C++]

```
HRESULT DrawRectangle( short X1, short Y1, short X2, short Y2, short Width,  
long Red [,long Green, long Blue, short Opacity ]);
```

Data Types [VB]

X1, Y1, X2, Y2: Integer

Width: Integer

Red, Green, Blue: Long

Opacity: Integer

Parameters [C/C++]

X1 [in], *Y1* [in]

Coordinates of the top left corner of the rectangle relative to the image origin.

X2 [in], *Y2* [in]

Coordinates of the bottom right corner of the rectangle relative to the image origin.

Width [in]

Width of the line in pixels. If zero, a filled rectangle will be drawn.

Red [in], *Green* [in], *Blue* [in]

Values specifying the color or intensity of the rectangle. If *Green* and *Blue* arguments are omitted, they will be assigned the value of *Red*.

Opacity [in]

An optional integer between 0 and 100 specifying the percentage of opacity at which the rectangle will be blended with the underlying pixels in the image. If omitted, default value of 100 is used.

Return Values

S_OK

Success

E_FAIL

Failure.

Example

The following VB example embeds a semi-transparent filled rectangle into the live video:

```
Private Sub ActiveUSB1_FrameAcquired()  
    ActiveUSB1.DrawRectangle 50,100, 300,200, 0, 255, 0, 0, 50  
End Sub
```

Remarks

To draw multiple rectangles, call this method several times.

3.2.15 DrawText

Description

Embeds a string of text in the current image frame.

Syntax

[VB]

```
objActiveUSB.DrawText X, Y, Text, Red [,Green, Blue, Opacity ]
```

[C/C++]

```
HRESULT DrawText( short X, short Y, bstr Text, long Red [,long Green, long Blue, short Opacity ]);
```

Data Types [VB]

X, Y: Integer

Text: String

Red, Green, Blue: Long

Opacity: Integer

Parameters [C/C++]

X [in], *Y* [in]

Coordinates of the beginning of the text relative to the image origin.

Text [in]

The string containing the text to be drawn.

Red [in], *Green* [in], *Blue* [in]

Values specifying the color or intensity of the text. If *Green* and *Blue* arguments are omitted, they will be assigned the value of *Red*.

Opacity [in]

An optional integer between 0 and 100 specifying the percentage of opacity at which the text will be blended with the underlying pixels in the image. If omitted, default value of 100 is used.

Return Values

S_OK

Success

E_FAIL

Failure.

Example

The following VB example embeds a string of text into the live video:

```
Private Sub ActiveUSB1_FrameAcquired()  
    ActiveUSB1.DrawText 50,100, "ActiveUSB SDK for USB3 Vision cameras", 255  
End Sub
```

Remarks

To select the font for drawing text strings, use [Font](#). To draw multiple strings, call this method several times.

3.2.16 GetAcquisitionFrameRateMax

Description

Returns the maximum value allowed for the camera's frame rate.

Syntax

[VB]

```
Value=objActiveUSB.GetAcquistionFrameRateMax
```

[C/C++]

```
HRESULT GetAcquisitionFrameRateMax( float* pValue );
```

Data Types [VB]

Return value: Single

Parameters [C/C++]

pValue [out, retval]

Pointer to the maximum value of the feature

Return Values

S_OK

Success

E_NOINTERFACE

Feature does not exist or not available

E_FAIL

Failure to read the value

Example

This VB example uses the minimum and maximum values to initialize the slider control:

```
Private Sub Form_Load()  
Slider1.Min = ActiveUSB1.GetAcquisitionFrameRateMin  
Slider1.Max = ActiveUSB1.GetAcquisitionFrameRateMax  
Slider1.Value = ActiveUSB1.AcquisitionFrameRate  
End Sub
```

Remarks

Note that this method is available only if the currently selected camera supports one of the following GenICam features: *AcquisitionFrameRate*, *AcquisitionFrameRateAbs*, *AcquisitionFrameRateRaw*.

3.2.17 GetAcquisitionFrameRateMin

Description

Returns the minimum value allowed for the camera's frame rate.

Syntax

[VB]

```
Value=objActiveUSB.GetAcquistionFrameRateMin
```

[C/C++]

```
HRESULT GetAcquisitionFrameRateMin( float* pValue );
```

Data Types [VB]

Return value: Single

Parameters [C/C++]

pValue [out, retval]

Pointer to the minimum value of the feature

Return Values

S_OK

Success

E_NOINTERFACE

Feature does not exist or not available

E_FAIL

Failure to read the value

Example

This VB example uses the minimum and maximum values to initialize the slider control:

```
Private Sub Form_Load()  
Slider1.Min = ActiveUSB1.GetAcquisitionFrameRateMin  
Slider1.Max = ActiveUSB1.GetAcquisitionFrameRateMax  
Slider1.Value = ActiveUSB1.AcquisitionFrameRate  
End Sub
```

Remarks

Note that this method is available only if the currently selected camera supports one of the following GenICam features: *AcquisitionFrameRate*, *AcquisitionFrameRateAbs*, *AcquisitionFrameRateRaw*.

3.2.18 GetAudioLevel

Description

Returns the recording level of the currently selected audio device.

Syntax

```
[VB]  
Value=objActiveUSB.GetAudioLevel
```

```
[C/C++]  
HRESULT GetAudioLevel(short* pValue );
```

Data Types [VB]

Return value: Integer

Parameters [C/C++]

pValue [out,retval]
Pointer to the current audio recording level, in percent.

Return Values

S_OK
Success
E_FAIL
Failure

This VB example initiates an AVI recording with a sound track and uses a scroll bar to adjust the audio recording level.

```
Private Sub Form_Load()  
ActiveUSB1.SetAudioSource 0  
HScroll1.Min=0  
HScroll1.Max=100  
HScroll1.Value=ActiveUSB1.GetAudioLevel  
End Sub  
  
Private Sub Capture_Click  
ActiveUSB1.StartCapture "c:\\capture_with_sound.avi"  
End Sub  
  
Private Sub HScroll1_Scroll()  
ActiveUSB1.SetAudioLevel HScroll1.Value  
End Sub
```

Remarks

In order for this method to work, the audio recording device must be selected with [SetAudioSource](#).

3.2.19 GetAudioList

Description

Returns the array of strings containing the names of audio recording devices available in the system.

Syntax

[VB]

```
Value=objActiveUSB.GetAudioList()
```

[C/C++]

```
HRESULT GetAudioList( VARIANT* pList );
```

Data Types [VB]

Return value: Variant (SAFEARRAY)

Parameters [C/C++]

pList [out,retval]

Pointer to the SAFEARRAY of strings containing available categories

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example initializes a combo box with the list of available audio recording devices:

```
AudioLst = ActiveUSB1.GetAudioList  
For i = 0 To UBound(AudioLst)  
  Combo1.AddItem (AudioLst(i))  
Next  
Combo1.ListIndex = 0
```

Remarks

Note that in C/C++ applications it is required to call `SafeArrayDestroy()` to delete the SAFEARRAY returned by **GetAudioList**.

3.2.20 GetAudioSource

Description

Gets the index of the currently selected audio recording device for the AVI capture.

Syntax

```
[VB]  
Value = objActiveUSB.GetAudioSource
```

```
[C/C++]  
HRESULT GetAudioSource(short* Index);
```

Data Types [VB]

Return value: Integer

Parameters [C/C++]

pValue [out,retval]
Pointer to the index of the currently selected audio device.

Return Values

S_OK
Success
E_FAIL
Failure to set the codec

Example

The following VB example prints the name of the currently selected audio source:

```
AudioLst = ActiveUSB1.GetAudioList  
index=ActiveUSB1.GetAudioSource  
MsgBox AudioLst(index)
```

Remarks

If the return value is -1, the audio recording is disabled.

3.2.21 GetBalanceRatioMax

Description

Returns the maximum value allowed for the [BalanceRatio](#) property. Used for the white balance control.

Syntax

[VB]

```
Value=objActiveUSB.GetBalanceRatioMax
```

[C/C++]

```
HRESULT GetBalanceRatioMax( float* pValue );
```

Data Types [VB]

Return value: Single

Parameters [C/C++]

pValue [out, retval]

Pointer to the maximum value of the feature

Return Values

S_OK

Success

E_NOINTERFACE

Feature does not exist or not available

E_FAIL

Failure to read the value

Example

This VB example uses the minimum and maximum values to initialize the slider control:

```
Private Sub Form_Load()  
Slider1.Min = ActiveUSB1.GetBalanceRatioMin  
Slider1.Max = ActiveUSB1.GetBalanceRatioMax  
Slider1.Value = ActiveUSB1.BalanceRatio  
End Sub
```

Remarks

Note that this method is available only if the currently selected camera supports one of the following GenICam features: *BalanceRatio*, *BalanceRatioAbs*, *BalanceRatioRaw*.

3.2.22 GetBalanceRatioMin

Description

Returns the minimum value allowed for the [BalanceRatio](#) property. Used for the white balance control.

Syntax

[VB]

```
Value=objActiveUSB.GetBalanceRatioMin
```

[C/C++]

```
HRESULT GetBalanceRatioMin( float* pValue );
```

Data Types [VB]

Return value: Single

Parameters [C/C++]

pValue [out, retval]

Pointer to the minimum value of the feature

Return Values

S_OK

Success

E_NOINTERFACE

Feature does not exist or not available

E_FAIL

Failure to read the value

Example

This VB example uses the minimum and maximum values to initialize the slider control:

```
Private Sub Form_Load()  
Slider1.Min = ActiveUSB1.GetBalanceRatioMin  
Slider1.Max = ActiveUSB1.GetBalanceRatioMax  
Slider1.Value = ActiveUSB1.BalanceRatio  
End Sub
```

Remarks

Note that this method is available only if the currently selected camera supports one of the following GenICam features: *BalanceRatio*, *BalanceRatioAbs*, *BalanceRatioRaw*.

3.2.23 GetBarcode

Description

Returns the character string decoded from a barcode found in the current frame. The following 1D and 2D barcode symbologies are supported: UPC-A, UPC-E, EAN-8, Code 128, Code 39, Interleaved 2/5, QR Code, DataMatrix, PDF417

Syntax

[VB]
`Value=objActiveUSB.GetBarcode([Type])`

[C/C++]
`HRESULT GetFeatureArray(short Type, bstr* pValue);`

Data Types [VB]

Type: Integer
Return value: String

Parameters [C/C++]

Type [in]
An optional short value specifying the type of the barcode symbology:

- 0 - All [default]*
Will attempt to decode all supported barcode types.
- 1 - Linear*
Will attempt to decode all supported 1D barcode types.
- 2 - QR*
Will attempt to decode the QR code.
- 3 - DataMatrix*
Will attempt to decode the DataMatrix code.
- 4 - PDF417*
Will attempt to decode the PDF417 code.

pValue [out,retval]
Pointer to the bstr value containing the decoded string.

Return Values

S_OK
Success

Example

The following VB example continuously analyzes the incoming image frames for all supported barcode types and displays the string decoded:

```
Private Sub Form_Load()  
ActiveUSB1.Acquire = True
```

```
End Sub

Private Sub ActiveUSB1_FrameAcquired()
Label1.Caption = ActiveUSB1.GetBarcode()
End Sub
```

Remarks

To increase the decoding speed, it is recommended to use this method with the *Type* argument indicating the specific barcode type.

If several barcodes are present in the current frame, ActiveUSB will attempt to decode the first one found, in the top-to-bottom left-to-right order.

Note that this function does not return an error code. If the barcode has not been decoded, the returned string will be empty. If a critical error occur during the decoding, the string will contain one blank (space) character.

3.2.24 GetBitsPerChannel

Description

Returns the number of bits per color component of a pixel.

Syntax

[VB]
`Value=objActiveUSB.GetBitsPerChannel()`

[C/C++]
`HRESULT GetBitsPerChannel(long* pValue);`

Data Types [VB]

Return value: Long

Parameters [C/C++]

pValue [out,retval]
Pointer to the number of bytes per pixel

Return Values

S_OK
Success
E_FAIL
Failure

Example

This VB example reads and displays the pixel depth:

```
value=ActiveUSB1.GetBitsPerChannel()  
MsgBox value
```

Remarks

The method returns the pixel depth of the internal image buffer of the *ActiveUSB* object after the unpacking and color interpolation is performed. Depending on the selected format, the following value will be returned:

Pixel Format	Bits per channel
Mono8, Mono8Signed	8
Mono10, Mono10Packed	10
Mono12, Mono12Packed	12
Mono14	14
Mono16	16
Bayer**8, Bayer**8Packed, RGB8Packed, BGR8Packed, RGBA8Packed, BGRA8Packed, YUV411Packed, YUV422Packed, YUV444Packed, RGB8Planar	8
Bayer**10, Bayer**10Packed, RGB10Packed, BGR10Packed, BGR10V1Packed, BGR10V2Packed, RGB10Planar	10
Bayer**12, Bayer**12Packed, RGB12Packed, BGR12Packed, RGB12Planar	12
Bayer**16, RGB16Planar	16

3.2.25 GetBlackLevelMax

Description

Returns the maximum value allowed for the camera's black level..

Syntax

[VB]

```
Value=objActiveUSB.GetBlackLevelMax
```

[C/C++]

```
HRESULT GetBlackLevelMax( float* pValue );
```

Data Types [VB]

Return value: Single

Parameters [C/C++]

pValue [out, retval]

Pointer to the maximum value of the feature

Return Values

S_OK

Success

E_NOINTERFACE

Feature does not exist or not available

E_FAIL

Failure to read the value

Example

This VB example uses the minimum and maximum values to initialize the slider control:

```
Private Sub Form_Load()  
Slider1.Min = ActiveUSB1.GetBlackLevelMin  
Slider1.Max = ActiveUSB1.GetBlackLevelMax  
Slider1.Value = ActiveUSB1.BlackLevel  
End Sub
```

Remarks

Note that this method is available only if the currently selected camera supports one of the following GenICam features: *BlackLevel*, *BlackLevelAbs*, *BlackLevelRaw*.

3.2.26 GetBlackLevelMin

Description

Returns the minimum value allowed for the camera's black level..

Syntax

```
[VB]  
Value=objActiveUSB.GetBlackLevelMin
```

```
[C/C++]  
HRESULT GetBlackLevelMin( float* pValue );
```

Data Types [VB]

Return value: Single

Parameters [C/C++]

pValue [out, retval]
Pointer to the minimum value of the feature

Return Values

S_OK
Success
E_NOINTERFACE
Feature does not exist or not available
E_FAIL
Failure to read the value

Example

This VB example uses the minimum and maximum values to initialize the slider control:

```
Private Sub Form_Load()  
Slider1.Min = ActiveUSB1.GetBlackLevelMin  
Slider1.Max = ActiveUSB1.GetBlackLevelMax  
Slider1.Value = ActiveUSB1.BlackLevel  
End Sub
```

Remarks

Note that this method is available only if the currently selected camera supports one of the following GenICam features: *BlackLevel*, *BlackLevelAbs*, *BlackLevelRaw*.

3.2.27 GetBlockId

Description

Returns the block ID of the last acquired frame.

Syntax

[VB]

```
Value=objActiveUSB.GetBlockId
```

[C/C++]

```
HRESULT GetBlockId(long* pValue);
```

Data Types [VB]

Return value: Long

Parameters [C/C++]

pValue [out,retval]

Pointer to the timestamp value

Return Values

S_OK

Success

E_FAIL

Failure.

Example

This VB example displays the block ID of each frame in a text box.

```
Private Sub ActiveUSB1_FrameAcquired()  
    Label1.Caption = ActiveUSB1.GetBlockId  
End Sub
```

Remarks

Block ID is an ordinal number (starting from 1) assigned by the camera to each acquired frame. Block IDs can be used to identify missing frames.

If this method is used in the [FrameAcquired](#) event handler, it must be called immediately after the event has been received. This will guarantee that the value of the block ID will not be taken from the next frame while the current frame is being processed.

3.2.28 GetBytesPerPixel

Description

Returns the number of bytes per pixel for the current video [Format](#).

Syntax

```
[VB]  
Value=objActiveUSB.GetBytesPerPixel()
```

```
[C/C++]  
HRESULT GetBytesPerPixel(long* pValue );
```

Data Types [VB]

Return value: Long

Parameters [C/C++]

pValue [out,retval]
Pointer to the number of bytes per pixel

Return Values

S_OK
Success
E_FAIL
Failure

Example

This VB example reads and displays the pixel depth:

```
value=ActiveUSB1.GetBytesPerPixel()  
MsgBox value
```

Remarks

The method returns the pixel depth of the internal image buffer of the ActiveUSB object, which can be different from the pixel depth of the raw image outputted by the camera. For instance, the YUV 4:1:1 and YUV 4:2:2 images are always converted to the RGB 8:8:8 video, therefore the number of bytes per pixel reported for these formats will be 3. Also, when the [Bayer](#) color conversion is activated for Mono 8 and Mono 10/12/16 formats, the resulting video will have 3 or 6 bytes per pixel accordingly.

3.2.29 GetCameraIP

Description

Returns the IP address of the currently selected camera in Internet standard dotted format.

Syntax

[VB]

```
Value=objActiveUSB.GetCameraIP
```

[C/C++]

```
HRESULT GetCameraIP(bstr* pValue );
```

Data Types [VB]

Return value: String

Parameters [C/C++]

pValue [out, retval]

Pointer to the string value specifying the IP address

Return Values

S_OK

Success

E_FAIL

Failed to read the feature

Example

The following VB example prints the IP address if the camera:

```
MsgBox ActiveUSB1.GetCameraIP
```

Remarks

The IP address is returned in the Internet standard dotted format such as 169.254.102.77

3.2.30 GetCameraList

Description

Returns the array of strings containing the names of USB3 Vision™ cameras connected to the system. The cameras are listed in the alphabetical order "model + serial number".

Syntax

[VB]
Value=objActiveUSB.GetCameraList()

[C/C++]
HRESULT GetCameraList(VARIANT* pList);

Data Types [VB]

Return value: Variant (SAFEARRAY)

Parameters [C/C++]

pList [out,retval]
Pointer to the SAFEARRAY containing camera names

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example initializes a combo box with camera names and uses it to switch between the cameras:

```
Private Sub Form_Load()  
CamLst = ActiveUSB1.GetCameraList  
For i = 0 To UBound(CamLst)  
Comb1.AddItem (CamLst(i))  
Next  
Comb1.ListIndex = 0  
ActiveUSB1.Acquire = True  
End Sub  
  
Private Sub Comb1_Click()  
ActiveUSB1.Camera = Comb1.ListIndex  
End Sub
```

This MFC example fills out a combo box with camera names:

```
VARIANT m_CamArray=m_ActiveUSB.GetCameraList();  
SAFEARRAY *pArray=m_CamArray.parray;
```

```
UINT nCam=pArray->rgsabound[0].cElements;

CString strCamera;
CComboBox *pCamera=(CComboBox*)GetDlgItem(IDC_CAMERA);
for(UINT i=0;i<nCam;i++)
{
    CString str; str.Format("Camera %d",i);
    pCamera->AddString(str);
}
int iCam=m_ActiveUSB.GetCamera();
pCamera->SetCurSel(iCam);

SafeArrayDestroy(pArray);
```

Remarks

The index of an element in the camera list can be used as an argument of the [Camera](#) property to select a specific camera.

Note that in C/C++ applications it is required to call `SafeArrayDestroy()` to delete the SAFEARRAY returned by **GetCameraList**.

3.2.31 GetCameraMAC

Description

Returns the MAC address of the currently selected camera in the string form.

Syntax

[VB]

```
Value=objActiveUSB.GetCameraMAC
```

[C/C++]

```
HRESULT GetCameraMAC(bstr* pValue );
```

Data Types [VB]

Return value: String

Parameters [C/C++]

pValue [out, retval]

Pointer to the string value specifying the MAC address

Return Values

S_OK

Success

E_FAIL

Failed to read the feature

Example

The following VB example prints the MAC address if the camera:

```
MsgBox ActiveUSB1.GetCameraMAC
```

Remarks

The IP address is returned in standard dashed format such as 00-05-5D-24-6E-5B.

3.2.32 GetCategoryList

Description

Returns the array of strings containing the names of all categories and sub-categories under which the camera features are grouped.

Syntax

```
[VB]  
Value=objActiveUSB.GetCategoryList()
```

```
[C/C++]  
HRESULT GetCategoryList( VARIANT* pList );
```

Data Types [VB]

Return value: Variant (SAFEARRAY)

Parameters [C/C++]

pList [out,retval]
Pointer to the SAFEARRAY of strings containing available categories

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example initializes a combo box with the list of available categories:

```
CatLst = ActiveUSB1.GetCategoryList  
For i = 0 To UBound(CatLst)  
  Combol.AddItem (CatLst(i))  
Next  
Combol.ListIndex = 0
```

Remarks

The list of features grouped under a specific category can be retrieved with [GetFeatureList](#).

Note that in C/C++ applications it is required to call `SafeArrayDestroy()` to delete the SAFEARRAY returned by **GetCategoryList**.

3.2.33 GetChunkPointer

Description

Returns the pointer to the data associated with the specified chunk feature.

Syntax

[VB]

```
Value=objActiveUSB.GetChunkPointer( Name )
```

[C/C++]

```
HRESULT GetChunkPointer( bstr Name, VARIANT* pValue );
```

Data Types [VB]

Return value: Variant (pointer)

Parameters [C/C++]

Name [in]

String specifying the name of the chunk feature

pValue [out,retval]

Pointer to the variant containing the pointer to the data associated with the feature

Return Values

S_OK

Success

E_FAIL

Failure.

E_INVALIDARG

Invalid input arguments.

Example

This fragment of the C++ code grabs a frame, retrieves a pointer to the chunk data and copies them into a byte array.

```
VARIANT var; LONG var_size;  
char buffer[255]; short v;
```

```
pActiveUSB->SetFeature( OLESTR("ChunkEnable"), 1);  
pActiveUSB->Grab(&v);
```

```
pActiveUSB->GetChunkPointer( OLESTR( "ChunkInputStatusAtLineTriggerValue" ), &var);  
pActiveUSB->GetChunkSize(OLESTR( "ChunkInputStatusAtLineTriggerValue" ), &var_size);  
memcpy( buffer, (BYTE*)var.byref, var_size);
```

Remarks

Chunks are tagged blocks of auxiliary data that are transmitted along with each image frame. The **GetImagePointer** method provides the most efficient way to quickly access the chunk data in pointer-aware programming languages. It is especially useful for accessing data associated with array-type features.

This method should be used in combination with [GetChunkSize](#).

Note that when several chunk features share the same Chunk ID, this method will return the pointer to the beginning of the data block identified by the given Chunk ID. For more information on the chunk data refer to *"USB3 Vision Camera Interface Standard For Machine Vision"* published by the Automated Imaging Association.

3.2.34 GetChunkSize

Description

Returns the size of the data associated with the specified chunk feature.

Syntax

[VB]
`Value=objActiveUSB.GetChunkSize(Name)`

[C/C++]
`HRESULT GetChunkSize(bstr Name, long* pValue);`

Data Types [VB]

Return value: Variant (pointer)

Parameters [C/C++]

Name [in]
String specifying the name of the chunk feature
pValue [out,retval]
Pointer to the size of the chunk data in bytes.

Return Values

S_OK
Success
E_FAIL
Failure.
E_INVALIDARG
Invalid input arguments.

Example

This fragment of the C++ code grabs a frame, retrieves a pointer to the chunk data and copies them into a byte array.

```
VARIANT var; LONG var_size;  
char buffer[255]; short v;
```

```
pActiveUSB->SetFeature( OLESTR("ChunkEnable"), 1);  
pActiveUSB->Grab(&v);
```

```
pActiveUSB->GetChunkPointer( OLESTR( "ChunkInputStatusAtLineTriggerValue" ), &var);  
pActiveUSB->GetChunkSize(OLESTR( "ChunkInputStatusAtLineTriggerValue" ), &var_size);  
memcpy( buffer, (BYTE*)var.byref, var_size);
```

Remarks

Chunks are tagged blocks of auxiliary data that are transmitted along with each image frame. The **GetChunkSize** and [GetChunkPointer](#) methods provides the most efficient way to quickly access the chunk data in pointer-aware programming languages. It is especially useful for accessing data associated with array-type features.

Note that when several chunk features share the same Chunk ID, this method will return the size of the data block identified by the given Chunk ID. For more information on the chunk data refer to "*USB3 Vision Camera Interface Standard For Machine Vision*" published by the Automated Imaging Association.

3.2.35 GetCodec

Description

Gets the name of the currently selected codec (video compressor) for the AVI capture.

Syntax

[VB]

```
Value = objActiveUSB.GetCodec
```

[C/C++]

```
HRESULT GetCodec();
```

Data Types [VB]

Return value: String

Parameters [C/C++]

pValue [out,retval]

Pointer to the bstr value containing the name of the currently selected codec.

Return Values

S_OK

Success

E_FAIL

Failure to get the codec

Example

The following VB example prints the name of the currently selected codec:

```
MsgBox ActiveUSB1.GetCodec
```

3.2.36 GetCodecList

Description

Returns the array of strings containing the names of AVI codecs (compressors) available in the system.

Syntax

```
[VB]  
Value=objActiveUSB.GetCodecList()
```

```
[C/C++]  
HRESULT GetCodecList( VARIANT* pList );
```

Data Types [VB]

Return value: Variant (SAFEARRAY)

Parameters [C/C++]

pList [out,retval]
Pointer to the SAFEARRAY of strings containing available categories

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example initializes a combo box with the list of available codecs:

```
CodecLst = ActiveUSB1.GetCodecList  
For i = 0 To UBound(CodecLst)  
  Combol.AddItem (CodecLst(i))  
Next  
Combol.ListIndex = 0
```

Remarks

Note that in C/C++ applications it is required to call `SafeArrayDestroy()` to delete the SAFEARRAY returned by **GetCodecList**.

3.2.37 GetCodecProperties

Description

Returns the generic parameters of the currently selected compression codec.

Syntax

```
[VB]  
value=objActiveUSB.GetCodecProperties
```

```
[C/C++]  
HRESULT GetCodecProperties(VARIANT* pProperties);
```

Data Types [VB]

Return value: Variant (Array of Long values)

Parameters [C/C++]

pProperties [out,retval]

Pointer to the SAFEARRAY (integer) containing the values of the codec properties. The values are written in the array as follows:

Properties [0] – *Quality*. Numerical value of the quality parameter used by the codec, if supported.

Properties [1] – *DataRate*. Data rate in kb/sec used by the codec, if supported.

Properties [2] – *KeyFrameRate*. Amount of frames after which a key frame is recorded, if supported.

Properties [3] – *PFramesPerKey*. Rate of predicted (P) frames per key frame, if supported.

Properties [4] – *WindowSize*. Amount of frames over which the compressor maintains the average data rate.

Return Values

S_OK

Success

E_FAIL

Failure to retrieve the codec's properties

Example

The following VB example displays the quality and datarate settings of the MJPEG codec:

```
ActiveUSB1.SetCodec "MJPEG Compressor"  
CodecParams=ActiveUSB1.GetCodecProperties  
LabelQuality.Caption=CodecParams[0]  
LabelDatarate.Caption=CodecParams[1]
```

Remarks

This method allows you to retrieve only the basic compression properties which may not be supported by certain codecs. To access the internal parameters of a codec, use [ShowCodecDialog](#).

3.2.38 GetComponentData

Description

Returns the two-dimensional array of pixel values in the specified color component of the current frame.

Syntax

[VB]

```
Value=objActiveUSB.GetComponentData( Component )
```

[C/C++]

```
HRESULT GetComponentData( short Component, VARIANT* pArray );
```

Data Types [VB]

Y: Integer

Component: Integer

Return value: Variant (SAFEARRAY)

Parameters [C/C++]

Component [in]

The index of the selected color component. Must be one of the following values:

0 - returns the luminance data

1 - returns the red component data

2 - returns the green component data

3 - returns the blue component data

pArray [out,retval]

Pointer to the SAFEARRAY containing the pixel values in the frame

Return Values

S_OK

Success

E_FAIL

Failure.

E_INVALIDARG

Invalid input argument.

Example

This VB example grabs a frame, retrieves its green component and displays the value of the specified element of the array.

```
Dim pix as Long
ActiveUSB1.Grab
dataArray=ActiveUSB1.GetComponentData(1)
x=16
y=32
pix=dataArray(x,y)
```

```
if pix < 0 then
pix=65535-pix
endif
```

Remarks

The array returned by **GetComponentData** has the dimensions 0 to [SizeX](#) - 1, 0 to [SizeY](#) - 1. The type of data in the array depends on the format of the video acquired. For the 24-bit video the method will return an array of bytes. If the video is of the high-bit depth, the array will contain integer (word) values.

Note that modifying elements of the array will not change actual pixel values in the frame buffer. The image data in the array are stored in the standard order from top to bottom, therefore the first element of the array is first pixel of the top line of the image. This is opposite to the order of rows in the array returned by [GetImageData](#).

If the video has a grayscale format, the *Component* parameter will have no effect and the output array will contain the luminance data.

For integer (word) type of data you can receive negative numbers if pixel values exceed 32767. In C and C# you can convert signed integers to unsigned ones by using data casting. To get rid of negative values in VB and Delphi, subtract them from 65535 (see the example above).

Note that in C/C++ applications it is required to call `SafeArrayDestroy()` to delete the `SAFEARRAY` returned by **GetComponentData**.

3.2.39 GetComponentLine

Description

Returns the array of pixel values of the specified color component at the specified horizontal line of the current frame.

Syntax

[VB]

```
Value=objActiveUSB.GetComponentLine( Y, Component )
```

[C/C++]

```
HRESULT GetComponentLine( short Y, short Component, VARIANT* pArray );
```

Data Types [VB]

Y: Integer

Component: Integer

Return value: Variant (SAFEARRAY)

Parameters [C/C++]

Y [in]

The y-coordinate of the line in the image

Component [in]

The index of the selected color component. Must be one of the following values:

0 - returns the luminance data

1 - returns the red component data

2 - returns the green component data

3 - returns the blue component data

pArray [out,retval]

Pointer to the SAFEARRAY containing the pixel values in the line

Return Values

S_OK

Success

E_FAIL

Failure.

E_INVALIDARG

Invalid input argument.

Example

This VB example grabs a frame, retrieves the 32th row of green pixels and displays the value of 10th pixel in the row.

```
ActiveUSB1.Grab  
Line=ActiveUSB1.GetComponentLine(32,2)  
MsgBox Line(10)
```

Remarks

The array returned by **GetComponentLine** has the dimension range from 0 to [SizeX](#) - 1. The type of data in the array depends on the format of the video acquired. For the 24-bit video the method will return an array of bytes. If the video is of the high-bit depth, the array will contain integer (word) values. If the video has a grayscale format, the *Component* parameter will have no effect and the output array will contain the luminance data (see [GetLine](#)).

Note that modifying elements of the array will not change actual pixel values in the frame buffer.

The value of the y coordinate must not exceed the height of the video frame, or the error will occur.

Note that in C/C++ applications it is required to call `SafeArrayDestroy()` to delete the SAFEARRAY returned by **GetComponentLine**.

3.2.40 GetDIB

Description

Returns the handler to a Device Independent Bitmap.

Syntax

```
[VB]  
Value=objActiveUSB.GetDIB()
```

```
[C/C++]  
HRESULT GetDIB(HGLOBAL* pDib);
```

Data Types [VB]

Return value: long

Parameters [C/C++]

X [in]
The x-coordinate of the pixel
Y [in]
The y-coordinate of the pixel
pDib [out,retval]
Pointer to the handler to *ActiveUSB*'s display DIB

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This C example demonstrates how to retrieve and display *ActiveUSB*'s DIB

```
BITMAPINFO *pDIB;  
pActiveUSB->GetDIB((ULONG*)&pDIB);  
RECT rect; long sx,sy;  
GetWindowRect(hWnd,&rect);  
pActiveUSB->GetSizeX(&sx);  
pActiveUSB->GetSizeY(&sy);  
BYTE* pData=(BYTE*)pDIB+sizeof(BITMAPINFOHEADER) + pDIB->  
bmiHeader.biClrUsed * sizeof(RGBQUAD);  
SetDIBitsToDevice(hDC, 0, 0, sx, sy, 0, 0, 0, sy, pData, pDIB,  
DIB_RGB_COLORS);
```

Remarks

The **GetDIB** method provides the most efficient way to display the internal image buffer when you do

not want to use *ActiveUSB*'s live display. *ActiveUSB* uses packed DIBs, the pixel data immediately following the BITMAPINFO structure. The method returns a GlobalAlloc handler which contains the pointer to a DIB. The application must not free the global handler after using the DIB data.

Note that a DIB contains only 8- and 24-bit pixel values, even if the current video mode is a 16- or 48-bit one. To access the actual pixel data, use the [GetImageData](#) or [GetPointer](#) methods.

3.2.41 GetEnumList

Description

Returns the array of string values representing the specified enumerated feature of the camera.

Syntax

[VB]

```
Value=objActiveUSB.GetEnumList( Name )
```

[C/C++]

```
HRESULT GetEnumList( bstr Name, VARIANT* pInfo );
```

Data Types [VB]

Name: String

Return value: Variant (Array)

Parameters [C/C++]

Name [in]

String containing the name of the enumerated feature

pInfo [out,retval]

Pointer to the SAFEARRAY containing the allowed string values of the feature:

Return Values

S_OK

Success

E_NOINTERFACE

Feature does not exist or not available

E_INVALIDARG

Feature is not of enumerated type

E_FAIL

Failed to read feature.

Example

This VB example uses a combo box to switch between different auto exposure modes:

```
Private Sub Form_Load()  
Lst = ActiveUSB1.GetEnumList("ExposureAuto")  
For i = 0 To UBound(Lst)  
Combol.AddItem (Lst(i))  
Next  
Combol.ListIndex = ActiveUSB1.GetFeature("ExposureAuto")  
ActiveUSB1.Acquire = True  
End Sub
```

```
Private Sub Combol_Click()  
ActiveUSB1.SetFeatureString Combol.Text  
End Sub
```

Remarks

The index of an element in the string list can be used as an argument of [SetFeature](#) to set a specific value for the feature.

Note that in C/C++ applications it is required to call `SafeArrayDestroy()` to delete the `SAFEARRAY` returned by **GetEnumList**.

If the specified feature is not of the enumerated type or not supported by the camera, the method will return an error.

3.2.42 GetExposureTimeMax

Description

Returns the maximum value allowed for the camera exposure time, in microseconds or raw units..

Syntax

[VB]

```
Value=objActiveUSB.GetExposureTimeMax
```

[C/C++]

```
HRESULT GetExposureTimeMax( float* pValue );
```

Data Types [VB]

Return value: Single

Parameters [C/C++]

pValue [out, retval]

Pointer to the maximum value of the feature

Return Values

S_OK

Success

E_NOINTERFACE

Feature does not exist or not available

E_FAIL

Failure to read the value

Example

This VB example uses the minimum and maximum values to initialize the slider control:

```
Private Sub Form_Load()  
Slider1.Min = ActiveUSB1.GetExposureTimeMin  
Slider1.Max = ActiveUSB1.GetExposureTimeMax  
Slider1.Value = ActiveUSB1.ExposureTime  
End Sub
```

Remarks

Note that this method is available only if the currently selected camera supports one of the following GenICam features: *ExposureTime*, *ExposureTimeAbs*, *ExposureTimeRaw*.

3.2.43 GetExposureTimeMin

Description

Returns the minimum value allowed for the camera exposure time, in microseconds or raw units..

Syntax

[VB]

```
Value=objActiveUSB.GetExposureTimeMin
```

[C/C++]

```
HRESULT GetExposureTimeMin( float* pValue );
```

Data Types [VB]

Return value: Single

Parameters [C/C++]

pValue [out, retval]

Pointer to the minimum value of the feature

Return Values

S_OK

Success

E_NOINTERFACE

Feature does not exist or not available

E_FAIL

Failure to read the value

Example

This VB example uses the minimum and maximum values to initialize the slider control:

```
Private Sub Form_Load()  
Slider1.Min = ActiveUSB1.GetExposureTimeMin  
Slider1.Max = ActiveUSB1.GetExposureTimeMax  
Slider1.Value = ActiveUSB1.ExposureTime  
End Sub
```

Remarks

Note that this method is available only if the currently selected camera supports one of the following GenICam features: *ExposureTime*, *ExposureTimeAbs*, *ExposureTimeRaw*.

3.2.44 GetFeature

Description

Returns the numerical value of the specified camera feature.

Syntax

[VB]

```
Value=objActiveUSB.GetFeature( Name )
```

[C/C++]

```
HRESULT GetFeature(bstr Name, float* pValue );
```

Data Types [VB]

Name: String

Return value: Single

Parameters [C/C++]

Name [in]

String specifying the name of the feature

pValue [out, retval]

Pointer to the numerical value of the feature

Return Values

S_OK

Success

E_NOINTERFACE

Feature does not exist or not available

E_FAIL

Failed to read the feature

Example

The following VB example demonstrates the use of a scroll control for real-time adjustment of the "GainRaw" feature.

```
Private Sub Form_Load()  
ActiveUSB1.Acquire=True  
HScroll1.Value = ActiveUSB1.GetFeature("GainRaw")  
HScroll1.Min = ActiveUSB1.GetFeatureMin("GainRaw")  
HScroll1.Max = ActiveUSB1.GetFeatureMax("GainRaw")  
End Sub
```

```
Private Sub HScroll1_Scroll()  
ActiveUSB1.SetFeature("GainRaw", HScroll1.Value)  
End Sub
```

Remarks

Depending on the [Type](#) of the feature **GetFeature** will return the following values:

Feature Type	Return value
Integer	Integer value converted to floating point
Float	Floating point value
Boolean	0 if False, 1 if True
Enumerated	Ordinal number in the enumeration list
String	Zero value
Command	Zero value

If the currently selected camera does not support the specified feature, the method will generate an error.

To retrieve the string value of a feature, use [GetFeatureString](#).

3.2.45 GetFeature64

Description

Returns the numerical value of the specified 64-bit camera feature.

Syntax

[VB]

```
Value=objActiveUSB.GetFeature64( Name )
```

[C/C++]

```
HRESULT GetFeature64(bstr Name, double* pValue );
```

Data Types [VB]

Name: String

Return value: Double

Parameters [C/C++]

Name [in]

String specifying the name of the feature

pValue [out, retval]

Pointer to the numerical value of the feature

Return Values

S_OK

Success

E_NOINTERFACE

Feature does not exist or not available

E_FAIL

Failed to read the feature

Example

The following VB example demonstrates how to retrieve the value of the ChunkTimeStamp feature.

```
Private Sub Form_Load()  
ActiveUSB1.Acquire=True  
ActiveUSB1.SetFeature("ChunkModeActive",True)  
ActiveUSB1.SetFeature("ChunkModeEnable",True)  
End Sub  
  
Private Sub ActiveUSB1_FrameAcquired()  
Label1.Caption = ActiveUSB1.GetFeature64("ChunkTimeStamp")  
End Sub
```

Remarks

It is recommended to use this method only for those integer and floating point features whose length is 64-bit.

If the currently selected camera does not support the specified feature, the method will generate an error.

3.2.46 GetFeatureAccess

Description

Returns the information on availability and access to the specified camera feature.

Syntax

[VB]

```
Value=objActiveUSB.GetFeatureAccess ( Name )
```

[C/C++]

```
HRESULT GetFeatureAccess( bstr Name, bstr pValue );
```

Data Types [VB]

Name: String

Return value: String

Parameters [C/C++]

Name [in]

String specifying the name of the feature

pValue [out, retval]

Pointer to the string representing the access to the feature in the currently selected camera:

"NI" - feature is not implemented

"NA" - feature is not available at the moment

"WO" - feature has the write-only access

"RO" - feature has the read-only access

"RW" - feature has the full access

Return Values

S_OK

Success

E_FAIL

Failure

Example

This VB example displays the access mode of the feature:

```
MsgBox GetFeatureAccess("DeviceFirmwareVersion")
```

Remarks

Note that the access to a feature can change depending on the value of other features.

3.2.47 GetFeatureArray

Description

Returns an array of values associated with the specified camera feature.

Syntax

[VB]

```
Value=objActiveUSB.GetFeatureArray( Name, ElementSize )
```

[C/C++]

```
HRESULT GetFeatureArray(bstr Name, short ElementSize, VARIANT* pArray);
```

Data Types [VB]

Name: String

ElementSize: Integer

Return value: Variant (SAFEARRAY)

Parameters [C/C++]

Name [in]

String specifying the name of the feature

ElementSize [in]

A short value specifying the size of each array's element, in bytes

pArray [out, retval]

Pointer to the SAFEARRAY containing the values of the buffer

Return Values

S_OK

Success

E_NOINTERFACE

Feature does not exist or not available

E_FAIL

Failed to read the feature

Example

The following VB example creates an inverse lookup table and copies it into the camera using the "LUTValueAll" feature:

```
Dim LUT(255) As Long
For i = 0 To 255
LUT(i) = 255-i
Next
ActiveUSB1.SetFeatureArray "LUTValueAll", 4, LUT
ActiveUSB1.SetFeature "LUTEnable", True
```

Remarks

This method sets the content of buffers associated with features of the IRegister type. For more information refer to GenICam standard specifications.

Note that the size of an element in the binary buffer linked to an IRegister feature cannot be determined automatically. Therefore, the *ElementSize* parameter is not optional and must specify the size of each element of the array in bytes.

3.2.48 GetFeatureDependents

Description

Returns the array of strings containing the names of camera features dependent on the specified feature.

Syntax

[VB]

```
Value=objActiveUSB.GetFeatureDependents( Name )
```

[C/C++]

```
HRESULT GetFeatureList( bstr Name, VARIANT* pList );
```

Data Types [VB]

Name: String

Return value: Variant (SAFEARRAY)

Parameters [C/C++]

Name [in]

Name of the feature for which the list of dependent features is requested.

pList [out,retval]

Pointer to the SAFEARRAY of strings containing dependent features.

Return Values

S_OK

Success

E_NOINTERFACE

Category is not supported by the camera

E_FAIL

Failure.

Example

This VB example retrieves the list of features dependent on the "TriggerSelector" feature:

```
FeatureLst = ActiveUSB1.GetFeatureDependents("TriggerSelector")
```

Remarks

Use this method to determine the features whose values may change when the value of the specified feature is changing. Your application can use it to update the values of the controls linked to the dependant features.

Note that in C/C++ applications it is required to call SafeArrayDestroy() to delete the SAFEARRAY returned by **GetFeatureDependents**.

3.2.49 GetFeatureDescription

Description

Returns the detailed description of the specified camera feature.

Syntax

[VB]

```
Value=objActiveUSB.GetFeatureDescription ( Name )
```

[C/C++]

```
HRESULT GetFeatureDescription(bstr Name, bstr* pType );
```

Data Types [VB]

Name: String

Return value: String

Parameters [C/C++]

Name [in]

String specifying the name of the feature

pType [out, retval]

Pointer to a string specifying the description of the feature

Return Values

S_OK

Success

E_NOINTERFACE

Feature does not exist or not available

E_FAIL

Failure

Example

This VB example displays the description of the specified feature:

```
MsgBox ActiveUSB1.GetFeatureDescription("ExposureAuto")
```

Remarks

Feature descriptions are retrieved from an XML file embedded in a camera per GenICam standard.

3.2.50 GetFeatureList

Description

Returns the array of strings containing the names of camera features and categories grouped under the specified category.

Syntax

```
[VB]  
Value=objActiveUSB.GetFeatureList( Category )
```

```
[C/C++]  
HRESULT GetFeatureList( bstr Category, VARIANT* pList );
```

Data Types [VB]

Category: String
Return value: Variant (SAFEARRAY)

Parameters [C/C++]

Category [in]
Name of the category or sub-category for the list of features. If this parameter is "Root", the list of upper-level categories and features will be returned.

pList [out,retval]
Pointer to the SAFEARRAY of strings containing the list of features under the specified category. An empty array indicates that the name of a feature is used instead of the name of a category.

Return Values

S_OK
Success

E_NOINTERFACE
Category is not supported by the camera

E_FAIL
Failure.

Example

This VB example initializes a combo box with the names of the camera features grouped under the "AnalogControls" category:

```
Private Sub Form_Load()  
FeatureLst = ActiveUSB1.GetFeatureList("AnalogControls")  
For i = 0 To UBound(FeatureLst)  
Combo1.AddItem (FeatureLst(i))  
Next  
Combo1.ListIndex = 0  
ActiveUSB1.Acquire = True  
End Sub
```

```
Private Sub Combo1_Click()  
ActiveUSB1.Feature = Combo1.ListIndex  
End Sub
```

This MFC example fills out a combo box with camera features grouped under the "TriggerAcquisition" category:

```
VARIANT m_FeatureArray=m_ActiveUSB.GetFeatureList("TriggerAcquisition");  
SAFEARRAY *pArray=m_FeatureArray.parray;  
UINT nFeatures=pArray->rgsabound[0].cElements;  
  
CString strFeature;  
CComboBox *pFeature=(CComboBox*)GetDlgItem(IDC_FEATURE);  
for(UINT i=0;i<nFeatures;i++)  
{  
    pFeature->AddString(strFeature);  
}  
int iFeature=m_ActiveUSB.GetFeature();  
pFeature->SetCurSel(iFeature);  
  
SafeArrayDestroy(pArray);
```

Remarks

To get or set the value of a specific camera feature, use [GetFeature](#), [GetFeatureString](#), [SetFeature](#), [SetFeatureString](#)

Note that in C/C++ applications it is required to call `SafeArrayDestroy()` to delete the `SAFEARRAY` returned by **GetFeatureList**.

3.2.51 GetFeatureIncrement

Description

Returns the increment value for the specified integer feature.

Syntax

[VB]

```
Value=objActiveUSB.GetFeatureIncrement ( Name )
```

[C/C++]

```
HRESULT GetFeatureIncrement( bstr Name, long* pValue );
```

Data Types [VB]

Name: String

Return value: Long

Parameters [C/C++]

Name [in]

String specifying the name of the feature

pValue [out, retval]

Pointer to the increment value of the feature

Return Values

S_OK

Success

E_NOINTERFACE

Feature does not exist or not available

E_INVALIDARG

Feature is not of integer type

E_FAIL

The gain control is not available for the selected camera

Example

This VB example uses the increment value of the horizontal image size to initialize the scroll control:

```
Private Sub Form_Load()  
HScroll1.Min = ActiveUSB1.GetFeatureMin ("Width")  
HScroll1.Max = ActiveUSB1.GetFeatureMax ("Width")  
HScroll1.SmallChange = ActiveUSB1.GetFeatureIncrement ("Width")  
HScroll1.Value = ActiveUSB1.GetFeature ("Width")  
End Sub
```

Remarks

The increment of the feature defines the minimum value at which the feature can increase or

decrease. Available only for integer features.

3.2.52 GetFeatureMin

Description

Returns the minimum numerical value allowed for the specified camera feature.

Syntax

[VB]

```
Value=objActiveUSB.GetFeatureMin ( Name )
```

[C/C++]

```
HRESULT GetFeatureMin( bstr Name, float* pValue );
```

Data Types [VB]

Name: String

Return value: Single

Parameters [C/C++]

Name [in]

String specifying the name of the feature

pValue [out, retval]

Pointer to the minimum value of the feature

Return Values

S_OK

Success

E_NOINTERFACE

Feature does not exist or not available

E_FAIL

Failure to read the value

Example

This VB example uses the minimum and maximum gain values to initialize the slider control:

```
Private Sub Form_Load()  
Slider1.Min = ActiveUSB1.GetFeatureMin("GainRaw")  
Slider1.Max = ActiveUSB1.GetFeatureMax("GainRaw")  
Slider1.Value = ActiveUSB1.GetFeature("GainRaw")  
End Sub
```

Remarks

Depending on the [Type](#) of the feature **GetFeatureMin** will return the following values:

Feature Type	Return value
Integer	Minimum allowed value converted to floating point
Float	Minimum allowed value
Boolean	Zero value
Enumerated	Zero value
String	Zero value
Command	Zero value

If the currently selected camera does not support the specified feature, the method will generate an error.

3.2.53 GetFeatureMax

Description

Returns the maximum numerical value allowed for the specified camera feature.

Syntax

[VB]

```
Value=objActiveUSB.GetFeatureMin ( Name )
```

[C/C++]

```
HRESULT GetFeatureMin( bstr Name, float* pValue );
```

Data Types [VB]

Name: String

Return value: Single

Parameters [C/C++]

Name [in]

String specifying the name of the feature

pValue [out, retval]

Pointer to the maximum value of the feature

Return Values

S_OK

Success

E_NOINTERFACE

Feature does not exist or not available

E_FAIL

Failure to read the value

Example

This VB example uses the minimum and maximum gain values to initialize the slider control:

```
Private Sub Form_Load()  
Slider1.Min = ActiveUSB1.GetFeatureMin("GainRaw")  
Slider1.Max = ActiveUSB1.GetFeatureMax("GainRaw")  
Slider1.Value = ActiveUSB1.GetFeature("GainRaw")  
End Sub
```

Remarks

Depending on the [Type](#) of the feature **GetFeatureMax** will return the following values:

Feature Type	Return value
Integer	Maximum allowed value converted to floating point
Float	Maximum allowed value
Boolean	1.
Enumerated	Maximum ordinal number in the enumeration list
String	Zero value
Command	1.

If the currently selected camera does not support the specified feature, the method will generate an error.

3.2.54 GetFeatureRepresentation

Description

Returns the information on the suggested representation for the specified camera feature.

Syntax

[VB]

```
Value=objActiveUSB.GetFeatureAccess ( Name )
```

[C/C++]

```
HRESULT GetFeatureAccess( bstr Name, bstr pValue );
```

Data Types [VB]

Name: String

Return value: String

Parameters [C/C++]

Name [in]

String specifying the name of the feature

pValue [out, retval]

Pointer to the string specifying the representation of the feature:

"Linear" - feature should be represented by a slider with the linear behaviour

"Logarithmic" - feature should be represented by a slider with the logarithmic behaviour

"Boolean" - feature should be represented by a check box

"PureNumber" - feature should be represented by a text box with decimal display

"HexNumber" - feature should be represented by a text box with hexadecimal display

"Undefined" - no representation information available for the feature

Return Values

S_OK

Success

E_FAIL

Failure

Example

This VB example displays the representation of the feature:

```
MsgBox GetFeatureRepresentation("ExposureTimeAbs")
```

Remarks

Per GenICam standard, the representation of a feature gives developers a hint about how to display and control a specific camera feature in the GUI.

3.2.55 GetFeatureString

Description

Returns the string value of the specified camera feature.

Syntax

[VB]

```
Value=objActiveUSB.GetFeatureString( Name )
```

[C/C++]

```
HRESULT GetFeatureString(bstr Name, bstr* pValue );
```

Data Types [VB]

Name: String

Return value: String

Parameters [C/C++]

Name [in]

String specifying the name of the feature

pValue [out, retval]

Pointer to the string value of the feature

Return Values

S_OK

Success

E_NOINTERFACE

Feature does not exist or not available

E_FAIL

Failed to read the feature

Example

The following VB example prints the value of the "GainAuto" feature:

```
MsgBox ActiveUSB1.GetFeatureString("GainAuto")
```

Remarks

Depending on the [Type](#) of the feature the *Value* argument has the following meaning:

Feature Type	Value
Integer	Integer value in form of string
Float	Floating point value in form of string
Boolean	"False" or "True"
Enumerated	Current string value
String	Current string value
Command	Zero value

If the currently selected camera does not support the specified feature, the method will generate an error.

3.2.56 GetFeatureTip

Description

Returns the tooltip (short description) for the specified camera feature.

Syntax

[VB]

```
Value=objActiveUSB.GetFeatureTip ( Name )
```

[C/C++]

```
HRESULT GetFeatureTip(bstr Name, bstr* pValue );
```

Data Types [VB]

Name: String

Return value: String

Parameters [C/C++]

Name [in]

String specifying the name of the feature

pValue [out, retval]

Pointer to a string specifying the short description of the feature

Return Values

S_OK

Success

E_NOINTERFACE

Feature does not exist or not available

E_FAIL

Failure

Example

This VB example displays the short description of the specified feature:

```
MsgBox ActiveUSB1.GetFeatureTip("ExposureAuto")
```

Remarks

Feature tooltips are retrieved from an XML file embedded in a camera per GenICam standard.

3.2.57 GetFeatureType

Description

Returns the type of the specified camera feature.

Syntax

[VB]

```
Value=objActiveUSB.GetFeatureType ( Name )
```

[C/C++]

```
HRESULT GetFeatureType(bstr Name, bstr* pType );
```

Data Types [VB]

Name: String

Return value: String

Parameters [C/C++]

Name [in]

String specifying the name of the feature

pType [out, retval]

Pointer to a string specifying the type of the feature

Return Values

S_OK

Success

E_NOINTERFACE

Feature does not exist or not available

E_FAIL

Failure

Example

This VB example prints the type of the specified feature:

```
MsgBox ActiveUSB1.GetFeatureType("ExposureAuto")
```

Remarks

Depending on the type of the feature the method returns the following string values:

Feature Type	Value
Integer	"Int"
Float	"Float"
Boolean	"Bool"
Enumerated	"Enum"
String	"String"
Command	"Command"
Feature category	"Category"
Unrecognized type	"Unknown"

3.2.58 GetFeatureVisibility

Description

Returns the information on the recommended visibility for the specified camera feature.

Syntax

[VB]

```
Value=objActiveUSB.GetFeatureAccess ( Name )
```

[C/C++]

```
HRESULT GetFeatureAccess( bstr Name, bstr pValue );
```

Data Types [VB]

Name: String

Return value: String

Parameters [C/C++]

Name [in]

String specifying the name of the feature

pValue [out, retval]

Pointer to the string specifying the visibility of the feature:

"Beginner" - access to the feature should be available for all users

"Expert" - access to the feature should be available for intermediate-level users

"Guru" - access to the feature should be limited to advanced-level users

"Invisible" - access to the feature should be available only in the API (no GUI visibility)

"Undefined" - no visibility information available for the feature

Return Values

S_OK

Success

E_FAIL

Failure

Example

This VB example displays the visibility of the feature:

```
MsgBox GetFeatureVisibility("GevTimestampControl")
```

Remarks

Per GenICam standard, the visibility of a feature gives developers a hint about how to limit the access to a specific feature depending on the user's qualification.

3.2.59 GetFileAccessMode

Description

Returns the access mode in which the specified file can be opened in the device.

Syntax

[VB]

```
Value=objActiveUSB.GetFileAccessMode ( Name )
```

[C/C++]

```
HRESULT GetFileAccessMode( bstr Name, bstr pValue );
```

Data Types [VB]

Name: String

Return value: String

Parameters [C/C++]

Name [in]

String specifying the name of the file in the device

pValue [out, retval]

Pointer to the string representing the access mode to the file in the currently selected device:

"WO" - file has the write-only access

"RO" - file has the read-only access

"RW" - file has both read and write access

Return Values

S_OK

Success

E_NOINTERFACE

File with the specified name does not exist in the device

E_FAIL

Failure

Example

This VB example displays the access mode of the file:

```
MsgBox GetFileAccessMode("UserSet1")
```

Remarks

The File Access functionality allows an application to read and write files hosted on the device. The data in those files may contain look-up tables, configuration sets, firmware, and other information.

This method lets you determine whether the file can be accessed for [Reading](#), [Writing](#), or for both operations.

3.2.60 GetFileList

Description

Returns the array of strings containing the names of all files in the device that can be accessed via the File Access functionality.

Syntax

[VB]
Value=objActiveUSB.GetFileList

[C/C++]
HRESULT GetFileList (VARIANT* pFiles);

Data Types [VB]

Return value: Variant (Array of strings)

Parameters [C/C++]

pFiles [out,retval]
Pointer to the SAFEARRAY of strings containing the list of files hosted in the camera. An empty array indicates that the File Access is not supported by the camera.

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example initializes a combo box with the names of files hosted on the camera and displays the size of the selected file:

```
Private Sub Form_Load()  
FileList = ActiveUSB1.GetFileList  
For i = 0 To UBound(FileList)  
Combo1.AddItem (FileList(i))  
Next  
End Sub  
  
Private Sub Combo1_Click()  
FileName=Combo1.Text  
Label1.Caption = ActiveUSB1.GetFileSize (FileName)  
End Sub
```

Remarks

The File Access functionality allows an application to read and write files hosted on the device. The data in those files may contain look-up tables, configuration sets, firmware, and other information.

This method lets you determine if the device hosts any files and retrieve their names.

3.2.61 GetFileSize

Description

Returns the size of the specified file in the device in bytes.

Syntax

[VB]

```
Value=objActiveUSB.GetFileSize( Name )
```

[C/C++]

```
HRESULT GetFileSize(bstr Name, long* pValue );
```

Data Types [VB]

Name: String

Return value: Single

Parameters [C/C++]

Name [in]

String specifying the name of the file in the device

pValue [out, retval]

Pointer to the size of the file in bytes

Return Values

S_OK

Success

E_NOINTERFACE

File with the specified name does not exist in the device

E_FAIL

Failure

Example

This VB example initializes a combo box with the names of files hosted on the camera and displays the size of the selected file:

```
Private Sub Form_Load()  
FileList = ActiveUSB1.GetFileList  
For i = 0 To UBound(FileList)  
Comb1.AddItem (FileList(i))  
Next  
End Sub  
  
Private Sub Comb1_Click()  
FileName=Comb1.Text  
Label1.Caption = ActiveUSB1.GetFileSize (FileName)  
End Sub
```

Remarks

The File Access functionality allows an application to read and write files hosted on the device. The data in those files may contain look-up tables, configuration sets, firmware, and other information.

The size of the file can be used to reserve a proper amount of memory for the data transfer or to determine the maximum amount of bytes that can be written into the file.

3.2.62 GetFileTransferProgress

Description

Returns the progress of the current file access operation in percent.

Syntax

[VB]

```
Value=objActiveUSB.GetFileTransferProgress( Name )
```

[C/C++]

```
HRESULT GetFileTransferProgress(bstr Name, float* pValue );
```

Data Types [VB]

Name: String

Return value: Single

Parameters [C/C++]

Name [in]

String specifying the name of the file

pValue [out, retval]

Pointer to the value indicating the progress of the current file access operation in percent

Return Values

S_OK

Success

E_NOINTERFACE

File with the specified name does not exist in the device

E_FAIL

Failed

Example

The following VB example demonstrates the use of a scroll control for real-time adjustment of the "GainRaw" feature.

```
Private Sub Form_Load()  
ActiveUSB1.Acquire=True  
HScroll11.Value = ActiveUSB1.GetFeature("GainRaw")  
HScroll11.Min = ActiveUSB1.GetFeatureMin("GainRaw")  
HScroll11.Max = ActiveUSB1.GetFeatureMax("GainRaw")  
End Sub  
  
Private Sub HScroll11_Scroll()  
ActiveUSB1.SetFeature("GainRaw", HScroll11.Value)  
End Sub
```


3.2.63 GetFormatList

Description

Returns the array of strings containing the descriptions of pixel formats supported by the currently selected camera.

Syntax

```
[VB]  
Value=objActiveUSB.GetFormatList()
```

```
[C/C++]  
HRESULT GetFormatList( VARIANT* pList );
```

Data Types [VB]

Return value: Variant (SAFEARRAY)

Parameters [C/C++]

pList [out,retval]
Pointer to the SAFEARRAY of strings containing available formats

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example initializes a combo box with the descriptions of available pixel formats and uses it to select a specific format:

```
Private Sub Form_Load()  
FmtLst = ActiveUSB1.GetFormatList  
For i = 0 To UBound(FmtLst)  
Combo1.AddItem (FmtLst(i))  
Next  
Combo1.ListIndex = 0  
ActiveUSB1.Acquire = True  
End Sub  
  
Private Sub Combo1_Click()  
ActiveUSB1.Format = Combo1.ListIndex  
End Sub
```

This MFC example fills out a combo box with available video Formats:

```
VARIANT m_FormatArray=m_ActiveUSB.GetFormatList();  
SAFEARRAY *pArray=m_FormatArray.parray;
```

```
UINT nFormats=pArray->rgsabound[0].cElements;

CString str;
CComboBox *pFormat=(CComboBox*)GetDlgItem(IDC_FORMAT);
for(UINT i=0;i<nFormats;i++)
{
    pFormat->AddString(strFormat);
}
int iFormat=m_ActiveUSB.GetFormat();
pFormat->SetCurSel(iFormat);

SafeArrayDestroy(pArray);
```

Remarks

The format list is built in the acceding order by browsing through all pixel formats supported by the current camera. The index of an element in the Format list can be used as an argument of the [Format](#) property to select a specific pixel format.

Note that in C/C++ applications it is required to call `SafeArrayDestroy()` to delete the SAFEARRAY returned by **GetFormatList**.

3.2.64 GetFPSAcquired

Description

Returns the acquired (displayed) frame rate in frames per second.

Syntax

[VB]

```
Value=objActiveUSB.GetFPSAcquired
```

[C/C++]

```
HRESULT GetFPSAcquired(float* pValue);
```

Data Types [VB]

Return value: Single

Parameters [C/C++]

pValue [out,retval]
Pointer to the fps value

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example displays the acquired frame rate.

```
Private Sub ActiveUSB1_FrameAcquired()  
    Label1.Caption = ActiveUSB1.GetFPSAcquired  
End Sub
```

Remarks

This method returns the frame rate based on the arrival of the video frames into the application buffer. The acquired frame rate depends on the CPU power, performance of the graphic card, color conversion required for a given video mode and custom image processing performed on each frame. The acquired frame rate is equal to the frequency at which the [FrameAcquired](#) event is called in your application. When the acquired frame rate is lower than [camera frame rate](#), *ActiveUSB* will fire the [FrameDropped](#) events.

3.2.65 GetFPS

Description

Returns the actual frame rate of the camera in frames per second.

Syntax

[VB]

```
Value=objActiveUSB.GetFPS
```

[C/C++]

```
HRESULT GetFPS(float* pValue);
```

Data Types [VB]

Return value: Single

Parameters [C/C++]

pValue [out,retval]
Pointer to the fps value

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example displays the actual frame rate of the camera.

```
Private Sub ActiveUSB1_FrameAcquired()  
    Label1.Caption = ActiveUSB1.GetFPS  
End Sub
```

Remarks

This method returns the actual frame rate of the camera. Note that the actual frame rate may differ from the [acquired frame rate](#). *ActiveUSB* calculates the actual frame rate using the [frame timestamps](#). If the camera doesn't support timestamps, the frame rate is calculated based on the arrival of the video packets to the system memory.

3.2.66 GetGainMax

Description

Returns the maximum value allowed for the camera's gain.

Syntax

[VB]

```
Value=objActiveUSB.GetGainMax
```

[C/C++]

```
HRESULT GetGainMax( float* pValue );
```

Data Types [VB]

Return value: Single

Parameters [C/C++]

pValue [out, retval]

Pointer to the maximum value of the feature

Return Values

S_OK

Success

E_NOINTERFACE

Feature does not exist or not available

E_FAIL

Failure to read the value

Example

This VB example uses the minimum and maximum values to initialize the slider control:

```
Private Sub Form_Load()  
Slider1.Min = ActiveUSB1.GetGainMin  
Slider1.Max = ActiveUSB1.GetGainMax  
Slider1.Value = ActiveUSB1.Gain  
End Sub
```

Remarks

Note that this method is available only if the currently selected camera supports one of the following GenICam features: *Gain*, *GainAbs*, *GainRaw*.

3.2.67 GetGainMin

Description

Returns the minimum value allowed for the camera's gain.

Syntax

[VB]

```
Value=objActiveUSB.GetGainMin
```

[C/C++]

```
HRESULT GetGainMin( float* pValue );
```

Data Types [VB]

Return value: Single

Parameters [C/C++]

pValue [out, retval]

Pointer to the minimum value of the feature

Return Values

S_OK

Success

E_NOINTERFACE

Feature does not exist or not available

E_FAIL

Failure to read the value

Example

This VB example uses the minimum and maximum values to initialize the slider control:

```
Private Sub Form_Load()  
Slider1.Min = ActiveUSB1.GetGainMin  
Slider1.Max = ActiveUSB1.GetGainMax  
Slider1.Value = ActiveUSB1.Gain  
End Sub
```

Remarks

Note that this method is available only if the currently selected camera supports one of the following GenICam features: *Gain*, *GainAbs*, *GainRaw*.

3.2.68 GetHistogram

Description

Returns the histogram of the current image frame over a selected [ROI](#) .

Syntax

[VB]

```
Value=objActiveUSB.GetHistogram(Component [,Bins, Step ])
```

[C/C++]

```
HRESULT GetHistogram( VARIANT* pHistogram, short Component, long Bins=256, short Step=1 );
```

Data Types [VB]

Channel : Integer

Bins : Long

Step : Integer

Return value: Variant (Array)

Parameters [C/C++]

Component [in]

Enumerated integer specifying the color component for which the histogram data are obtained.

This parameter is disregarded for grayscale images. Must be one of the following values:

- 0 - collects the histogram of the luminance of the image
- 1 - collects the histogram of the red component of the image
- 2 - collects the histogram of the green component of the image
- 3 - collects the histogram of the blue component of the image

Bins [in]

Number of intervals used for histogram collection. It is recommended to set this parameter to a number of possible intensity levels in the image or to an integer fraction of that value, otherwise some precision will be lost due to averaging that occurs within the consolidated bins. A typical number of histogram bins for 8-bit and 16-bit images will be 256.

Step [in]

An optional integer between 1 and 16 specifying the sampling step, which is used to collect pixel values for the histogram calculations. If this number is 1, every pixel is taking into consideration. If the number is 2, every second pixel in a row will be skipped, as well as every second row in the image, etc. Increasing the sampling step raises the speed of the histogram calculations, but sacrifices some accuracy.

pHistogram [out,retval]

Pointer to the SAFEARRAY containing the histogram values.

Return Values

S_OK

Success

E_FAIL

Failure.

Example

This VB example retrieves the histogram of the red component per each frame acquired:

```
Private Sub ActiveUSB1_FrameAcquired()  
ActiveUSB1.SetROI 100,100,500, 400  
HistArray=ActiveUSB1.GetHistogram (1, 256)  
End Sub
```

Remarks

The histogram is calculated over the rectangular region of interest defined by [SetROI](#). The luminance thresholds of the current ROI are disregarded. If the ROI is not set, the whole image frame will be used.

3.2.69 GetImageData

Description

Returns the two-dimensional array of pixel values in the currently acquired frame.

Syntax

[VB]

```
Value=objActiveUSB.GetImageData
```

[C/C++]

```
HRESULT GetImageData( VARIANT* pArray );
```

Data Types [VB]

Return value: Variant (SAFEARRAY)

Parameters [C/C++]

pArray [out,retval]

Pointer to the SAFEARRAY containing the pixel values in the frame

Return Values

S_OK

Success

E_FAIL

Failure.

Example

This VB example uses the [FrameAcquired](#) event to invert pixel value in the bottom left corner of the current frame and display the processed frame in real time.

```
Private Sub Form_Load()  
ActiveUSB1.Display = False  
ActiveUSB1.Acquire = True  
End Sub  
  
Private Sub ActiveUSB1_FrameAcquired()  
a = ActiveUSB1.GetImageData  
For x = 0 To 200  
For y = 0 To 200  
a(x, y) = 255 - a(x, y)  
Next  
Next  
ActiveUSB1.Draw  
End Sub
```

Remarks

GetImageData does not copy the image data, so the array returned contains the actual image buffer of the currently acquired frame. Modifying elements of the array will change actual pixel values in the image buffer. This can be used to perform custom image processing and display a processed image in real time. Note that this feature cannot be used in .NET as its framework creates a copy of the array returned. For direct access to *ActiveUSB*'s image frame in VB.NET and C# use [GetImagePointer](#).

Images in *ActiveUSB* are stored bottom up, therefore the first element of the array is first pixel of the bottom line of the image. The type of data and dimensions of the array returned by **GetImageData** depends on the output format of the video, its horizontal width [SizeX](#) and the number of lines acquired, as specified in the following table:

Camera Pixel Format	Output Format	Data type	Dimensions
Mono8	8-bit monochrome	Byte	0 to SizeX - 1, 0 to Lines - 1
Mono10, Mono12, Mono16	16-bit gray monochrome	Integer (word)	0 to SizeX - 1, 0 to Lines - 1
YUV411, YUV422, YUV444, RGB8, BGR8, Bayer8	24-bit RGB	Byte	0 to SizeX * 3 - 1, 0 to Lines - 1
RGB10, RGB12, RGB16, BGR10, BGR12, Bayer10, Bayer12, Bayer16	48-bit RGB	Integer (word)	0 to SizeX * 3 - 1, 0 to Lines - 1

For integer (word) type of data you can receive negative numbers if pixel values exceed 32767. In C and C# you can convert signed integers to unsigned ones by using data casting. To get rid of negative values in VB and Delphi, subtract them from 65535.

3.2.70 GetImageLine

Description

Returns the array of pixel values at the specified horizontal line of the currently acquired frame.

Syntax

```
[VB]  
Value=objActiveUSB.GetImageLine( Y )
```

```
[C/C++]  
HRESULT GetImageLine( short Y, VARIANT* pArray );
```

Data Types [VB]

Y: Integer
Return value: Variant (SAFEARRAY)

Parameters [C/C++]

Y [in]
The y-coordinate of the line in the image
pArray [out,retval]
Pointer to the SAFEARRAY containing the pixel values in the line

Return Values

S_OK
Success
E_FAIL
Failure.
E_INVALIDARG
Invalid input argument.

Example

This VB example grabs a frame, retrieves the 33th row of pixels and displays the value of 11th pixel in the row.

```
Dim pix as Long  
ActiveUSB1.Grab  
Line=ActiveUSB1.GetImageLine(32)  
pix=Line(10)  
if pix < 0 then  
pix=65535-pix  
endif  
MsgBox Line(10)
```

Remarks

The array returned by **GetImageLine** is a copy of the actual raw of pixels. Modifying elements of the array will not change actual pixel values in the frame buffer. The type of data and dimension of the array returned by **GetImageLine** depends on the output format of the video as specified in the following table:

Camera Pixel Format	Output Format	Data type	Dimension
Mono8	8-bit monochrome	Byte	0 to SizeX -1
Mono10, Mono12, Mono16	16-bit gray monochrome	Integer (word)	0 to SizeX -1
YUV411, YUV422, YUV444, RGB8, BGR8, Bayer8	24-bit RGB	Byte	0 to SizeX * 3 - 1
RGB10, RGB12, RGB16, BGR10, BGR12, Bayer10, Bayer12, Bayer16	48-bit RGB	Integer (word)	0 to SizeX * 3 - 1

The value of the y coordinate must not exceed the height of the video frame, or the error will occur. For integer (word) type of data you can receive negative numbers if pixel values exceed 32767. In C and C# you can convert signed integers to unsigned ones by using data casting. To get rid of negative values in VB and Delphi, subtract them from 65535 (see example above).

Note that in C/C++ applications it is required to call `SafeArrayDestroy()` to delete the `SAFEARRAY` returned by **GetImageLine**.

3.2.71 GetImagePointer

Description

Returns the pointer to the pixel at the specified coordinates of the current frame.

Syntax

[VB]

```
Value=objActiveUSB.GetImagePointer( X, Y )
```

[C/C++]

```
HRESULT GetImagePointer( short X, short Y, VARIANT* pValue );
```

Data Types [VB]

X: Integer

Y: Integer

Return value: Variant (pointer)

Parameters [C/C++]

X [in]

The x-coordinate of the pixel

Y [in]

The y-coordinate of the pixel

pValue [out,retval]

Pointer to the variant containing the pointer to the specified memory location

Return Values

S_OK

Success

E_FAIL

Failure.

E_INVALIDARG

Invalid input arguments.

Example

This C++ example grabs a frame, retrieves a pointer to the 32th line in the frame memory and copies the pixels values into a byte array . A wrapper class CActiveUSB is used to access the *ActiveUSB* control:

```
BYTE line[4096];  
int iWidth=m_ActiveUSB.GetSizeX;  
int iHeight=m_ActiveUSB.GetSizeY;  
m_ActiveUSB.Grab()  
VARIANT v=m_ActiveUSB.GetImagePointer(0,iHeight-1-32);  
memcpy(&line,v.pvVal,iWidth);
```

This C# example uses the [FrameAcquired](#) event to retrieve a pointer to the 32th line in the frame

memory and change pixel values in the line to zeros:

```
private void axActiveUSB1_FrameAcquired(object sender,
AxActiveUSBLib._IActiveUSBEvents_FrameAcquiredEvent e)
{
    int sx=axActiveUSB1.SizeX;
    int sy=axActiveUSB1.SizeY;
    object obj=axActiveUSB1.GetImagePointer(0, sy-1-32);
    int a = (int)obj;
    byte* ptr= (byte*)a;
    for (int x=0; x<sx; x++, ptr++)
        *ptr=0;
}
```

Remarks

The **GetImagePointer** method provides the most efficient way to quickly access the internal image buffer in pointer-aware programming languages. Unlike [GetImageData](#), this method doesn't modify original pixel values of high-bit depth images. The number of bytes in each line of the image buffer depends on the format and horizontal size of the video as specified in the following table:

Camera Pixel Format	Output Format	Line width in bytes
Mono8	8-bit monochrome	SizeX
Mono10, Mono12, Mono16	16-bit monochrome	SizeX * 2
YUV411, YUV422, YUV444, RGB8, BGR8, Bayer8	24-bit RGB	SizeX * 3
RGB10, RGB12, RGB16, BGR10, BGR12, Bayer10, Bayer12, Bayer16	48-bit RGB	SizeX * 6

Images in *ActiveUSB* are stored bottom up, therefore the zero vertical coordinate corresponds to the bottom line of the image.

The values of the x and y coordinates must not exceed the width and height of the video frame, or the error will occur.

3.2.72 GetImageStat

Description

Returns the array containing statistical data of the current image frame over a selected [ROI](#).

Syntax

[VB]

```
Value=objActiveUSB.GetImageStat( Component [,Step] )
```

[C/C++]

```
HRESULT GetImageStat( VARIANT* pStat, short Component, short Step=1 );
```

Data Types [VB]

Channel : Integer

Bins : Long

Step : Integer

Return value: Variant (Array of Single values)

Parameters [C/C++]

Component [in]

Enumerated integer specifying the color component for which the image statistics is obtained. This parameter is disregarded for grayscale images. Must be one of the following values:

- 0 - collects the statistics of the luminance of the image
- 1 - collects the statistics of the red component of the image
- 2 - collects the statistics of the green component of the image
- 3 - collects the statistics of the blue component of the image

Step [in]

An optional integer between 1 and 16 specifying the sampling step, which is used to collect pixel values for the calculations. If this number is 1, every pixel is taking into consideration. If the number is 2, every second pixel in a row will be skipped, as well as every second row in the image, etc. Increasing the sampling step raises the speed of the image statistics collection, but sacrifices some accuracy.

pStat [out,retval]

Pointer to the SAFEARRAY (float) containing the image statistics. The statistics is written to the array as follows:

- Stat [0] – Mean value
- Stat [1] – Standard deviation
- Stat [2] – Pixel count
- Stat [3] – Minimum
- Stat [4] – Maximum
- Stat [5] – Median
- Stat [6] – Skewness
- Stat [7] – Kurtosis

Return Values

S_OK

Success
E_FAIL
Failure.

Example

This VB example displays the mean value and standard deviation of the red component per each frame acquired:

```
Private Sub ActiveUSB1_FrameAcquired()  
ActiveUSB1.SetROI 100,100,500, 400  
StatArray=ActiveUSB1.GetImageStat 1  
LabelMean.Caption=StatArray[0]  
LabelStd.Caption=StatArray[1]  
End Sub
```

Remarks

The image statistics is calculated over the rectangular region of interest and luminance range defined by [SetROI](#). If the ROI is not set, the whole size and luminance range of the image will be used.

3.2.73 GetImageWindow

Description

Returns the two-dimensional array of pixel values corresponding to the selected window in the currently acquired frame.

Syntax

[VB]

```
Value=objActiveUSB.GetImageWindow (X, Y, Width, Height)
```

[C/C++]

```
HRESULT GetImageWindow( short X, short Y, short Width, short Height,  
VARIANT* pArray );
```

Data Types [VB]

X, Y, Width, Height: Integer

Return value: Variant (SAFEARRAY)

Parameters [C/C++]

X [in], Y[in]

The x- and y-coordinates of the top left pixel of the window in the entire frame

Width [in], Height [in]

The horizontal and vertical size of the window in pixels.

pArray [out,retval]

Pointer to the SAFEARRAY containing the pixel values in the frame

Return Values

S_OK

Success

E_FAIL

Failure.

Example

This VB example uses the [FrameAcquired](#) event to increase the brightness in the central area of the live image:

```
Private Sub Form_Load()  
ActiveUSB1.Display = False  
ActiveUSB1.Acquire = True  
End Sub  
  
Private Sub ActiveUSB1_FrameAcquired()  
xc = ActiveUSB1.SizeX / 2  
yc = ActiveUSB1.SizeY / 2  
w = ActiveUSB1.GetImageWindow(xc - 70, yc - 50, 140, 100)  
For y = 0 To UBound(w, 2)  
For x = 0 To UBound(w, 1)
```

```

pix = w(x, y) + 50
If pix > 255 Then
pix = 255
End If
w(x, y) = pix
Next
Next
ActiveUSB1.SetImageWindow xc - 70, yc - 50, w
ActiveUSB1.Draw
End Sub

```

Remarks

The image data in the array are stored in the standard order from top to bottom, therefore the first element of the array is the top left pixel of the window. This is opposite to the order of rows in the arrays returned by [GetImageData](#). The type of data and dimensions of the array returned by **GetImageWindow** depends on the output format of the video, the width and height of the window and the number of lines acquired, as specified in the following table:

Camera Pixel Format	Output Format	Data type	Dimensions
Mono8	8-bit monochrome	Byte	0 to SizeX - 1, 0 to Lines - 1
Mono10, Mono12, Mono16	16-bit gray monochrome	Integer (word)	0 to SizeX - 1, 0 to Lines - 1
YUV411, YUV422, YUV444, RGB8, BGR8, Bayer8	24-bit RGB	Byte	0 to SizeX * 3 - 1, 0 to Lines - 1
RGB10, RGB12, RGB16, BGR10, BGR12, Bayer10, Bayer12, Bayer16	48-bit RGB	Integer (word)	0 to SizeX * 3 - 1, 0 to Lines - 1

If the dimensions of the window are too large to accommodate the frame size, they will be clipped to the frame boundaries.

For integer (word) type of data you can receive negative numbers if pixel values exceed 32767. In C and C# you can convert signed integers to unsigned ones by using data casting. To get rid of negative values in VB and Delphi, subtract them from 65535.

Note that modifying elements of the array will not change actual pixel values in the frame buffer. Use [SetImageWindow](#) to transfer pixel values into *ActiveUSB*.

Note that in C/C++ applications it is required to call `SafeArrayDestroy()` to delete the `SAFEARRAY` returned by **GetImageWindow**.

3.2.74 GetLevels

Description

Returns the array containing the current levels for the Window/Level operation.

Syntax

[VB]

```
value = objActiveUSB.GetLevels
```

[C/C++]

```
HRESULT SetROI( VARIANT* pLevels );
```

Data Types [VB]

Return value: Variant (Array)

Parameters [C/C++]

pLevels [out,retval]

Pointer to the SAFEARRAY containing the current levels for the Window/Level operation:

ROI [0] - current minR level

ROI [1] - current maxR level

ROI [2] - current minG level

ROI [3] - current maxG level

ROI [4] - current minB level

ROI [5] - current maxB level

Return Values

S_OK

Success

E_FAIL

Failure.

Example

The following VB example displays the currently selected levels for a monochrome image:

```
Levels=ActiveUSB1.GetLevels  
MsgBox "Min: ", Levels[0], "Max: ", Levels[1]
```

Remarks

For more information on the Window/Level parameters see [SetLevels](#).

3.2.75 GetLUT

Description

Returns the array containing the current lookup table.

Syntax

[VB]
`value = objActiveUSB.GetLUT`

[C/C++]
`HRESULT GetLUT(VARIANT* pLUT);`

Data Types [VB]

Return value: Variant (Array)

Parameters [C/C++]

pLUT [out,retval]
Pointer to the SAFEARRAY of floating point values containing the current lookup table.

Return Values

S_OK
Success
E_FAIL
Failure.

Example

The following VB example displays several elements of the current lookup table:

```
LUT=ActiveUSB1.GetLUT
MsgBox "LUT(100): ", LUT(100), "LUT(101): ", LUT(101), "LUT(102): ", LUT
(102)
```

Remarks

For more information on the lookup table refer to [SetLUT](#)

3.2.76 GetOptimalPacketSize

Description

Returns the optimal packet size for the video streaming in the current network configuration.

Syntax

[VB]

```
Value=objActiveUSB.GetOptimalPacketSize
```

[C/C++]

```
HRESULT GetOptimalPacketSize( long* pValue );
```

Data Types [VB]

Return value: long

Parameters [C/C++]

pValue [out,retval]

Pointer to the optimal packet value.

Return Values

S_OK

Success

E_FAIL

Failure

E_NOINTERFACE

Camera does not support packet size negotiation

Example

This VB example finds the optimal packet size and modifies the packet size register of the camera accordingly. If the optimal packet size negotiation failed, the packet size is set to 1500.

```
ActiveUSB1.Camera = 2
MTU=ActiveUSB1.GetOptimalPacketSize
if MTU=0 then
MTU=1500
endif
ActiveUSB1.PacketSize=MTU
```

Remarks

The optimal packet size is determined using the packet negotiation functionality of USB3 Vision cameras. Network packets of different sizes are requested from the camera until the maximum packet size possible for the current configuration is found. If the camera does not support the packet size negotiation, this method will return zero.

The value of the optimal packet size is typically defined by the maximum packet size supported by the network card and camera, whichever is lower. Setting the packet size to the optimal value reduces the CPU load during the video transmission.

negotiation

3.2.77 GetPicture

Description

Returns a Picture object created from the currently acquired frame.

Syntax

[VB]

```
Value=objActiveUSB.GetPicture()
```

[C/C++]

```
HRESULT GetPicture(IPictureDisp* *pPicture);
```

Data Types [VB]

Return value: Picture

Parameters [C/C++]

pPicture [out,retval]

Pointer to the *IPictureDisp* interface object

Return Values

S_OK
Success

E_FAIL
Failure.

Example

This VB example uses the [FrameAcquired](#) event to display a live video in a PictureBox:

```
Private Sub ActiveUSB1_FrameAcquired(ByVal Lines As Integer)
    Picture1.Picture=ActiveUSB1.GetPicture
End Sub
```

This VB.NET example shows how to display a live video in a PictureBox:

```
Private Sub AxActiveUSB1_FrameAcquired(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles AxActiveUSB1.FrameAcquired
    If Not (PictureBox1.Image Is Nothing) Then PictureBox1.Image.Dispose()
    PictureBox1.Image = Bitmap.FromHbitmap(AxActiveUSB1.GetPicture.Handle)
End Sub
```

This C# statement shows how to display an image in a PictureBox:

```
pictureBox1.Image =
Bitmap.FromHbitmap((System.IntPtr)axActiveUSB1.GetPicture().Handle);
```

Remarks

The **GetPicture** method provides the most convenient graphic interface to *ActiveUSB* internal image and allows you to display the last acquired frame in popular graphic controls such as PictureBox. Note that a Picture object contains only 8-bit pixel values, even if the current video mode is a 16- or 48-bit one. To access the actual pixel data, use the [GetImageData](#) or [GetPointer](#) methods.

In .NET programming environment calling **GetPicture** inside the [FrameAcquiredX](#) event handler may not work. Use the [FrameAcquired](#) event instead. It is also necessary to apply the Dispose() method to the PictureBox Image property before each subsequent call to **GetPicture** in order to prevent the memory leak.

3.2.78 GetPixel

Description

Returns the pixel value at the specified coordinates.

Syntax

[VB]

```
Value=objActiveUSB.GetPixel( X, Y )
```

[C/C++]

```
HRESULT GetPixel( short X, short Y, long* pValue );
```

Data Types [VB]

X: Integer

Y: Integer

Return value: Long

Parameters [C/C++]

X [in]

The x-coordinate of the pixel

Y [in]

The y-coordinate of the pixel

pValue [out,retval]

Pointer to the pixel's value

Return Values

S_OK

Success

E_FAIL

Failure.

E_INVALIDARG

Invalid input arguments.

Example

This VB example grabs a frame and displays the value of the pixel at the specified coordinates.

```
ActiveUSB1.Grab  
value=ActiveUSB1.GetPixel(64,32)  
MsgBox value
```

Remarks

The value returned by **GetPixel** depends on the format of the video acquired. For monochrome video the method will retrieve the data from the internal image memory. For the color video the RGB data in the specified coordinates will be converted to the luminance using the formula: $L=(R+G+B) / 3$.

The values of the x and y coordinates must not exceed the width and height of the video frame, or the error will occur.

3.2.79 GetRawData

Description

Returns the two-dimensional array of values in the currently acquired data buffer or pointer to this buffer.

Syntax

```
[VB]  
Value=objActiveUSB.GetRawData ([isPointer=False])
```

```
[C/C++]  
HRESULT GetRawData( bool isPointer, VARIANT* pArray );
```

Data Types [VB]

Return value: Variant (SAFEARRAY)

Parameters [C/C++]

isPointer [in]
If false returns the array of raw data, otherwise returns pointer to data.

pArray [out,retval]
Pointer to the SAFEARRAY containing the raw data buffer.

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example grabs a frame, retrieves the raw data array and displays the value of the specified element of the array.

```
Dim pix as Long  
ActiveUSB1.Grab  
RawData=ActiveUSB1.GetRawData  
x=16  
y=32  
pix=RawData(x,y)  
if pix < 0 then  
pix=65535-pix  
endif  
MsgBox RawData(x,y)
```

Remarks

This function returns the frame data as they arrive from the camera (before being converted to a viewable format, such as RGB). **GetRawData** does not copy the image data, so the array returned contains the actual image buffer of the currently acquired frame. Modifying elements of the array will change actual pixel values in the image buffer. This can be used to perform custom image processing and display a processed image in real time. Note that this feature cannot be used in .NET as its framework creates a copy of the array returned. For direct access to raw data in C# set *isPointer* to True and use the pointer instead of an array.

Raw image data are stored in the standard order from top to bottom, therefore the first element of the array is first pixel of the top line of the image. The type of data and dimensions of the array returned by **GetRawData** depends on the current pixel format, horizontal width [SizeX](#) and the number of lines acquired, as specified in the following table:

Pixel Format	Data type	Dimensions
Mono8, Mono8Signed	Byte	0 to SizeX -1, 0 to Lines - 1
Bayer**8	Byte	0 to SizeX -1, 0 to Lines - 1
Mono10, Mono12, Mono16	Integer (word)	0 to SizeX -1, 0 to Lines - 1
Bayer**10, Bayer**12, Bayer**16	Integer (word)	0 to SizeX -1, 0 to Lines - 1
Mono10Packed, Mono12Packed	Byte	0 to SizeX*3/2 -1, 0 to Lines - 1
YUV411Packed	Byte	0 to SizeX*3/2 -1, 0 to Lines - 1
YUV422Packed, YUV444Packed	Integer (word)	0 to SizeX -1, 0 to Lines - 1
RGB8Packed, BGR8Packed	Byte	0 to SizeX * 3 - 1, 0 to Lines - 1
RGBA8Packed, BGRA8Packed	Byte	0 to SizeX * 4 - 1, 0 to Lines - 1
RGB10Packed, BGR10Packed, RGB12Packed, BGR12Packed	Integer (word)	0 to SizeX * 3 - 1, 0 to Lines - 1
BGR10V1Packed, BGR10V2Packed	Long	0 to SizeX -1, 0 to Lines - 1
RGB8Planar	Byte	0 to SizeX * 3 - 1, 0 to Lines - 1
RGB10Planar, RGB12Planar, RGB16Planar	Integer (word)	0 to SizeX * 3 - 1, 0 to Lines - 1

For integer (word) type of data you can receive negative numbers if pixel values exceed 32767. In C and C# you can convert signed integers to unsigned ones by using data casting. To get rid of negative values in VB and Delphi, subtract them from 65535 (see the example above).

Note that setting both [Display](#) and [Magnification](#) properties to zero will disable an image conversion pipeline that is normally applied to each raw frame. This will significantly increase the performance of your application as long as it does not utilize *ActiveUSB's* built-in image decoding/display.

3.2.80 GetRGBPixel

Description

Returns the array of RGB values at the specified coordinates.

Syntax

[VB]

```
Value=objActiveUSB.GetRGBPixel( X, Y )
```

[C/C++]

```
HRESULT GetRGBPixel( short X, short Y, VARIANT* pArray );
```

Data Types [VB]

X, Y: Integer

Return value: Variant (SAFEARRAY)

Parameters [C/C++]

X [in]

The x-coordinate of the pixel

Y [in]

The y-coordinate of the pixel

pArray [out,retval]

Pointer to the SAFEARRAY of the dimension of 3 containing long R, G and B values of the current pixel

Return Values

S_OK

Success

E_FAIL

Failure.

E_INVALIDARG

Invalid input arguments.

Example

This VB example grabs a frame and displays the value of the G-value of the pixel at the specified coordinates.

```
ActiveUSB1.Grab  
RGB=ActiveUSB1.GetRGBPixel(64,32)  
MsgBox RGB(1)
```

Remarks

The values returned by **GetRGBPixel** depend on the format of the video acquired. For 24-bit video the method will retrieve the data from the internal image memory. For the grayscale video the R, G, and B

values will be the same and equal to the luminance value in the specified coordinates.

The values of the x and y coordinates must not exceed the width and height of the video frame, or the error will occur.

3.2.81 GetROI

Description

Returns the array containing the current ROI settings.

Syntax

[VB]

```
value = objActiveUSB.GetROI
```

[C/C++]

```
HRESULT GetROI( VARIANT* pROI );
```

Data Types [VB]

Return value: Variant (Array)

Parameters [C/C++]

pROI [out,retval]

Pointer to the SAFEARRAY containing the current ROI settings as follows:

ROI [0] - horizontal coordinate of the top left corner of the ROI, relative to the image origin

ROI [1] - vertical coordinate of the top left corner of the ROI, relative to the image origin.

ROI [2] - horizontal coordinate of the bottom right corner of the ROI, relative to the image origin.

ROI [3] - vertical coordinate of the bottom right corner of the ROI, relative to the image origin.

ROI [4] - lower threshold of the luminance range for image statistics calculations

ROI [5] - higher threshold of the luminance range for image statistics calculations

Return Values

S_OK

Success

E_FAIL

Failure.

Example

The following VB example displays the settings of the current ROI:

```
ROI=ActiveUSB1.GetROI  
MsgBox "X1: ", ROI[0], "Y1: ", ROI[1], "X2: ", ROI [2], "Y2: ", ROI [3]
```

Remarks

For more information on the region of interest see [SetROI](#), [GetImageStat](#), [GetHistogram](#).

3.2.82 GetSequenceFrameCount

Description

Returns the current number of frames in the memory sequence.

Syntax

[VB]

```
Value=objActiveUSB.GetSequenceFrameCount()
```

[C/C++]

```
HRESULT GetSequenceFrameCount(long* pValue );
```

Data Types [VB]

Return value: Long

Parameters [C/C++]

pValue [out,retval]

Pointer to the number of frames

Return Values

S_OK

Success

E_FAIL

Failure

Example

This VB example displays the number of frames in the memory sequence.

```
value=ActiveUSB1.GetSequenceFrameCount()  
MsgBox value
```

Remarks

For more information on the sequence capture see [StartSequenceCapture](#).

3.2.83 GetSequencePicture

Description

Returns a Picture object representing a selected frame in the memory sequence.

Syntax

[VB]

```
Value=objActiveUSB.GetSequencePicture( Frame )
```

[C/C++]

```
HRESULT GetSequencePicture(long iFrame, IPictureDisp* *pPicture);
```

Data Types [VB]

Long: Frame

Return value: Picture

Parameters [C/C++]

iFrame

The zero-based index of the frame for which a Picture object will be created

pPicture [out,retval]

Pointer to the *IPictureDisp* interface object

Return Values

S_OK

Success

E_FAIL

Failure

E_INVALIDARG

Invalid frame index

Example

This VB example demonstrated how to display the 15th frame from the memory sequence in a picture box:

```
Picture1.Picture=ActiveUSB1.GetSequencePicture (15)
```

Remarks

The **GetSequencePicture** method provides allows you to display frames from the memory sequence in popular graphic controls such as PictureBox. Note that a Picture object contains only 8-bit pixel values, even if the current video mode is a 16- or 48-bit one. To access the actual pixel data, use the [GetSequenceData](#) or [GetSequencePointer](#) methods.

3.2.84 GetSequencePixel

Description

Returns the pixel value at the specified coordinates.

Syntax

[VB]

```
Value=objActiveUSB.GetSequencePixel( Frame, X, Y )
```

[C/C++]

```
HRESULT GetSequencePixel( long iFrame, short X, short Y, long* pValue );
```

Data Types [VB]

Long: Frame

X: Integer

Y: Integer

Return value: Long

Parameters [C/C++]

iFrame [in]

The zero-based index of the frame in the memory sequence

X [in]

The x-coordinate of the pixel

Y [in]

The y-coordinate of the pixel

pValue [out,retval]

Pointer to the pixel's value

Return Values

S_OK

Success

E_FAIL

Failure.

E_INVALIDARG

Invalid input arguments.

Example

This VB example displays the value of the pixel at the specified coordinates of the memory sequence frame #0.

```
ActiveUSB1.Grab  
value=ActiveUSB1.GetSequencePixel(0,64,32)  
MsgBox value
```

Remarks

The value returned by **GetSequencePixel** depends on the format of the video acquired. For monochrome video the method will return the actual pixel value. For the color video the RGB data in the specified coordinates will be converted to the luminance using the formula: $L=(R+G+B) / 3$.

The values of the x and y coordinates must not exceed the width and height of the video frame, or the error will occur.

3.2.85 GetSequencePointer

Description

Returns the pointer to the pixel at the specified coordinates of the selected frame in the memory sequence.

Syntax

[VB]

```
Value=objActiveUSB.GetSequencePointer( Frame, X, Y )
```

[C/C++]

```
HRESULT GetSequencePointer( long iFrame, short X, short Y, VARIANT* pValue );
```

Data Types [VB]

Frame: Long

X: Integer

Y: Integer

Return value: Variant (pointer)

Parameters [C/C++]

iFrame [in]

The zero-based index of the frame in the memory sequence

X [in]

The x-coordinate of the pixel

Y [in]

The y-coordinate of the pixel

pValue [out,retval]

Pointer to the variant containing the pointer to the specified memory location

Return Values

S_OK

Success

E_FAIL

Failure.

E_INVALIDARG

Invalid input arguments.

Example

This C++ example grabs a frame, retrieves a pointer to the 32th line in the frame # 7 of the memory sequence and copies the pixels values into a byte array . A wrapper class CActiveUSB is used to access the *ActiveUSB* control:

```
BYTE line[4096];  
int iWidth=m_ActiveUSB.GetSizeX;  
int iHeight=m_ActiveUSB.GetSizeY;
```

```
VARIANT v=m_ActiveUSB.GetSequencePointer(7,0,iHeight-1-32);  
memcpy(&line,v.pcVal,iWidth);
```

Remarks

The **GetSequencePointer** method provides the most efficient way to quickly access the memory sequence data in pointer-aware programming languages. The number of bytes in each line of the image buffer depends on the format and horizontal size of the video as specified in the following table:

Camera Pixel Format	Output Format	Line width in bytes
Mono8	8-bit monochrome	SizeX
Mono10, Mono12, Mono16	16-bit monochrome	SizeX * 2
YUV411, YUV422, YUV444, RGB8, BGR8, Bayer8	24-bit RGB	SizeX * 3
RGB10, RGB12, RGB16, BGR10, BGR12, Bayer10, Bayer12, Bayer16	48-bit RGB	SizeX * 6

Note that this method automatically converts the raw sequence data into standard monochrome or RGB output formats. To access the raw image data, use [GetSequenceRawData](#).

Images in *ActiveUSB* are stored bottom up, therefore the zero vertical coordinate corresponds to the bottom line of the image.

The values of the x and y coordinates must not exceed the width and height of the video frame, or the error will occur.

3.2.86 GetSequenceRawData

Description

Returns the two-dimensional array of raw values in the selected frame buffer of the memory sequence or pointer to this buffer.

Syntax

[VB]

```
Value=objActiveUSB.GetSequenceRawData (Frame [, isPointer=False])
```

[C/C++]

```
HRESULT GetSequenceRawData( long iFrame, bool isPointer, VARIANT* pArray );
```

Data Types [VB]

Frame: Long

Return value: Variant (SAFEARRAY)

Parameters [C/C++]

iFrame [in]

The zero-based index of the frame in the memory sequence

isPointer [in]

If false returns the array of raw data, otherwise returns pointer to data.

pArray [out,retval]

Pointer to the SAFEARRAY containing the raw data buffer.

Return Values

S_OK

Success

E_FAIL

Failure

E_INVALIDARG

Invalid input arguments

Example

This VB retrieves the raw data array from frame #4 in the memory sequence and displays the value of the specified element of the array.

```
Dim pix as Long
RawData=ActiveUSB1.GetRawSequenceData
x=16
y=32
pix=RawSequenceData(4,x,y)
if pix < 0 then
pix=65535-pix
endif
```

`MsgBox RawData(x,y)`

Remarks

This function returns the sequence data as they have been recorded (not converted). Raw image data are stored in the standard order from top to bottom, therefore the first element of the array is first pixel of the top line of the image. The type of data and dimensions of the array returned by **GetSequenceRawData** depends on the current pixel format, horizontal width [SizeX](#) and the number of lines acquired, as specified in the following table:

Pixel Format	Data type	Dimensions
Mono8, Mono8Signed	Byte	0 to SizeX - 1, 0 to Lines - 1
Bayer**8	Byte	0 to SizeX - 1, 0 to Lines - 1
Mono10, Mono12, Mono16	Integer (word)	0 to SizeX - 1, 0 to Lines - 1
Bayer**10, Bayer**12, Bayer**16	Integer (word)	0 to SizeX - 1, 0 to Lines - 1
Mono10Packed, Mono12Packed	Byte	0 to SizeX*3/2 - 1, 0 to Lines - 1
YUV411Packed	Byte	0 to SizeX*3/2 - 1, 0 to Lines - 1
YUV422Packed, YUV444Packed	Integer (word)	0 to SizeX - 1, 0 to Lines - 1
RGB8Packed, BGR8Packed	Byte	0 to SizeX * 3 - 1, 0 to Lines - 1
RGBA8Packed, BGRA8Packed	Byte	0 to SizeX * 4 - 1, 0 to Lines - 1
RGB10Packed, BGR10Packed, RGB12Packed, BGR12Packed	Integer (word)	0 to SizeX * 3 - 1, 0 to Lines - 1
BGR10V1Packed, BGR10V2Packed	Long	0 to SizeX - 1, 0 to Lines - 1
RGB8Planar	Byte	0 to SizeX * 3 - 1, 0 to Lines - 1
RGB10Planar, RGB12Planar, RGB16Planar	Integer (word)	0 to SizeX * 3 - 1, 0 to Lines - 1

For integer (word) type of data you can receive negative numbers if pixel values exceed 32767. In C and C# you can convert signed integers to unsigned ones by using data casting. To get rid of negative values in VB and Delphi, subtract them from 65535 (see the example above).

3.2.87 GetSequenceTimestamp

Description

Returns the timestamp of the selected frame in the memory sequence, in seconds.

Syntax

[VB]

```
Value=objActiveUSB.GetSequenceTimestamp (Frame)
```

[C/C++]

```
HRESULT GetSequenceTimestamp(long iFrame, double* pValue);
```

Data Types [VB]

Frame: Long

Return value: Double

Parameters [C/C++]

iFrame [in]

The zero-based index of the frame in the memory sequence

pValue [out,retval]

Pointer to the timestamp value

Return Values

S_OK

Success

E_FAIL

Failure

E_INVALIDARG

Invalid frame index

Example

This VB example displays the timestamp value of each frame of the sequence in a text box.

```
Private Sub ActiveUSB1_FrameLoaded()  
Label1.Caption = ActiveUSB1.GetSequenceTimestamp  
End Sub
```

Remarks

If your USB3 Vision™ camera support the timestamp function, it will mark each frame with a very accurate time value indicating the precise moment at which the frame was acquired by the camera relative to the moment at which the camera was powered up.

If the camera doesn't support timestamps, this method will return the time passed since January 1, 1970.

3.2.88 GetSequenceWindow

Description

Returns the two-dimensional array of pixel values corresponding to the selected window in the frame of the memory sequence.

Syntax

[VB]

```
Value=objActiveUSB.GetSequenceWindow (Frame, X, Y, Width, Height)
```

[C/C++]

```
HRESULT GetSequenceWindow( long iFrame, short X, short Y, short Width,  
short Height, VARIANT* pArray );
```

Data Types [VB]

X, Y, Width, Height: Integer

Return value: Variant (SAFEARRAY)

Parameters [C/C++]

iFrame [in]

The zero-based index of the frame in the memory sequence

X [in], *Y* [in]

The x- and y-coordinates of the top left pixel of the window in the entire frame

Width [in], *Height* [in]

The horizontal and vertical size of the window in pixels.

pArray [out,retval]

Pointer to the SAFEARRAY containing the pixel values in the frame

Return Values

S_OK

Success

E_FAIL

Failure.

Example

This VB example fills a 3D array (x, y, frame) with pixel values from the memory sequence.

```
For z= 0 To ActiveUSB1.GetSequenceFrameCount-1  
sx=ActiveUSB1.SizeX  
sy=ActiveUSB1.SizeY  
a = ActiveUSB1.GetSequenceWindow (z, 0, 0, sx, sy)  
For x = 0 To sx-1  
For y = 0 To sy-1  
M(x,y,z) = a(x, y)  
Next
```

[Next](#)
[Next](#)

Remarks

The image data in the array are stored in the standard order from top to bottom, therefore the first element of the array is the top left pixel of the window. The type of data and dimensions of the array returned by **GetSequenceWindow** depends on the output format of the video, the width and height of the window and the number of lines acquired, as specified in the following table:

Camera Pixel Format	Output Format	Data type	Dimensions
Mono8	8-bit monochrome	Byte	0 to SizeX - 1, 0 to Lines - 1
Mono10, Mono12, Mono16	16-bit gray monochrome	Integer (word)	0 to SizeX - 1, 0 to Lines - 1
YUV411, YUV422, YUV444, RGB8, BGR8, Bayer8	24-bit RGB	Byte	0 to SizeX * 3 - 1, 0 to Lines - 1
RGB10, RGB12, RGB16, BGR10, BGR12, Bayer 10, Bayer 12, Bayer16	48-bit RGB	Integer (word)	0 to SizeX * 3 - 1, 0 to Lines - 1

If the dimensions of the window are too large to accommodate the frame size, they will be clipped to the frame boundaries.

For integer (word) type of data you can receive negative numbers if pixel values exceed 32767. In C and C# you can convert signed integers to unsigned ones by using data casting. To get rid of negative values in VB and Delphi, subtract them from 65535.

Note that in C/C++ applications it is required to call `SafeArrayDestroy()` to delete the `SAFEARRAY` returned by **GetSequenceWindow**.

3.2.89 GetTimestamp

Description

Returns the timestamp of the last acquired frame in seconds.

Syntax

[VB]

```
Value=objActiveUSB.GetTimestamp
```

[C/C++]

```
HRESULT GetTimestamp(double* pValue);
```

Data Types [VB]

Return value: Double

Parameters [C/C++]

pValue [out,retval]

Pointer to the timestamp value

Return Values

S_OK

Success

E_FAIL

Failure.

Example

This VB example displays the timestamp value of each frame in a text box.

```
Private Sub ActiveUSB1_FrameAcquired()  
    Label1.Caption = ActiveUSB1.GetTimestamp  
End Sub
```

Remarks

If your USB3 Vision™ camera support the timestamp function, it will mark each frame with a very accurate time value indicating the precise moment at which the frame was acquired by the camera relative to the moment at which the camera was powered up. Timestamps can be used to imprint a text with the timeline information into images during the AVI capture.

If this method is used in the [FrameAcquired](#) event handler, it must be called immediately after the event has been received. This will guarantee that the value of the timestamp will not be taken from the next frame while the current frame is being processed.

If the camera doesn't support timestamps, this method will return zero.

3.2.90 GetTriggerDelayMax

Description

Returns the maximum value allowed for the camera's trigger delay.

Syntax

```
[VB]  
Value=objActiveUSB.GetTriggerDelayMax
```

```
[C/C++]  
HRESULT GetTriggerDelayMax( float* pValue );
```

Data Types [VB]

Return value: Single

Parameters [C/C++]

pValue [out, retval]
Pointer to the maximum value of the feature

Return Values

S_OK
Success
E_NOINTERFACE
Feature does not exist or not available
E_FAIL
Failure to read the value

Example

This VB example uses the minimum and maximum values to initialize the slider control:

```
Private Sub Form_Load()  
Slider1.Min = ActiveUSB1.GetTriggerDelayMin  
Slider1.Max = ActiveUSB1.GetTriggerDelayMax  
Slider1.Value = ActiveUSB1.TriggerDelay  
End Sub
```

Remarks

Note that this method is available only if the currently selected camera supports one of the following GenICam features: *TriggerDelay*, *TriggerDelayAbs*, *TriggerDelayRaw*.

3.2.91 GetTriggerDelayMin

Description

Returns the minimum value allowed for the camera's trigger delay.

Syntax

```
[VB]  
Value=objActiveUSB.GetTriggerDelayMin
```

```
[C/C++]  
HRESULT GetTriggerDelayMin( float* pValue );
```

Data Types [VB]

Return value: Single

Parameters [C/C++]

pValue [out, retval]
Pointer to the minimum value of the feature

Return Values

S_OK
Success
E_NOINTERFACE
Feature does not exist or not available
E_FAIL
Failure to read the value

Example

This VB example uses the minimum and maximum values to initialize the slider control:

```
Private Sub Form_Load()  
Slider1.Min = ActiveUSB1.GetTriggerDelayMin  
Slider1.Max = ActiveUSB1.GetTriggerDelayMax  
Slider1.Value = ActiveUSB1.TriggerDelay  
End Sub
```

Remarks

Note that this method is available only if the currently selected camera supports one of the following GenICam features: *TriggerDelay*, *TriggerDelayAbs*, *TriggerDelayRaw*.

3.2.92 GetVideoFPS

Description

Returns the playback frame rate of the currently open video file or memory sequence. The default frame rate at which the video was recorded can be changed by calling [SetVideoFPS](#).

Syntax

[VB]

```
Value=objActiveUSB.GetVideoFPS
```

[C/C++]

```
HRESULT GetVideoFPS(float* pValue);
```

Data Types [VB]

Return value: Single

Parameters [C/C++]

pValue [out,retval]
Pointer to the fps value

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example opens an AVI file and displays its frame rate.

```
ActiveUSB1.OpenVideo "c:\\video1.avi"  
MsgBox ActiveUSB1.GetVideoFPS
```

3.2.93 GetVideoFrameCount

Description

Returns the number of frames in the currently [open video](#) file.

Syntax

```
[VB]  
Value=objActiveUSB.GetVideoFrameCount()
```

```
[C/C++]  
HRESULT GetVideoFrameCount(long* pValue );
```

Data Types [VB]

Return value: Long

Parameters [C/C++]

pValue [out,retval]
Pointer to the number of frames

Return Values

S_OK
Success
E_FAIL
Failure

Example

This VB example displays the number of frames in a TIFF video file.

```
ActiveUSB1.OpenVideo "c:\\sequence.tif"  
value=ActiveUSB1.GetVideoFrameCount()  
MsgBox value
```

3.2.94 GetVideoPosition

Description

Returns the zero-based position of the current frame in the open video file or sequence.

Syntax

[VB]
`Value=objActiveUSB.GetVideoPosition`

[C/C++]
`HRESULT GetVideoPosition(long* pFrame);`

Data Types [VB]

Return value: Long

Parameters [C/C++]

pFrame [out,retval]
Pointer to the zero-based index of the current frame.

Return Values

S_OK
Success
E_FAIL
Failure

Example

This VB example uses the [FrameLoaded](#) event to display the number of a currently played frame.

```
Private Sub ActiveUSB1_FrameLoaded()  
frameindex = ActiveUSB1.GetVideoPosition + 1  
LabelCount.Caption = frameindex
```

Remarks

When the video file or sequence has just been opened, the video position is reset to zero.

3.2.95 GetVideoVolume

Description

Returns the volume level of the currently open AVI file.

Syntax

```
[VB]  
Value=objActiveUSB.GetVideoVolume
```

```
[C/C++]  
HRESULT GetVideoVolume(short* pValue );
```

Data Types [VB]

Return value: Integer

Parameters [C/C++]

pValue [out,retval]
Pointer to the current volume, in percent.

Return Values

S_OK
Success
E_FAIL
Failure

Example

This VB example displays the volume of the currently open video file.

```
ActiveUSB.OpenVideo "C:\\\video1.avi", True  
MsgBox ActiveUSB1.GetVideoVolume
```

Remarks

In order for this method to work, the AVI file must be recorded with the sound and [opened](#) with the sound option enabled.

3.2.96 Grab

Description

Grabs a single frame into the internal memory. If the [Display](#) property is enabled, the frame will be automatically displayed in the control window.

Syntax

[VB]
`objActiveUSB.Grab`

[C/C++]
`HRESULT Grab();`

Parameters

None

Return Values

S_OK
Success
E_ABORT
Timeout occurred
E_ACCESSDENIED
Camera not found
E_FAIL
Failure

Example

This VB example grabs a frame and saves it as a tiff file

```
Private Sub Form_Load()  
ActiveUSB1.Acquire=False  
End Sub  
  
Private Sub Button_Click()  
ActiveUSB1.Grab  
ActiveUSB1.SaveImage "image1.tif"  
End Sub
```

Remarks

The **Grab** method will wait for the current frame to be acquired before returning. When the acquisition of the current frame is complete, the [FrameAcquired](#) event will be raised.

Note that the use of the Grab method is incompatible with the [Acquire](#) property set to TRUE. Make sure to set Acquire to FALSE when using **Grab**.

3.2.97 IsFeatureAvailable

Description

Checks the availability of the specified camera feature.

Syntax

[VB]

```
Value=objActiveUSB.IsFeatureAvailable ( Name )
```

[C/C++]

```
HRESULT IsFeatureAvailable( bstr Name, BOOL pValue );
```

Data Types [VB]

Name: String

Return value: String

Parameters [C/C++]

Name [in]

String specifying the name of the feature

pValue [out, retval]

Pointer to the boolean value which is TRUE if the feature is available and FALSE otherwise

Return Values

S_OK

Success

E_FAIL

Failure

Example

This VB example uses the availability of the feature to enable or disable a slider:

```
if IsFeatureAvailable("ExposureTimeAbs") then
Slider1.Enable(TRUE)
else
Slider1.Enable(FALSE)
endif
```

Remarks

A feature is considered available when its access flag is RO or RW. See [GetFeatureAccess](#) for more details.

3.2.98 LoadImage

Description

Loads the image from the specified file into the internal frame buffer and displays it in *ActiveUSB* window. The method supports BMP, TIFF and JPEG formats.

Syntax

[VB]
`objActiveUSB.LoadImage File`

[C/C++]
`HRESULT LoadImage(bstr File);`

Data Types [VB]

File: String

Parameters [C/C++]

File [in]
The string containing the file name and path

Return Values

S_OK
Success
E_FAIL
Failure.
E_INVALIDARG
Invalid file name.

Example

This VB example shows how to load an image file.

```
ActiveUSB1.LoadImage "C:\\myframe.jpg"
```

Remarks

When the image has been loaded into the internal buffer, the [FrameLoaded](#) event will be fired.

The image format is defined by the file extension indicated in the *File* argument. Use "bmp" for a BMP file, "tif" for TIFF, and "jpg" for JPEG. If none of these extensions are found in the *File* string, an error will occur.

3.2.99 LoadSettings

Description

Loads previously stored camera settings from the data file.

Syntax

```
[VB]  
objActiveUSB.LoadSettings( Name )
```

```
[C/C++]  
HRESULT LoadSettings( BSTR Name );
```

Data Types [VB]

Profile: String

Return value: None

Parameters [C/C++]

Name [in]

The name of the profile or path to the file in which the settings are stored.

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example restores previously saved camera settings:

This VB example shows how to restore previously saved camera settings using two options - profile name and file path:

```
Private Sub LoadSettings_Profile()  
    ActiveUSB1.LoadSettings ("FluorescentHighRes")  
End Sub  
  
Private Sub LoadSettings_File()  
    ActiveUSB1.LoadSettings ("C:/CamSettings/HighRes.dat")  
End Sub
```

Remarks

If no colon or slash symbols are found in the Name argument, *ActiveUSB* will treat it as a profile name and will attempt to find the profile with the camera settings in the Profile subfolder of the ActiveUSB folder. Otherwise the exact path to the data file will be used.

This method verifies if the currently selected camera model coincides with the model for which the settings were saved. If the model is different, the function will return an error.

Also see [SaveSettings](#).

3.2.100 LoadSequence

Description

Loads a fragment of the specified AVI or TIFF file into the memory sequence.

Syntax

```
[VB]  
objActiveUSB.LoadSequence File [,Start, End ]
```

```
[C/C++]  
HRESULT LoadSequence( bstr File, long Start, long End );
```

Data Types [VB]

File: String

Start: Long

End: Long

Parameters [C/C++]

File [in]

The string containing the file name and path. The extension of the file must be "avi" or "tif".

Start [in]

The zero-based index of the first frame of the video fragment to be loaded. If omitted, frame 0 will be used.

End [in]

The zero-based index of the last frame of the video fragment to be loaded. If omitted, the last frame will be used.

Return Values

S_OK

Success

E_FAIL

Failure.

E_INVALIDARG

Invalid file name or file not found.

Example

This VB example loads a video sequence from a TIFF file into the memory and plays it.

```
ActiveUSB1.LoadSequence "video1.tif"  
ActiveUSB1.OpenVideo "ram"  
ActiveUSB1.PlayVideo
```

Remarks

If the system memory doesn't have enough capacity to accommodate the specified video fragment, the video will be truncated to fit the maximum memory size available for the application.

3.2.101 OpenVideo

Description

Opens a specified video file (AVI or TIF) or previously recorded memory sequence for a playback. Used in combination with [PlayVideo](#), [StopVideo](#), [CloseVideo](#).

Syntax

[VB]
`objActiveUSB.OpenVideo Source [, Sound = False]`

[C/C++]
`HRESULT OpenVideo (bstr Source, bool Sound);`

Data Types [VB]

Source : String
Sound : Boolean (optional)

Parameters [C/C++]

File [in]
The string containing the path to the AVI or TIF file; or "RAM" for a memory sequence.
Sound [in]
Optional boolean parameter for AVI files. If TRUE, the playback will be performed with the sound.

Return Values

S_OK
Success
E_FAIL
Failure.
E_INVALIDARG
Invalid file name.

Example

This VB example opens and plays a memory sequence.

```
Private Sub Play_Click()  
ActiveUSB1.OpenVideo "ram"  
ActiveUSB1.PlayVideo  
End Sub
```

Remarks

For the *Sound* option to work, the AVI must contain the sound track.

If an AVI file is open with the *Sound* enabled, its maximum playback frame rate will be limited to x 16 of the recorded frame rate, and the synchronous playback will not be available.

Only one video file or memory sequence can be open at a time by one *ActiveUSB* object. To play back several video sources, use several *ActiveUSB* objects.

3.2.102 OverlayClear

Description

Clears graphics and text from the overlay.

Syntax

[VB]
`objActiveUSB.OverlayClear`

[C/C++]
`HRESULT OverlayClear();`

Parameters

None

Return Values

S_OK
Success
E_FAIL
Failure.

Example

The following VB example moves a red rectangle over the live image by repeatedly erasing and drawing it:

```
Private Sub Form_Load()  
x = 0  
ActiveUSB1.Acquire = True  
ActiveUSB1.OverlayColor = RGB(255, 0, 0)  
ActiveUSB1.Overlay = True  
End Sub  
  
Dim x As Integer  
Private Sub ActiveUSB1_FrameAcquired()  
ActiveUSB1.OverlayClear  
ActiveUSB1.OverlayRectangle x, 10, x + 50, 80, 3  
x = x + 2  
If x = ActiveUSB1.SizeX Then  
x = 0  
End If  
End Sub
```

Remarks

To create animation effects, use this method in combination with [OverlayColor](#), [OverlayPixel](#), [OverlayLine](#), [OverlayRectangle](#), [OverlayEllipse](#), [OverlayText](#).

3.2.103 OverlayEllipse

Description

Draws an empty or filled ellipse in the overlay.

Syntax

[VB]

```
objActiveUSB.OverlayEllipse StartX, StartY, EndX, EndY [,Width = 1]
```

[C/C++]

```
HRESULT OverlayEllipse( short StartX, short StartY, short EndX, short EndY,  
short Width);
```

Data Types [VB]

StartX, StartY, EndX, EndY: Integer

Width: Integer (optional)

Parameters [C/C++]

StartX [in], *StartY* [in]

Pixel coordinates of the top left corner of the ellipse's bounding rectangle, relative to the image origin.

EndX [in], *EndY* [in]

Pixel coordinates of the bottom right corner of the ellipse's bounding rectangle, relative to the image origin.

Width [in]

Width of the outline of the ellipse in pixels. If zero, a filled ellipse is drawn.

Return Values

S_OK

Success

E_FAIL

Failure.

Example

The following VB example overlays a filled red ellipse on the live video:

```
ActiveUSB1.Acquire = True  
ActiveUSB1.OverlayEllipse 100,100,200,200,0  
ActiveUSB1.OverlayColor=RGB(255,0,0)  
ActiveUSB1.Overlay= True
```

Remarks

To draw a filled ellipse, use *Width=0*. Also see [OverlayColor](#), [OverlayClear](#).

3.2.104 OverlayLine

Description

Draws a line in the overlay.

Syntax

[VB]

```
objActiveUSB.OverlayLine StartX, StartY, EndX, EndY [,Width = 1]
```

[C/C++]

```
HRESULT OverlayLine( short StartX, short StartY, short EndX, short EndY,  
short Width);
```

Data Types [VB]

StartX, StartY, EndX, EndY: Integer

Width: Integer (optional)

Parameters [C/C++]

StartX [in], *StartY* [in]

Pixel coordinates of the starting point of the line, relative to the image origin.

EndX [in], *EndY* [in]

Pixel coordinates of the end point of the line, relative to the image origin.

Width [in]

Width of the line in pixels.

Return Values

S_OK

Success

E_FAIL

Failure.

Example

The following VB example overlays a filled red line of width 3 on the live video:

```
ActiveUSB1.Acquire = True  
ActiveUSB1.OverlayLine 100,100,200,200,3  
ActiveUSB1.OverlayColor=RGB(255,0,0)  
ActiveUSB1.Overlay= True
```

Remarks

To draw multiple lines, call this method several times. Also see [OverlayColor](#), [OverlayClear](#).

3.2.105 OverlayPixel

Description

Draws a pixel in the overlay.

Syntax

```
[VB]  
objActiveUSB.OverlayPixel X, Y
```

```
[C/C++]  
HRESULT OverlayPixel( short X, short Y );
```

Data Types [VB]

X, Y: Integer

Parameters [C/C++]

X [in], Y [in]
Coordinates of the pixel relative to the image origin.

Return Values

S_OK
Success
E_FAIL
Failure.

Example

The following VB example overlays four red pixels on the live video:

```
ActiveUSB1.Acquire = True  
ActiveUSB1.OverlayPixel 100,100  
ActiveUSB1.OverlayPixel 100,200  
ActiveUSB1.OverlayPixel 200,100  
ActiveUSB1.OverlayPixel 200,200  
ActiveUSB1.OverlayColor=RGB(255,0,0)  
ActiveUSB1.Overlay= True
```

Remarks

To draw custom shapes, call this method multiple times. Also see [OverlayColor](#), [OverlayClear](#).

3.2.106 OverlayRectangle

Description

Draws an empty or filled rectangle in the overlay.

Syntax

[VB]

```
objActiveUSB.OverlayRectangle StartX, StartY, EndX, EndY [,Width = 1]
```

[C/C++]

```
HRESULT OverlayRectangle( short StartX, short StartY, short EndX, short EndY, short Width);
```

Data Types [VB]

StartX, StartY, EndX, EndY: Integer

Width: Integer (optional)

Parameters [C/C++]

StartX [in], *StartY* [in]

Pixel coordinates of the top left corner of the rectangle, relative to the image origin.

EndX [in], *EndY* [in]

Pixel coordinates of the bottom right corner of the rectangle, relative to the image origin.

Width [in]

Width of the outline of the rectangle in pixels. If zero, a filled rectangle is drawn.

Return Values

S_OK

Success

E_FAIL

Failure.

Example

The following VB example overlays a filled red rectangle on the live video:

```
ActiveUSB1.Acquire = True  
ActiveUSB1.OverlayRectangle 100,100,200,200,0  
ActiveUSB1.OverlayColor=RGB(255,0,0)  
ActiveUSB1.Overlay= True
```

Remarks

To draw a filled rectangle, use *Width=0*. To draw multiple rectangles, call this method several times. Also see [OverlayColor](#), [OverlayClear](#).

3.2.107 OverlayText

Description

Draws a string of text in the overlay.

Syntax

```
[VB]  
objActiveUSB.OverlayText X, Y, Text
```

```
[C/C++]  
HRESULT OverlayText( short X, short Y, bstr Text );
```

Data Types [VB]

X, Y: Integer

Text: String

Parameters [C/C++]

X [in], Y [in]
Coordinates of the text relative to the image origin.

Text [in]
The string containing the text to be drawn.

Return Values

S_OK
Success
E_FAIL
Failure.

Example

The following VB example overlays a string of text on the live video:

```
Dim Font As New StdFont  
Font.Name = "Arial"  
Font.Size = 18  
Font.Bold = True  
ActiveUSB1.OverlayFont = Font  
ActiveUSB1.OverlayText 10, 100, "ActiveUSB rules!"  
ActiveUSB1.OverlayColor = RGB(255, 0, 0)  
ActiveUSB1.Overlay = True
```

Remarks

To select the font for drawing text strings, use [OverlayFont](#). To draw multiple strings, call this method several times. Also see [OverlayColor](#), [OverlayClear](#).

3.2.108 PlayVideo

Description

Plays back the currently open video file (AVI or TIF) or memory sequence in the *ActiveUSB* window.

Syntax

```
[VB]  
objActiveUSB.PlayVideo [ Start, End, Increment ]
```

```
[C/C++]  
HRESULT PlayVideo( long Start, long End, long Increment );
```

Data Types [VB]

Start: Long

End: Long

Increment: Long

Parameters [C/C++]

Start [in]

The zero-based index of the frame of the video file or sequence at which the playback will start. If omitted, the whole file or sequence will be played forward. If -1, the whole file or sequence will be played backward.

End [in]

The zero-based index of the frame at which the playback will stop. If omitted, the playback will stop at the last frame. If *End* < *Start*, the specified fragment will be played backward.

Increment [in]

The increment of frames during the playback. If 2, every second frame will be played. If 3, every third frame will be played, and so on. If 0, 1 or omitted, the playback of every frame will occur.

Return Values

S_OK

Success

E_FAIL

Failure.

Example

This VB example opens an AVI file and plays its fragment while updating the frame count.

```
Private Sub Play_Click()  
ActiveUSB1.OpenVideo "C:\\video1.avi"  
ActiveUSB1.PlayVideo 0, 50  
End Sub
```

```
Dim frameplayed As Integer
```

```
Private Sub ActiveUSB1_FrameLoaded()  
frameplayed = frameplayed + 1  
LabelCount.Caption = frameplayed
```

End Sub

```
Private Sub ActiveUSB1_PlayCompleted (ByVal Frames As Long)
ActiveUSB1.CloseVideo
End Sub
```

Remarks

The video will be played at the frame rate at which it was recorded unless the [SetVideoFPS](#) method is applied.

If the video file was recorded with compression, a corresponding codec must be installed on the system in order to play it back. Note that the playback frame rate of a compressed video file can become significantly reduced if the video is played backward or the *Increment* parameter used.

Per each frame of the video file or sequence played, the [FrameLoaded](#) event will be fired. When the playback is finished, the [PlayCompleted](#) event will be fired.

During the playback frames are being extracted from the video file or memory sequence and loaded into the internal *ActiveUSB* buffer. The pixels of the currently loaded frame can be accessed via *ActiveUSB* image access functions ([GetImageData](#), [GetImagePointer](#), [GetImageWindow](#) etc).

During the playback of a memory sequence all current conversion properties (such as [Bayer](#), [WindowLevel](#), [BkgCorrect](#), [Rotate](#)) will be applied to the frames of the sequence in real time.

Calling this method will automatically turn off the live video by setting the [Acquire](#) property to FALSE.

3.2.109 ReadBlock

Description

Reads the block of data from the internal camera memory starting from the specified bootstrap address.

Syntax

[VB]

```
objActiveUSB.ReadBlock Offset, Buffer, nBytes
```

[C/C++]

```
HRESULT ReadBlock( long Offset, Variant Buffer, long nBytes );
```

Data Types [VB]

Offset: Long

Buffer: Variant (pointer)

nBytes: Long

Parameters [C/C++]

Offset [in]

The 32-bit offset into the camera register address space.

Buffer [in]

Variant containing pointer to a buffer that will receive the data read from the specified offset of the camera memory.

nBytes [in]

The number of bytes to read from the specified offset

Return Values

S_OK

Success

E_FAIL

Failure.

Example

This C++ example reads a block of 128 bytes from a specified offset in the camera address space:

```
unsigned char buffer[128]  
VARIANT v;  
v.pvVal=buffer;  
ActiveUSB.ReadBlock(0xA0000,v,128);
```

Remarks

Using this method provides a much faster access to the internal camera memory than repetitive calls to [ReadRegister](#).

If the specified offset does not exist or the camera cannot accommodate the size of the data block, this method will return an error.

3.2.110 ReadFile

Description

Transfers the indicated amount of bytes from the specified file in the device to the data array in memory.

Syntax

[VB]

```
Value=objActiveUSB.ReadFile Name, Buffer, Length
```

[C/C++]

```
HRESULT ReadFile( bstr Name, long Offset, long Length , VARIANT* pData);
```

Data Types [VB]

Name: String

Offset: Long

Length: Long

Return value: Variant (Array of Bytes)

Parameters [C/C++]

Name [in]

String specifying the name of the file in the device

Offset [in]

The transfer starting position from the beginning of the file in bytes

Length [in]

The number of bytes to transfer from the specified file

pData [out]

Pointer to SAFEARRAY of bytes containing data transferred from the file

Return Values

S_OK

Success

E_NOINTERFACE

File with the specified name does not exist in the device

E_FAIL

Failure.

Example

This VB code transfers data from the "UserSet1" file in the device to the DataArray in memory.

```
length=ActiveUSB1.GetFileSize("UserSet1")  
DataArray=ActiveUSB1.ReadFile("UserSet1",0,length)
```

Remarks

The File Access functionality allows an application to read and write files hosted on the device. The data in those files may contain look-up tables, configuration sets, firmware, and other information.

Once you transferred the file data from the device to memory, you can save them in a file on a hard drive. Note that in order for this method to work, the [Access Mode](#) of the file should be "R" or "RW".

3.2.111 ReadRegister

Description

Reads a 32-bit integer value from the specified bootstrap register of the current camera.

Syntax

[VB]

```
Value=objActiveUSB.ReadRegister( Reg )
```

[C/C++]

```
HRESULT ReadRegister( long Reg, long* pValue );
```

Data Types [VB]

Reg: Long

Return value: Long

Parameters [C/C++]

Reg [in]

The 32-bit offset of the register in the camera's address space

pValue [out,retval]

Pointer to the registers's 32-bit value

Return Values

S_OK

Success

E_FAIL

Failure.

Examples

This C++ example reads the value of the heartbeat timeout from the corresponding USB3 Vision™ register :

```
value=ActiveUSB.ReadRegister(0x00D8);
```

This VB example reads the value of the custom camera feature:

```
value=ActiveUSB.ReadRegister(&HA080)
```

Remarks

The list of standard bootstrap registers can be found in "*USB3 Vision Camera Interface Standard For Machine Vision*" published by the Automated Imaging Association.

This method can be used to access custom camera attributes not available through USB3 Vision™ feature interface. Refer to the camera documentation for the list of the manufacturer-specific registers.

3.2.112 SaveBkg

Description

Stores a dark or bright background image on the hard drive. The background image is produced as a result of temporal frame averaging.

Syntax

```
[VB]  
objActiveUSB.SaveBkg Type [, Frames = 8 ]
```

```
[C/C++]  
HRESULT SaveBkg( short Type, short Frames );
```

Data Types [VB]

Type: Integer
Frames: Integer

Parameters [C/C++]

Type [in]
Enumerated integer indicating the type of background to be saved: 1 - Dark Field, 2 - Bright Field
Frames [in]
Number of consecutive frames to be averaged for background calculation.

Return Values

S_OK
Success
E_FAIL
Failure.

Example

The following VB example uses two buttons to save dark and bright background images:

```
Private Sub SaveDark_Click  
ActiveUSB1.SaveBkg (1)  
End Sub
```

```
Private Sub SaveBright_Click  
ActiveUSB1.SaveBkg (2)  
End Sub
```

Remarks

In general, to perform the background correction you have to prepare two auxiliary images: dark field and bright field. Dark field is a background image captured with no light transmitted through the camera lens; it is a measure of the dark current in the CCD. The bright field is a background image captured with the maximum light transmitted and no objects in the field of view; it is a measure of the difference in pixel-to-pixel sensitivity as well as the uniformity of illumination. While capturing the bright

field image you should adjust the light intensity so that it stays just below the saturation level of the camera. That will provide the maximum dynamic range for the image acquisition. To control the saturation for monochrome cameras, use the **Saturated Palette**. For more information on the background correction see [BkgCorrect](#).

Background data are stored as raw image files in the Bkgnd subfolder of ActiveUSB program directory. See [BkgName](#) for more details.

When the background saving is complete, the [FrameRecorded](#) event is fired.

Note that this function will only work if the [Acquire](#) property is set to True or the [Grab](#) method is repeatedly called.

3.2.113 SaveImage

Description

Saves the current frame buffer in the specified image file. The method supports RAW, BMP, TIFF and JPEG formats with a selectable compression.

Syntax

[VB]
`objActiveUSB.SaveImage File [, Compression]`

[C/C++]
`HRESULT SaveImage(bstr File, long Compression);`

Data Types [VB]

File: String
Compression: Integer (optional)

Parameters [C/C++]

File [in]
The string containing the file name and path
Compression [in]
The file compression ratio

Return Values

S_OK
Success
E_FAIL
Failure.
E_INVALIDARG
Invalid file name.

Example

This VB example saves the current frame in a JPEG file with the specified compression quality.

```
ActiveUSB1.SaveImage "C:\\myframe.jpg", 75
```

Remarks

The image file format in which the frame will be saved is defined by the file extension indicated in the *File* argument. Use "raw" for a RAW file, "bmp" for a BMP file, "tif" for TIFF, "jpg" for JPEG and "dpx" for DPX. If none of these extensions are found in the *File* string, an error will occur.

When the image saving is complete, the [FrameRecorded](#) event is fired.

The way the *Compression* argument is treated depends on the file format.

BMP files:

Compression = 0 - no compression

Compression = 1 - RLE compression (not implemented in this version)

TIFF files:

Compression = 0 - no compression

Compression = 1 - LZW compression

Compression = 2 - PackBits compression

JPEG files:

Compression is an integer value in the range 0-100 specifying the quality of the image. Lower values correspond to a lower quality with a higher compression, while higher values correspond to a higher quality with a lower compression.

DPX files:

Compression = 0 - RGB order

Compression = 1 - BRG order

RAW files:

Compression has no effect.

If *Compression* parameter is omitted, BMP and TIFF files will be recorded with no compression, while JPEG files will be recorded with quality of 75. DPX files are recorded in the 10-bit packed RGB or monochrome format.

When the RAW format is selected, the resulting file will contain undecoded frame data as they arrived from the camera. See [GetRawData](#) for more details.

3.2.114 SaveSettings

Description

Stores the current camera settings in a data file.

Syntax

```
[VB]  
objActiveUSB.SaveSettings( Name )
```

```
[C/C++]  
HRESULT SaveSettings( BSTR Name );
```

Data Types [VB]

Profile: String

Return value: None

Parameters [C/C++]

Name [in]

The name of the profile or path to the file in which the settings will be saved.

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example shows how to save the current camera settings using two options - profile name and file path:

```
Private Sub SaveSettings_Profile()  
    ActiveUSB1.SaveSettings ("FluorescentHighRes")  
End Sub  
  
Private Sub SaveSettings_File()  
    ActiveUSB1.SaveSettings ("C:/CamSettings/HighRes.dat")  
End Sub
```

Remarks

If no colon or slash symbols are found in the Name argument, *ActiveUSB* will treat it as a profile name and will attempt to save the camera settings in the Profile subfolder of the ActiveUSB folder. Otherwise the settings will be saved using the exact path to the file.

Note that due to User Account Control (UAC) your application may not have a permission to write data into the Program File folder where the ActiveUSB/Profile subfolder is located. In this case you should

create a folder that has both read and write permissions and store the camera settings in it by using the full path and file name as an argument.

Also see [LoadSettings](#).

3.2.115 SaveSequence

Description

Saves the current memory sequence or its fragment in the specified AVI or TIFF file.

Syntax

[VB]

```
objActiveUSB.SaveSequence File [, Start, End, FPS ]
```

[C/C++]

```
HRESULT SaveSequence( bstr File, long Start, long End, float FPS );
```

Data Types [VB]

File: String

Start: Long

End: Long

FPS: Single

Parameters [C/C++]

File [in]

The string containing the file name and path. The extension of the file must be "avi" or "tif".

Start [in]

The zero-based index of the first frame of the fragment of the sequence to be saved. If omitted, frame 0 will be used.

End [in]

The zero-based index of the fragment of the sequence to be saved. If omitted, the last frame will be used.

FPS [in]

The frame rate to be assigned to the saved video. If zero or omitted, the recorded frame rate will be used.

Return Values

S_OK

Success

E_FAIL

Failure.

E_INVALIDARG

Invalid file name.

Example

This VB example records 1000 frames into the memory sequence and uses the [CaptureCompleted](#) event to save it in an AVI file.

```
Private Sub StartButton_Click()  
ActiveUSB1.StartSequenceCapture 1000  
End Sub
```

```
Private Sub ActiveUSB1_CaptureCompleted(ByVal Frames As Long)
SaveSequence "C:\\video.avi"
End Sub
```

Remarks

If the sequence is saved in the AVI format, the currently selected [compression codec](#) will be used.

If the external device (typically, a hard drive or memory card) doesn't have enough space to store the whole sequence, it will be truncated to accommodate the remaining space.

3.2.116 SetAudioLevel

Description

Sets the recording level of the currently selected audio device.

Syntax

```
[VB]  
objActiveUSB.SetAudioLevel Value
```

```
[C/C++]  
HRESULT SetAudioLevel(short Value);
```

Data Types [VB]

Value: Integer

Parameters [C/C++]

Value [in]
The recording level to be set, in percent.

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example initiates an AVI recording with a sound track and uses a scroll bar to adjust the audio recording level.

```
Private Sub Form_Load()  
ActiveUSB1.SetAudioSource 0  
HScroll1.Min=0  
HScroll1.Max=100  
HScroll.Value=ActiveUSB1.GetAudioLevel  
End Sub  
  
Private Sub Capture_Click  
ActiveUSB1.StartCapture "c:\\capture_with_sound.avi"  
End Sub  
  
Private Sub HScroll1_Scroll()  
ActiveUSB1.SetAudioLevel HScroll1.Value  
End Sub
```

Remarks

In order for this method to work, the audio recording device must be selected with [SetAudioSource](#).

3.2.117 SetAudioSource

Description

Sets the index of the audio recording device to be used during the AVI capture.

Syntax

```
[VB]  
objActiveUSB.SetAudioSource Index
```

```
[C/C++]  
HRESULT SetAudioSource(short Index);
```

Data Types [VB]

Source: Integer

Parameters [C/C++]

Index [in]
Zero-based index of the selected audio source in the system list of audio recording devices. If -1 (default), no audio track will be recorded.

Return Values

S_OK
Success
E_FAIL
Failure to set the audio source

Example

This VB example initializes a combo box with the descriptions of available audio recording devices and uses it to select a specific device:

```
AudioLst = ActiveUSB1.GetAudioList  
For i = 0 To UBound(AudioLst)  
    Combo1.AddItem (AudioLst(i))  
Next  
Combo1.ListIndex = 0  
ActiveUSB1.SetAudioSource 0  
  
Private Sub Combo1_Click()  
    ActiveUSB1.SetAudioSource Combo1.ListIndex  
End Sub  
  
Private Sub Capture_Click  
    ActiveUSB1.StartCapture "c:\\capture_with_sound.avi"  
End Sub
```

Remarks

This method allows you to add the audio track to your AVI file and select a specific audio recording device. If only one recording device (such as a microphone) is present in the system, use 0 as the value of *Index*. To disable the audio recording, call **SetAudioSource** with the value of *Index* -1. Note that by default the audio recording is disabled in *ActiveUSB*.

To initiate the AVI recording, use [StartCapture](#).

3.2.118 SetCodec

Description

Sets the name of the codec (video compressor) to be used during the AVI capture.

Syntax

```
[VB]  
objActiveUSB.SetCodec Name
```

```
[C/C++]  
HRESULT SetCodec(bstr Name);
```

Data Types [VB]

Name: String

Parameters [C/C++]

Name [in]
String specifying the name of the feature

Return Values

S_OK
Success
E_FAIL
Failure to set the codec

Example

The following VB example demonstrates the use of the DivX codec:

```
Private Sub Form_Load()  
ActiveUSB1.SetCodec "DivX 5.0.2 Codec"  
ActiveUSB1.Acquire=True  
End Sub  
  
Private Sub StartButton_Click()  
ActiveUSB1.StartCapture "c:\mycapture.avi"  
End Sub  
  
Private Sub StopButton_Click()  
ActiveUSB1.StopCapture  
End Sub
```

The following VB example demonstrates how to set up a codec by its index in the system:

```
CodecList=ActiveUSB1.GetCodecList  
CodecName=CodecList(9)  
ActiveUSB1.SetCodec CodecName
```

Remarks

The name of the codec must be precise in order for **SetCodec** to work. An extra space in the name of the codec may cause this function to fail. To avoid errors, it is recommended to use an index of the codec rather than its name, as shown in the second example above.

ActiveUSB video recording engine includes a proprietary "Raw Uncompressed" codec. When the camera streams video in the Bayer format and the "Raw Uncompressed" codec is selected, *ActiveUSB* records the video in its original raw format while displaying it in color. This reduces the disk space requirements by three times without any degradation in the image quality. In addition, when high-depth pixel formats are used (such as Bayer10, Bayer12, Bayer16, Mono10, Mono 12, Mono16), the recorded video will be packed into the 12-bit per pixel format as opposed to 16 bit, thus saving extra 25% of the disk space. To decode and play back AVI files recorded with the "Raw Uncompressed" codec, use video playback methods of the DVR version of *ActiveUSB*.

3.2.119 SetCodecProperties

Description

Sets the generic parameters of the currently selected compression codec.

Syntax

[VB]

```
objActiveUSB.SetCodecProperties Quality [, DataRate, KeyFrameRate,  
PFramesPerKey, WindowSize ]
```

[C/C++]

```
HRESULT SetCodecProperties(long Quality = -1 [, long DataRate = -1, long  
KeyFrameRate = -1,  
long PFramesPerKey = -1, long WindowSize = -1 ]);
```

Data Types [VB]

Quality: Integer

DataRate: Integer

KeyFrameRate: Integer

PFramesPerKey: Integer

WindowSize: Integer

Parameters [C/C++]

Quality [in]

Numerical value of the quality parameter used by the codec, if supported.

DataRate [in]

Data rate in kb/sec used by the codec, if supported.

KeyFrameRate [in]

Amount of frames after which a key frame is recorded, if supported.

PFramesPerKey [in]

Rate of predicted (P) frames per key frame, if supported.

WindowSize [in]

Amount of frames over which the compressor maintains the average data rate.

Return Values

S_OK

Success

E_FAIL

Failure to set the codec's properties

Example

The following VB example sets the quality and datarate settings of the MJPEG codec:

```
ActiveUSB1.SetCodec "MJPEG Compressor"  
ActiveUSB1.SetCodecProperties 50, 1000
```

Remarks

This method allows you to set only the basic compression properties which may not be supported by certain codecs. To adjust the internal parameters of a codec, use [ShowCodecDialog](#).

If some parameters of this methods are omitted or set to -1, corresponding properties of the compression codec will not be changed.

3.2.120 SetColorMatrix

Description

Sets the matrix coefficients for the color correction operation. Used in combination with the [ColorCorrect](#) property.

Syntax

[VB]

```
objActiveUSB.SetColorMatrix rr, rg, rb, gr, gg, gb, br, bg, bb
```

[C/C++]

```
HRESULT SetColorMatrix (float rr, float rg, float rb,  
                        float gr, float gg, float gb,  
                        float br, float bg, float bb);
```

Data Types [VB]

rr, rg, rb, gr, gg, gb, br, bg, bb: Single

Parameters [C/C++]

rr[in], rg[in], rb[in], gr[in], gg[in], gb[in], br[in], bg[in], bb[in]
Coefficients of the color correction matrix. Must be in the range from -2.0 to +2.0

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example defines the color correction matrix and turns on the color correction:

```
ActiveUSB1.SetColorMatrix 0.85, 0.1, 0.05, 0.05, 0.98, -0.03, 0.13, -0.15,  
1.02  
ActiveUSB1.ColorCorrect=True
```

Remarks

The relations between the original color components of each pixel (R,G,B) and resulting components (R',G',B') are defined by the following formulas:

$$\begin{aligned}R' &= R*rr + G*rg + B*rb \\G' &= R*gr + G*gg + B*gb \\B' &= R*br + G*bg + B*bb\end{aligned}$$

Color correction is used to eliminate the overlap in the color channels caused by the fact that the light

intended to be detected only by pixels of a certain color is partially seen by pixels of other colors. For example, the red light is partially seen by green and blue elements of the CCD. A properly adjusted color correction matrix can eliminate this overlap.

In order for the white balance to work properly, the sum of each row in the color correct matrix (i.e. $r_r+r_g+r_b$) should be equal to 1.

3.2.121 SetImageWindow

Description

Copies pixel values from the two-dimensional array into the selected window of the current frame.

Syntax

[VB]

```
objActiveUSB.SetImageWindow X, Y, A
```

[C/C++]

```
HRESULT SetImageWindow( short X, short Y, VARIANT Data );
```

Data Types [VB]

X, Y: Integer

A: Variant (SAFEARRAY)

Parameters [C/C++]

X [in], Y [in]

The x- and y- frame coordinates at which the top left corner of the window will be copied.

Data [in]

SAFEARRAY variant containing the pixel values to be copied.

Return Values

S_OK

Success

E_FAIL

Failure

E_INVALIDARG

Input array has wrong data type

Example

This VB example uses the [FrameAcquired](#) event to increase the brightness in the central area of the live image:

```
Private Sub Form_Load()  
ActiveUSB1.Display = False  
ActiveUSB1.Acquire = True  
End Sub  
  
Private Sub ActiveUSB1_FrameAcquired()  
xc = ActiveUSB1.SizeX / 2  
yc = ActiveUSB1.SizeY / 2  
w = ActiveUSB1.GetImageWindow(xc - 70, yc - 50, 140, 100)  
For y = 0 To UBound(w, 2)  
For x = 0 To UBound(w, 1)  
pix = w(x, y) + 50  
If pix > 255 Then
```

```
pix = 255
End If
w(x, y) = pix
Next
Next
ActiveUSB1.SetImageWindow xc - 70, yc - 50, w
ActiveUSB1.Draw
End Sub
```

Remarks

The array submitted to **SetImageWindow** must have the type and dimensions corresponding of those of the frame buffer, as specified in the following table:

Camera Pixel Format	Output Format	Data type	Horizontal Dimension
Mono8	8-bit monochrome	Byte	Width
Mono10, Mono12, Mono16	16-bit gray monochrome	Integer (word)	Width
YUV411, YUV422, YUV444, RGB8, BGR8	24-bit RGB	Byte	Width*3
RGB10, RGB12, RGB16, BGR10, BGR12	48-bit RGB	Integer (word)	Width*3

where *Width* is the intended horizontal size of the window in pixels. If the dimensions of the window are too large to accommodate the frame size, they will be clipped to the frame boundaries.

For real-time image processing **SetImageWindow** should be used in conjunction with the [Draw](#) method.

3.2.122 SetFeature

Description

Sets the numerical value of the specified camera feature.

Syntax

```
[VB]  
objActiveUSB.SetFeature Name, Value
```

```
[C/C++]  
HRESULT SetFeature(bstr Name, float Value );
```

Data Types [VB]

Name: String

Value: Single

Parameters [C/C++]

Name [in]
String specifying the name of the feature

Value [in]
Numerical value of the feature to be set

Return Values

S_OK
Success

E_NOINTERFACE
Feature does not exist or not available

E_INVALIDARG
Feature is not of numerical type

E_OUTOFMEMORY
Value is out of range

E_FAIL
Failure to set the feature

Example

The following VB example demonstrates the use of a scroll control for real-time adjustment of the "GainRaw" feature.

```
Private Sub Form_Load()  
ActiveUSB1.Acquire=True  
HScroll1.Value = ActiveUSB1.GetFeature("GainRaw")  
HScroll1.Min = ActiveUSB1.GetFeatureMin("GainRaw")  
HScroll1.Max = ActiveUSB1.GetFeatureMax("GainRaw")  
End Sub  
  
Private Sub HScroll1_Scroll()
```

```
ActiveUSB1.SetFeature("GainRaw", HScroll1.Value)  
End Sub
```

Remarks

Depending on the [Type](#) of the feature the *Value* argument has the following meaning:

Feature Type	Value
Integer	Integer value converted to floating point
Float	Floating point value
Boolean	0. if False, 1. if True
Enumerated	Ordinal number in the enumeration list
String	N/A
Command	Use non-zero value to execute

If the currently selected camera does not support the specified feature or if the feature is read-only, the method will generate an error.

To set the string value of a feature, use [SetFeatureString](#).

3.2.123 SetFeature64

Description

Sets the numerical value of the specified 64-bit camera feature.

Syntax

[VB]

```
Value=objActiveUSB.SetFeature64 Name, Value
```

[C/C++]

```
HRESULT SetFeature64(bstr Name, double Value );
```

Data Types [VB]

Name: String

Value: Double

Parameters [C/C++]

Name [in]

String specifying the name of the feature

Value [in]

Numerical value of the feature to be set

Return Values

S_OK

Success

E_NOINTERFACE

Feature does not exist or not available

E_FAIL

Failed to read the feature

Example

The following VB example demonstrates how to set the value of a 64-bit feature.

```
Dim dx As Double  
dx=3.141592653589793238  
ActiveUSB1.SetFeature64 "LargeFeature",dx
```

Remarks

It is recommended to use this method only for those integer and floating point features whose length is 64-bit.

If the currently selected camera does not support the specified feature, the method will generate an error.

3.2.124 SetFeatureArray

Description

Sets an array of values associated with the specified camera feature.

Syntax

[VB]

```
Value=objActiveUSB.SetFeatureArray( Name, ElementSize, Buffer )
```

[C/C++]

```
HRESULT SetFeatureArray(bstr Name, short ElementSize, VARIANT* Data);
```

Data Types [VB]

Name: String

ElementSize: Integer

Buffer: Variant (SAFEARRAY)

Parameters [C/C++]

Name [in]

String specifying the name of the feature

ElementSize [in]

A short value specifying the size of each element of the array, in bytes

Data [out, retval]

SAFEARRAY variant containing the values to be copied to the feature's buffer

Return Values

S_OK

Success

E_NOINTERFACE

Feature does not exist or not available

E_FAIL

Failed to read the feature

Example

The following VB example retrieves the camera LUT array and displays the value of the 16th element:

```
LUT=ActiveUSB1.GetFeatureArray( "LUTValueAll" , 4)  
Label.Caption=LUT(16)
```

Remarks

This method retrieves the content of buffers associated with features of the IRegister type. For more information refer to GenICam standard specifications.

Note that the size of an element in the binary buffer returned by an IRegister feature cannot be

determined automatically. Therefore, the *ElementSize* parameter is not optional and must specify the size of each element of the array in bytes.

3.2.125 SetFeatureString

Description

Sets the string value of the specified camera feature.

Syntax

```
[VB]  
objActiveUSB.SetFeatureString Name, Value
```

```
[C/C++]  
HRESULT SetFeatureString(bstr Name, bstr Value );
```

Data Types [VB]

Name: String

Value: String

Parameters [C/C++]

Name [in]
String specifying the name of the feature

Value [in]
String value of the feature to be set

Return Values

S_OK
Success

E_NOINTERFACE
Feature does not exist or not available

E_OUTOFMEMORY
Value is not valid for the feature

E_FAIL
Failed to set the feature

Example

This VB example uses a combo box to switch between different auto exposure modes:

```
Private Sub Form_Load()  
Lst = ActiveUSB1.GetEnumList("ExposureAuto")  
For i = 0 To UBound(Lst)  
Combol.AddItem (Lst(i))  
Next  
Combol.ListIndex = ActiveUSB1.GetFeature("ExposureAuto")  
ActiveUSB1.Acquire = True  
End Sub  
  
Private Sub Combol_Click()  
ActiveUSB1.SetFeatureString Combol.Text  
End Sub
```

Remarks

Depending on the [Type](#) of the feature the *Value* argument has the following meaning:

Feature Type	Value
Integer	Integer value in form of string
Float	Floating point value in form of string
Boolean	"0" or "False", "1" or "True"
Enumerated	String value from the enumerated set
String	String value
Command	"1" or "Execute"

If the currently selected camera does not support the specified feature or if the value is not valid for the feature, the method will generate an error.

3.2.126 SetGains

Description

Sets the levels for the software gain control. If the [LUTMode](#) property is enabled, this operation multiplies the value of each pixel by a given floating point factor thus changing the contrast of the video or its color components.

Syntax

[VB]
`objActiveUSB.SetGains fR [, fG, fB, fGb]`

[C/C++]
`HRESULT SetGains (fR [, fG, fB, fGb])`

Data Types [VB]

fR, fG, fB, fGb: Single

Parameters [C/C++]

fR [in]

Gain factor for the red channel. If the rest of arguments are -1 or omitted, this factor will be applied to all color channels in the video.

fG [in]

Gain factor for the green channel. If four arguments are used, this factor will be applied to Gr pixels of the raw Bayer image.

fB [in]

Gain factor for the blue channel.

fGb [in]

Gain factor for the Gb pixels of the raw Bayer image. If -1 or omitted, $fGb=fG$.

Return Values

S_OK

Success

E_FAIL

Failure.

Example

The following VB example uses three sliders to increase the gain for R, G and B channels up to the factor of 10:

```
Private Sub Form_Load()  
ActiveUSB1.Acquire=True  
HScroll1.Min = 1  
HScroll1.Max = 10  
HScroll2.Min = 1  
HScroll2.Max = 10  
HScroll3.Min = 1  
HScroll3.Max = 10  
ActiveUSB1.LUTMode=True
```

```
End Sub

Private Sub HScroll11_Scroll()
L1 = HScroll11.Value
L2 = HScroll12.Value
L3 = HScroll13.Value
ActiveUSB1.SetGains L1, L2, L3
End Sub

Private Sub HScroll12_Scroll()
L1 = HScroll11.Value
L2 = HScroll12.Value
L3 = HScroll13.Value
ActiveUSB1.SetGains L1, L2, L3
End Sub
```

Remarks

The gain adjustment will be applied to the image only when [LUTMode](#) is enabled. Otherwise the original pixel values will remain intact.

3.2.127 SetLensDistortion

Description

Sets distortion parameters for the lens distortion correction.

Syntax

[VB]

```
objActiveUSB.SetLensDistortion Alpha, Beta, Interpolate
```

[C/C++]

```
HRESULT SetLensDistortion (float Alpha, float Beta, bool Interpolate)
```

Data Types [VB]

Alpha: Single

Beta: Single

Interpolate: Boolean

Parameters [C/C++]

Alpha [in]

Floating point value of the radial distortion coefficient. The allowable range of values is from -1.0 to +1.0.

Beta [in]

Floating point value of the tangential distortion coefficient. The allowable range of values is from -1.0 to +1.0.

Interpolate [in]

If TRUE, the bilinear interpolation will be used for the lens distortion correction.

Return Values

S_OK

Success

E_FAIL

Failure.

Example

The following VB examples activates the lens distortion correction and uses two scroll bars to adjust the distortion parameters in real time :

```
Private Sub Form_Load()  
ActiveUSB1.Acquire=True  
HScroll1.Min = -1000  
HScroll1.Max = 1000  
HScroll2.Min = -1000  
HScroll2.Max = 1000  
ActiveUSB1.SetLensDistortion 0,0,False  
ActiveUSB1.LensCorrect=True  
ActiveUSB1.Acquire=True  
End Sub
```

```
Private Sub HScroll1_Scroll()  
ActiveUSB1.SetLensDistortion HScroll1.Value/1000., HScroll2.Value/1000.,  
False  
End Sub  
  
Private Sub HScroll2_Scroll()  
ActiveUSB1.SetLensDistortion HScroll1.Value/1000., HScroll2.Value/1000.,  
False  
End Sub
```

Remarks

The lens distortion correction is used to compensate for the barrel or pincushion image distortion caused by camera lenses. The barrel distortion makes straight lines at the edges of the image bow outwards and it is commonly seen on wide angle lenses with short focal length. The pincushion distortion makes straight lines at the edges of the image bow inwards, and it is commonly seen on telephoto lenses with long focal length. Some lenses can have a combination of both types of distortion.

Use negative values for Alpha and Beta parameters to correct for the barrel distortion, and positive values to correct for the pincushion distortion. A combination of positive and negative values can be used to correct for the complex type of distortion.

The most practical way to select the distortion parameters is to point the camera to a square grid pattern and empirically adjust the values of both coefficients so that the lines at the edges of the frame become as close to straight ones as possible.

3.2.128 SetLevels

Description

Sets the levels for the Window/Level operation. If the [WindowLevel](#) property is enabled, the operation performs a linear scaling of the histogram of each video frame thus optimizing the contrast, brightness and color of the video.

The lower limit indicates the darkest pixel value that will be mapped to the black (zero) level of the image or its color component. The upper limit indicates the brightest pixel value that will be mapped to the maximum pixel value of the image or its color component. Increasing the lower limit will make the shadows of the image darker. Decreasing the upper limit will make the highlights of the image brighter.

This method also allows you to assign the limits automatically by selecting the *Auto Fit* or/and *Auto White Balance* options. The Auto Fit optimizes the values of the lower and upper limits by providing the maximum contrast between the brightest and darkest pixel in the image. The Auto White Balance optimizes the values of the limits of each color component so that the average color of the image (or selected ROI) becomes gray.

Syntax

[VB]

```
objActiveUSB.SetLevels minR, maxR [, minB, maxG, minB , maxB]
```

[C/C++]

```
HRESULT SetLevels(long minR, long maxR, long minG, long maxG, long minB,  
long maxB);
```

Data Types [VB]

minR, *maxR*: Long
minG, *maxG*, *minB*, *maxB*: Long (optional)

Parameters [C/C++]

minR [in], *maxR* [in]

Lower and upper limits for a monochrome image or for the red channel of a color image. The values are given in percents of the maximum pixel value for the currently selected image format.

If *minR* is set to -1, all the limits will be set automatically based on the Auto Fit algorithm.

If *minR* is set to -2, all the limits will be adjusted automatically based on the Auto White Balance algorithm.

minB [in], *maxB* [in], *minG* [in], *maxG* [in]

Lower and upper limits for the green and blue channels. If these parameters are omitted or set to zero, the green and blue channels are scaled proportionally with the red channel thus preserving the colors on the image. These parameters are ignored for a monochrome video.

Return Values

S_OK

Success
E_FAIL
Failure.

Example

The following VB example uses a slider to adjust the brightness and contrast of the video without changing its color:

```
Private Sub Form_Load()
ActiveUSB1.Acquire=True
HScroll1.Min = 0
HScroll1.Max = 100
HScroll2.Min = 0
HScroll2.Max = 100
ActiveUSB1.WindowLevel=True
End Sub

Private Sub HScroll1_Scroll()
L1 = HScroll1.Value
L2 = HScroll2.Value
ActiveUSB1.SetLevels L1, L2
End Sub

Private Sub HScroll2_Scroll()
L1 = HScroll1.Value
L2 = HScroll2.Value
ActiveUSB1.SetLevels L1, L2
End Sub
```

Remarks

The limits for the Window/Level operation are given in percents of the maximum pixel value for the currently selected image format. The following table shows the maximum pixel value for standard USB3 Vision formats

Pixel Format	Maximum pixel value
Mono8	255
Mono10, Mono10Packed	1023
Mono12, Mono12Packed	4095
Mono14	16383
Mono16	65535
Bayer**8, RGB8Packed, BGR8Packed, RGBA8Packed, BGRA8Packed, YUV411Packed, YUV422Packed, YUV444Packed, RGB8Planar	255
Bayer**10, RGB10Packed, BGR10Packed, BGR10V1Packed, BGR10V2Packed, RGB10Planar	1023
Bayer**12, RGB12Packed, BGR12Packed, RGB12Planar	4095
RGB16Planar	65535

The *Auto Fit* and *Auto White Balance* algorithms work on the currently selected [SetROI](#). When applying the Auto White Balance, make sure to select the ROI corresponding to the gray area in the image.

To retrieve the values of the limits assigned by **SetLevels**, use the [GetLevels](#) method.

3.2.129 SetLUT

Description

Assigns the array of values for the software lookup table. If the [LUTMode](#) is active, the pixel values in the incoming frames will be transformed based on a corresponding factor in the LUT array.

Syntax

[VB]
`objActiveUSB.SetROI LUT, Channels`

[C/C++]
`HRESULT SetROI(VARIANT LUT, short Channels);`

Data Types [VB]

LUT: Variant (Array)

Channels: Integer

Parameters [C/C++]

LUT

SAFEARRAY of floating point factors to be used in the lookup table. The value of 1.0 corresponds to the maximum pixel value in the currently selected video format.

Channels

Number of color channels in the submitted array. Can be one of the following values:

- 1 - the array contains a one-channel LUT which will be used for all color components
- 3 - the array contains a three-channel LUT where the first 1/3 of elements represent the red channel, the second 1/3 of elements - green channel and the last 1/3 of elements - blue channel.
- 4 - the array contains a four-channel LUT used for the operation on a raw Bayer video. The first 1/4 of elements represent the R channel, the second 1/4 of elements - Gr channel, the third 1/4 of elements - B channel and the last 1/4 of elements - Gb channel.

Return Values

S_OK
Success
E_FAIL
Failure.

Example

The following VB example prepares a 3-channel linear LUT with an amplified green channel and applies it to the video:

```
Dim LUT(256 * 3) As Single
For i = 0 To 255
    LUT(i) = i/255.
Next
For i = 256 To 511
    LUT(i) = (i-256)*2/255.
Next
For i = 512 To 767
    LUT(i) = (i-512)/255.
```

[Next](#)

[ActiveUSB1.SetLUT A, 3](#)

Remarks

The LUT processing in ActiveUSB works in two stages. An array submitted by **SetLUT** and containing coefficients in the range 0 - 1.0 is converted to the internal LUT array with the number of elements and their values corresponding to the actual range of pixels in the currently selected format. This guarantees that the input LUT will have an identical effect on images of different types and pixel depths.

For example, if a submitted array has 256 elements with values gradually increasing from 0 to 0.5 and the currently selected format is Mono12 (for which the maximum pixel value is 4095), it will be converted to the internal LUT with 4096 elements and values gradually increasing from 0 to 2047. If the format is changed to Mono8, the internal LUT will automatically shrink to 256 elements with values gradually increasing from 0 to 255. If [LUTMode](#) is active, the value of each pixel will be mapped to the value of a corresponding element in the internal LUT.

For a single-channel LUT the input value of a pixel is used directly as an index in the LUT array. Multi-channel LUTs are logically divided into several subsequent LUTs starting from the red channel array. Therefore, if a submitted lookup table contains 768 elements and 3 channels, the first 256 elements will be used to index and remap red pixels, the second 256 elements - green pixels, and the last 256 elements - blue pixels.

To obtain the values of the current lookup table, use the [GetLUT](#) method.

3.2.130 SetROI

Description

Sets the rectangular region of interest and luminance range for the [histogram](#) and [image statistics](#) calculations.

Syntax

[VB]

```
objActiveUSB.SetROI X1, Y1, X2, Y2 [,L1 , L2]
```

[C/C++]

```
HRESULT SetROI(long X1, long Y1, long X2, long Y2, long L1, long L2);
```

Data Types [VB]

X1, Y1, X2, Y2: Long
L1, L2: Long (optional)

Parameters [C/C++]

X1 [in], Y1 [in]

Pixel coordinates of the top left corner of the ROI, relative to the image origin.

X2 [in], Y2 [in]

Pixel coordinates of the bottom right corner of the ROI, relative to the image origin.

L1 [in], L2 [in]

Optional threshold values specifying the luminance range for image statistics calculations. The luminance range of interest is defined as follows:

L1<L2 Only those pixel values greater or equal to L1 and less or equal to L2 are included into the calculations

L2<L1 Only those pixel values less than L2 or greater than L1 are included into the calculations

L1=L2 Only those pixel values equal to L1 are included into the calculations

Return Values

S_OK

Success

E_FAIL

Failure.

Example

The following VB example uses a slider to adjust a luminance range for the ROI:

```
Private Sub Form_Load()  
ActiveUSB1.Palette = 8  
HScroll1.Min = 0  
HScroll1.Max = 255  
X1 = 0  
Y1 = 0  
X2 = ActiveUSB1.SizeX
```

```
Y2 = ActiveUSB1.SizeY  
L1 = 0  
L2 = 255  
End Sub
```

```
Private Sub HScrollThreshold1_Scroll()  
L1 = HScrollThreshold1.Value  
LabelThreshold1.Caption = L1  
ActiveUSB1.SetROI X1, Y1, X2, Y2, L1, L2  
End Sub
```

Remarks

If all four parameters are zero, the ROI will be reset to the maximum image area.

3.2.131 SetVideoFPS

Description

Sets the playback frame rate of the currently open video file or memory sequence.

Syntax

[VB]
`objActiveUSB.SetVideoFPS Value`

[C/C++]
`HRESULT SetVideoFPS(float Value);`

Data Types [VB]

Value: Single

Parameters [C/C++]

Value [in]
The fps value to be set

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example opens an AVI file, doubles its frame rate and starts the playback.

```
ActiveUSB1.OpenVideo "c:\\video1.avi"  
fps=ActiveUSB1.GetVideoFPS * 2  
ActiveUSB1.SetVideoFPS fps  
ActiveUSB1.PlayVideo
```

Remarks

The actual frame rate during the playback of a video file may be lower than the set frame rate. This depends on the hard drive performance and the use of compression during the recording. If an AVI file is [open](#) with the sound option, the playback rate cannot be higher than 16 times of the recorded fps.

3.2.132 SetVideoPosition

Description

Seeks the specified frame in the currently open video file or memory sequence, extracts it and displays in the *ActiveUSB* window.

Syntax

```
[VB]  
objActiveUSB.SetVideoPosition Frame [, Mode=0 ]
```

```
[C/C++]  
HRESULT SetVideoPosition(long Frame, short Mode);
```

Data Types [VB]

Frame: Long

Mode: Integer

Parameters [C/C++]

Frame [in]

The zero-based index of the frame to set.

Mode [in]

If 0, *Frame* will be used as an absolute zero-based position of the frame in the file or sequence.

If 1, *Frame* will be used as an incremental move relative to the current frame position (i.e.

Frame=2 will move the video position two frames forward, while Frame=-2 will move it two frames back.

Return Values

S_OK

Success

E_FAIL

Failure.

Example

This VB example shows how to use a scroll bar to move between the frames in the AVI file.

```
Private Sub Form_Load()  
ActiveUSB1.OpenVideo "C:\\video1.avi"  
HScroll1.Min=0  
HScroll1.Max=ActiveUSB1.GetVideoFrameCount - 1  
End Sub
```

```
Private Sub HScroll1_Scroll()  
frameposition = HScroll1.Value  
ActiveUSB1.SetVideoPosition frameposition  
End Sub
```

Remarks

This method extracts the specified frame from the video file or memory sequence and loads it into the

internal *ActiveUSB* buffer. The [FrameLoaded](#) event will be fired, when **SetVideoPosition** is called successfully. The pixels of the loaded frame can be accessed via *ActiveUSB* image access functions ([GetImageData](#), [GetImagePointer](#), [GetImageWindow](#) etc).

The speed at which a random frame can be extracted from a video file depends on the performance of the hard drive and the compression used during the recording.

If the frame is extracted from a memory sequence, all current conversion properties (such as [Bayer](#), [WindowLevel](#), [BkgCorrect](#), [Rotate](#)) will be applied to the frame for the display.

Calling this method will automatically turn off the live video by setting the [Acquire](#) property to FALSE.

3.2.133 SetVideoSync

Description

Sets the playback synchronization mode (freerun or triggered) for the currently open video file or memory sequence. Used in combination with [PlayVideo](#).

Syntax

```
[VB]  
objActiveUSB.SetVideoSync Mode
```

```
[C/C++]  
HRESULT SetVideoSync(short Mode);
```

Data Types [VB]

Mode: Integer

Parameters [C/C++]

Mode [in]

The synchronization mode of the currently open video source. Select one of the following values:
0 - internal (freerun) synchronization. Video will be played by *ActiveUSB* at the currently set frame rate.
1 - external (triggered) synchronization. Video will advanced to the next frame when [TriggerVideo](#) is called.

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example shows how to synchronously play two AVI files using two *ActiveUSB* objects.

```
Private Sub Form_Load()  
ActiveUSB1.OpenVideo "C:\\video1.avi"  
ActiveUSB2.OpenVideo "C:\\video2.avi"  
ActiveUSB1.SetVideoSync 1  
ActiveUSB2.SetVideoSync 1  
ActiveUSB1.PlayVideo  
ActiveUSB2.PlayVideo  
fps=ActiveUSB1.GetVideoFPS  
Timer1.Interval=1000/fps  
End Sub  
  
Private Sub Timer1_Timer()  
ActiveUSB1.TriggerVideo  
ActiveUSB2.TriggerVideo  
End Sub
```

Remarks

When the video file or sequence has just been opened, the playback is set to the internal synchronization.

Setting the video playback to the external synchronization mode is necessary when two video files or sequences need to be played synchronously.

Note that the external synchronization cannot be applied to an AVI files opened with the sound option.

3.2.134 SetVideoVolume

Description

Sets the audio volume level of the AVI file currently open for a playback.

Syntax

[VB]
`objActiveUSB.SetVideoVolume Value`

[C/C++]
`HRESULT SetVideoVolume(short Value);`

Data Types [VB]

Value: Integer

Parameters [C/C++]

Value [in]
The volume to be set, in percent.

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example uses a scroll bar to adjust the video volume.

```
Private Sub Form_Load()  
HScroll1.Min=0  
HScroll1.Max=100  
ActiveUSB1.OpenVideo "C:\\video1.avi", True  
HScroll1.Value=ActiveUSB1.GetVideoVolume  
ActiveUSB1.PlayVideo  
End Sub  
  
Private Sub HScroll1_Scroll()  
ActiveUSB1.SetVideoVolume HScroll1.Value  
End Sub
```

Remarks

In order for this method to work, the AVI file must be recorded with the sound and [opened](#) with the sound option enabled.

3.2.135 SetWebStreamer

Description

Configures parameters of the RTSP web streaming.

Syntax

[VB]

```
objActiveGige.SetWebStreamer srcAddress, dstAddress, frameRate, quality[,  
tunnelOverHttp]
```

[C/C++]

```
HRESULT SetDestinationIP(BSTR srcAddress, BSTR dstAddress, float frameRate,  
short quality, bool tunnelOverHttp);
```

Data Types [VB]

dstAddress: String
srcAddress: String
frameRate: Float
quality: Integer
tunnelOverHttp: Boolean

Parameters [C/C++]

srcAddress [in]

String specifying the source address of the web streaming in the following format:

"xxx.xxx.xxx.xxx[:pppp/filename]", where

xxx.xxx.xxx.xxx - the IP address of the local network interface to be used for the web streaming (typically a local area interface)

pppp - the local source port to be used for the web streaming; if omitted, port 8554 will be used

filename - the file name assigned to the stream; if omitted, "ActiveUsb.sdp" will be used

dstAddress [in]

String specifying the IP address of a remote playback device. For the multicast streaming, use an address in the range 239.0.0.0 - 239.255.255.255

frameRate [in]

Floating point value specifying the frame rate limit for the web stream.

quality [in]

Integer value in the range 0-6 specifying the H.264 compression quality. The higher the quality is, the higher the streaming bandwidth will be.

tunnelOverHttp [in]

Enabling this boolean value will reroute the web streaming via HTTP ports 80, 8000 or 8080

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example sets the web streamer's parameters and activates the streaming. The port number is omitted in the source address, so the default port 8554 is used:

```
sourceAddr="192.168.0.5/test.sdp"  
destAddr="192.168.0.10"  
streamFPS=15  
streamQuality=4  
ActiveUSB1.SetWebStreamer sourceAddr, destAddr, streamFPS, streamQuality  
ActiveUSB1.Acquire=True  
ActiveUSB1.WebStream=True
```

Remarks

The web streaming option allows you to automatically convert video outputted by the camera into the H.264 compression format and transmit it over the wireless or wired network using the RTSP protocol to a remote playback devices, such as PCs, tablets and smartphones. Use the **SetWebStreamer** method to configure parameters of the web streamer before turning the [web streaming](#) on.

When specifying the source address, make sure to use the IP address of a local interface connected to the same network on which your remote client device operates. The port number and/or filename can be omitted in the source address string, in which case default values will be used.

When specifying the destination address, use the intranet IP address of a remote device or network interface on a remote PC (you can look up IP addresses assigned to different devices on your network by accessing your router's web interface). If you want to multicast the web stream to all devices on your network, enter a multicast IP address such as 239.0.0.1

If the value of the frameRate parameter exceeds the camera's fps, the camera's fps will be used as the frame rate of the web video.

To watch a transmitted video on a remote playback device, use an RTSP/RTP client such as VLC Media Player or Fresh Video Player.

The following steps describe how to use VLC Media Player on a remote Windows-based client PC to display a web video stream transmitted by an ActiveUSB-based streaming application:

- Run VLC media player on a client PC (the IP address of the client PC must be same as the destination address used in your streaming application).
 - Select Media -> Open Network Stream in the player's menu. The Open Media dialog box will be displayed.
 - In the Network URL field enter the full RTSP web address matching the source address used in your ActiveUSB-based application, for example:
rtsp://192.168.0.5:8554/test.asp
 - Click the Play button. Within a few seconds a decoded video should appear on the player's screen.
-

3.2.136 ShowAudioDlg

Description

Displays the audio Input dialog for adjusting the mixer properties of the audio source.

Syntax

[VB]
`objActiveUSB.ShowAudioDlg`

[C/C++]
`HRESULT ShowAudioDlg();`

Parameters

None

Return Values

S_OK
Success
E_FAIL
Failure

Example

This VB example selects the audio source and displays the audio input dialog:

```
ActiveUSB1.SetAudioSource 0  
ActiveUSB1.ShowAudioDlg
```

Remarks

The options available in the audio input dialog will depend on the configuration of your audio devices.

3.2.137 ShowCodecDlg

Description

Displays the configuration dialog for the currently selected codec. The dialog is provided by the codec's manufacturer.

Syntax

[VB]
`objActiveUSB.ShowCodecDlg`

[C/C++]
`HRESULT ShowCodecDlg();`

Parameters

None

Return Values

S_OK
Success
E_FAIL
Failure

Example

This VB example displays the configuration dialog for the DivX codec:

```
ActiveUSB1.SetCodec "DivX 5.0.2 Codec"  
ActiveUSB1.ShowCodecDlg
```

Remarks

The settings selected in the codec configuration dialog will be remembered as long as the corresponding ActiveUSB control remains in the memory.

3.2.138 ShowCompressionDlg

Description

Displays the compression dialog for the AVI recording. The dialog allows you to select a desired codec and adjust its parameters.

Syntax

```
[VB]  
objActiveUSB.ShowCompressionDlg
```

```
[C/C++]  
HRESULT ShowCompressionDlg( );
```

Parameters

None

Return Values

S_OK
Success
E_FAIL
Failure

Example

This VB example demonstrates a simple video recording application

```
Private Sub Form_Load()  
ActiveUSB1.Acquire=True  
End Sub
```

```
Private Sub CompressionButton_Click()  
ActiveUSB1.ShowCompressionDlg  
End Sub
```

```
Private Sub StartButton_Click()  
ActiveUSB1.StartCapture "c:\mycapture.avi"  
End Sub
```

```
Private Sub StopButton_Click()  
ActiveUSB1.StopCapture  
End Sub
```

Remarks

The settings selected in the compression dialog are remembered as long as the corresponding *ActiveUSB* control remains in the memory.

ActiveUSB video recording engine includes a proprietary "Raw Uncompressed" codec. When the

camera streams video in the Bayer format and the "Raw Uncompressed" codec is selected, *ActiveUSB* records the video in its original raw format while displaying it in color. This reduces the disk space requirements by three times without any degradation in the image quality. In addition, when high-depth pixel formats are used (such as Bayer10, Bayer12, Bayer16, Mono10, Mono 12, Mono16), the recorded video will be packed into the 12-bit per pixel format as opposed to 16 bit, thus saving extra 25% of the disk space. To decode and play back AVI files recorded with the "Raw Uncompressed" codec, use video playback methods of the DVR version of *ActiveUSB*.

3.2.139 ShowProperties

Description

Displays *ActiveUSB* property pages in runtime mode.

Syntax

```
[VB]  
objActiveUSB.ShowProperties [ EnableCamList, Page ]
```

```
[C/C++]  
HRESULT ShowProperties( bool bEnableCamList, short iPage );
```

Parameters

EnableCamList: Boolean
Page: Integer

Parameters [C/C++]

bEnableCamList [in]

Set to TRUE to enable the camera selection box in the Source property page, or set to FALSE to disable it. Default value - TRUE.

iPage [in]

The index of the property page to be activated. Use one of the following values to activate a specific property page:

- 0 or omitted - the property page which was active last time the property pages were used.
- 1 - the Source page
- 2 - the Format page
- 3 - the Analog page
- 4 - the Input/Output page
- 5 - the Advanced page
- 6 - the Display page

Return Values

S_OK
Success
E_FAIL
Failure

Example

This VB example demonstrates how to display the property pages in runtime mode.

```
Private Sub Properties_Click()  
ActiveUSB1.ShowProperties False  
End Sub
```

Remarks

Setting the *EnableCamList* parameter to False allows you to disable the camera selection box in the **Source** property page thus preventing a user from switching to another camera. This is a recommended scenario in case your application uses several *ActiveUSB* objects configured for different cameras. See [PropertyPages](#) for more information.

3.2.140 SoftTrigger

Description

Generates an internal trigger signal.

Syntax

```
[VB]  
objActiveUSB.SoftTrigger
```

```
[C/C++]  
HRESULT SoftTrigger();
```

Parameters

None

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example demonstrates the use of the software trigger:

```
Private Sub Form_Load()  
ActiveUSB1.Acquire = True  
ActiveUSB1.TriggerSource = "Software"  
ActiveUSB1.Trigger = True  
End Sub  
  
Private Sub SoftTrig_Click()  
ActiveUSB1.SoftTrigger  
End Sub
```

Remarks

Use **SoftTrigger** to simulate the trigger signal. Note that this method will only have effect if [TriggerSource](#) is set to the "Software".

3.2.141 StartCapture

Description

Starts time-lapse video capture to the specified AVI file or series of image files (avi, raw, bmp, tif, jpg or dpx). Use [StopCapture](#) to end video capture.

Syntax

[VB]

```
objActiveUSB.StartCapture File [, Timelapse = 0, Playrate = 0 ]
```

[C/C++]

```
HRESULT StartCapture( bstr File, float Timelapse, float Playrate );
```

Data Types [VB]

File : String

Timelapse : Single (optional)

Playrate : Single (optional)

Parameters [C/C++]

File [in]

The string containing the path to the avi file or image file (raw, bmp, tif, jpg or dpx).

Timelapse [in]

The interval between consecutive frames in seconds

Playrate [in]

The frame rate at which AVI file will be played in frames per second.

Return Values

S_OK

Success

E_FAIL

Failure.

E_INVALIDARG

Invalid file name.

Example

This VB example demonstrates how to capture the video to an AVI file at the current frame rate:.

```
Private Sub StartButton_Click()  
ActiveUSB1.StartCapture "c:\mycapture.avi"  
End Sub
```

```
Private Sub StopButton_Click()  
ActiveUSB1.StopCapture  
End Sub
```

This C# example demonstrates how to capture a series of TIFF images at 0.5 sec interval:

```
private void startbutton_Click(object sender, System.EventArgs e)
{
    axActiveUSB1.StartCapture("c:\\images\\myframe.tif", 0.5, 0.);
}

private void stopbutton_Click(object sender, System.EventArgs e)
{
    axActiveUSB1.StopCapture();
}
```

Remarks

The [Acquire](#) property must be set to TRUE prior to calling this method.

To record compressed AVI files, use [ShowCompressionDlg](#), [SetCodec](#) and [ShowCodecDlg](#). If no codec has been selected, *ActiveUSB* will record AVI files in the uncompressed format, 8- or 24-bits per pixel.

If bmp, tif or jpg extension is specified for the file path, the file name is used as a template to which the ordinal number of the frame captured is appended. In the C# example above consecutive frames will be stored to files "myframe00001.tif", "myframe00002.tif" and so on. Note that TIF format can store 16- and 48-bit per pixel images.

If *Timelapse* is not specified, the video will be recorded at the maximum frame rate defined by the camera settings and system throughput.

If *Playrate* is not specified, the video will be played back at the same rate it was captured. If *Playrate* is specified for multiframe capture, it will define the compression value for individual image files. See [SaveImage](#) for more detail.

If the external device (typically, a hard drive or memory card) doesn't have enough space, the capture will stop automatically when the device is full and the [CaptureCompleted](#) event will be fired.

3.2.142 StopCapture

Description

Stops video capture to an AVI file or series of images.

Syntax

[VB]
`objActiveUSB.StopCapture`

[C/C++]
`HRESULT StopCapture();`

Parameters

None

Return Values

S_OK
Success
E_FAIL
Failure

Example

This VB example demonstrates how to capture the video to an AVI file with 0.5 sec time lapse between the frames.

```
Private Sub StartButton_Click()  
ActiveUSB1.StartCapture "c:\mycapture.avi", 0.5  
End Sub
```

```
Private Sub StopButton_Click()  
ActiveUSB1.StopCapture  
End Sub
```

Remarks

Use this method to end the video capture initiated by [StartCapture](#). The [CaptureCompleted](#) event will be generated when this method is called.

3.2.143 StartSequenceCapture

Description

Starts acquiring a sequence of frames into the memory. Use [StopSequenceCapture](#) to end the sequence capture process.

Syntax

[VB]

```
objActiveUSB.StartSequenceCapture Frames [, Timelapse = 0 ]
```

[C/C++]

```
HRESULT StartSequenceCapture( long Frames, float Timelapse);
```

Data Types [VB]

Frames : Long

Timelapse : Single (optional)

Parameters [C/C++]

Frames [in]

The number of frames to capture.

Frames>0 - sequence capture will stop automatically after the specified number of frames is reached, unless [StopSequenceCapture](#) is called before that.

Frames<0 - loop recording mode; when the specified number of frames is reached, the capture will continue in a loop until [StopSequenceCapture](#) is called.

Timelapse [in]

The interval between consecutive frame acquisitions in seconds. If zero or not specified, the sequence will be captured at the current frame rate.

Return Values

S_OK

Success

E_FAIL

Failure.

E_INVALIDARG

Invalid file name.

Example

This VB example demonstrates how to capture the video loop into the system memory at the current frame rate. The maximum sequence size is set to 1000 frames:

```
Private Sub StartButton_Click()  
ActiveUSB1.StartSequenceCapture -1000  
End Sub
```

```
Private Sub StopButton_Click()  
ActiveUSB1.StopSequenceCapture  
End Sub
```

Remarks

It is recommended to use [CreateSequence](#) for memory allocation before calling **StartSequenceCapture**. If [CreateSequence](#) was not called or did not allocate enough memory, **StartSequenceCapture** will have to reserve the necessary amount of memory itself, which will introduce a delay between the function call and actual start of the sequence capture.

During the sequence capture the [FrameRecorded](#) event will be fired per each frame recorded into the memory.

In the loop recording mode (*Frames* < 0) after the frame limit is reached, ActiveUSB will maintain the specified number of recorded frames in the memory by removing a frame from the head of the sequence and adding the latest frame to its tail. Thus, in the example above, at any given moment the sequence will contain the most recent 1000 frames.

To reduce the memory consumption, *ActiveUSB* records frames in a sequence in the original raw image format. When the sequence is played back, its frames are converted to a regular monochrome or RGB format on the fly. See [PlayVideo](#) for more details.

If *Timelapse* is not specified, the sequence will be recorded at the maximum frame rate defined by the camera settings and system throughput.

3.2.144 StopSequenceCapture

Description

Stops acquiring a sequence of frames into the memory.

Syntax

[VB]
`objActiveUSB.StopSequenceCapture`

[C/C++]
`HRESULT StopSequenceCapture();`

Parameters

None

Return Values

S_OK
Success
E_FAIL
Failure

Example

This VB example demonstrates how to end the sequence capture using a button click.

```
Private Sub StopButton_Click()  
ActiveUSB1.StopSequenceCapture  
End Sub
```

Remarks

Use this method to explicitly end the sequence capture initiated by [StartSequenceCapture](#).

When the sequence capture stops, the [CaptureComplete](#) event will be fired.

3.2.145 StartVideoCapture

Description

Starts time-lapse video capture to the AVI file created by [CreateVideo](#). Use [StopVideoCapture](#) to end video capture.

Syntax

[VB]

```
objActiveUSB.StartVideoCapture [Frames = 0 [, Timelapse = 0, Playrate = 0 ]
```

[C/C++]

```
HRESULT StartVideoCapture( long Frames, float Timelapse, float Playrate );
```

Data Types [VB]

Frames : Long

Timelapse : Single (optional)

Playrate : Single (optional)

Parameters [C/C++]

Frames [in]

The number of frames to capture. The video capture will stop automatically after the specified number of frames is reached or [StopVideoCapture](#) is called. If zero or omitted, the video capture will continue until the disk is full or [StopVideoCapture](#) is called.

Timelapse [in]

The interval between consecutive frames in seconds. If zero or omitted, the video will be recorded at the maximum frame rate define by the camera settings and system throughput.

Playrate [in]

The frame rate at which AVT file will be played in frames per second. If zero or omitted, the video will be played back at the same rate it was captured.

Return Values

S_OK

Success

E_FAIL

Failure.

Example

This VB example demonstrates how to create an AVI file and capture 1000 frames into it.

```
Private Sub LoadForm()  
ActiveUSB1.CreateVideo "c:\\mycapture.avi"  
ActiveUSB1.Acquire = True  
End Sub
```

```
Private Sub StartButton_Click()  
ActiveUSB1.StartVideoCapture 1000  
End Sub
```

Remarks

The advantage of using this method as opposed to [StartCapture](#) is that **StartVideoCapture** is called after an AVI file is already opened and preallocated with [CreateVideo](#). As a result, the video capture starts immediately with no delay.

The [Acquire](#) property must be set to TRUE prior to calling this method.

To record compressed AVI files, use [ShowCompressionDlg](#), [SetCodec](#) and [ShowCodecDlg](#). If no codec has been selected, *ActiveUSB* will record AVI files in the uncompressed format, 8- or 24-bits per pixel.

If the external device (typically, a hard drive or memory card) does not have enough space, the capture will stop automatically when the device is full and the [CaptureCompleted](#) event will be fired.

3.2.146 StopVideoCapture

Description

Stops video capture to an AVI file.

Syntax

[VB]
`objActiveUSB.StopVideoCapture`

[C/C++]
`HRESULT StopVideoCapture();`

Parameters

None

Return Values

S_OK
Success
E_FAIL
Failure

Example

This VB example demonstrates how to capture the video to an AVI file with 0.5 sec time lapse between the frames.

```
Private Sub LoadForm()  
ActiveUSB1.CreateVideo "c:\\mycapture.avi"  
ActiveUSB1.Acquire = True  
End Sub
```

```
Private Sub StartButton_Click()  
ActiveUSB1.StartVideoCapture 0, 0.5  
End Sub
```

```
Private Sub StopButton_Click()  
ActiveUSB1.StopVideoCapture  
End Sub
```

Remarks

Use this method to end the video capture initiated by [StartVideoCapture](#). The [CaptureCompleted](#) event will be raised when this method is called.

3.2.147 StopVideo

Description

Stops the playback of a video file or memory sequence that has been initiated by [PlayVideo](#).

Syntax

```
[VB]  
objActiveUSB.StopVideo
```

```
[C/C++]  
HRESULT StopVideo( );
```

Parameters [C/C++]

None

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example shows how to stop and resume the video playback using push buttons.

```
Private Sub Form_Load()  
ActiveUSB1.OpenVideo "C:\\video1.avi"  
End Sub
```

```
Private Sub Play_Click()  
ActiveUSB1.PlayVideo -1  
End Sub
```

```
Private Sub StopButton_Click()  
ActiveUSB1.StopVideo  
End Sub
```

Remarks

When this method is called, the [PlayCompleted](#) event will be fired.

3.2.148 TriggerVideo

Description

Advances the playback of the video file or memory sequence to the next frame. Used in combination with [SetVideoSync](#) and [PlayVideo](#).

Syntax

[VB]
`objActiveUSB.TriggerVideo`

[C/C++]
`HRESULT TriggerVideo();`

Parameters [C/C++]

None

Return Values

S_OK
Success
E_FAIL
Failure

Example

This VB example shows how to synchronously play two AVI files using two *ActiveUSB* objects.

```
Private Sub Form_Load()  
ActiveUSB1.OpenVideo "C:\\video1.avi"  
ActiveUSB2.OpenVideo "C:\\video2.avi"  
ActiveUSB1.SetVideoSync 1  
ActiveUSB2.SetVideoSync 1  
ActiveUSB1.PlayVideo  
ActiveUSB2.PlayVideo  
fps=ActiveUSB1.GetVideoFPS  
Timer1.Interval=1000/fps  
End Sub  
  
Private Sub Timer1_Timer()  
ActiveUSB1.TriggerVideo  
ActiveUSB2.TriggerVideo  
End Sub
```

Remarks

This method can only be used when the video synchronization has been set to the external mode(see [SetVideoSync](#)) and [PlayVideo](#) has been called.

3.2.149 WriteBlock

Description

Writes a block of data into the internal camera memory starting from the specified bootstrap address.

Syntax

[VB]

```
objActiveUSB.WriteBlock Offset, Buffer, nBytes
```

[C/C++]

```
HRESULT WriteBlock( long Offset, Variant Buffer, long nBytes );
```

Data Types [VB]

Offset: Long

Buffer: Variant (pointer)

nBytes: Long

Parameters [C/C++]

Offset [in]

The 32-bit offset into the camera register address space.

Buffer

Variant containing pointer to a buffer that contains the data to be written into the camera memory starting from the specified offset

nBytes

The number of bytes to write to the specified offset

Return Values

S_OK

Success

E_FAIL

Failure.

Example

This C++ example writes a block of 128 bytes starting from a specified offset in the camera address space:

```
unsigned char buffer[128]
for (int i=0;i<128;i++)
    buffer[i]=i;
VARIANT v;
v.pvVal=buffer;
ActiveUSB.WriteBlock(0xA0000,v,128);
```

Remarks

Using this method typically provides a much faster access to the internal camera memory than

repetitive calls to [WriteRegister](#).

If the specified offset does not exist or the camera cannot accommodate the size of the data block, this method will return an error.

3.2.150 WriteFile

Description

Transfers the indicated amount of bytes from the data array in memory to the specified file in the device.

Syntax

[VB]

```
objActiveUSB.WriteFile Name, Offset, Length, Data
```

[C/C++]

```
HRESULT WriteFile( bstr Name, long Offset, long Length, VARIANT Data );
```

Data Types [VB]

Name: String

Offset: Long

Length: Long

Data: Variant (Array of Bytes)

Parameters [C/C++]

Name [in]

String specifying the name of the file in the device

Offset [in]

The transfer starting position from the beginning of the file in bytes

Length [in]

The number of bytes to transfer to the specified file

Data [in]

Pointer to SAFEARRAY of bytes containing data to be transferred to the file

Return Values

S_OK

Success

E_NOINTERFACE

File with the specified name does not exist in the device

E_FAIL

Failure.

Example

This VB example reads a LUT array from the file on the device, inverts it and writes back to the device

```
w = ActiveUSB1.ReadFile("LUTArray1", 0, 256)
For y = 0 To 255
w(y) = 255- w(y)
Next
ActiveUSB1.WriteFile "LutArray1",0,256,w
```

Remarks

The File Access functionality allows an application to read and write files hosted on the device. The data in those files may contain look-up tables, configuration sets, firmware, and other information.

You can use the WriteFile method to transfer the content of a file on a hard drive to the file in the device. Note that in order for this method to work, the [Access Mode](#) of the file should be "W" or "RW".

3.2.151 WriteRegister

Description

Writes a 32-bit integer value to the specified bootstrap register of the current camera.

Syntax

[VB]
`objActiveUSB.WriteRegister Reg, Value`

[C/C++]
`HRESULT WriteRegister(long Reg, long Value);`

Data Types [VB]

Reg: Long

Parameters [C/C++]

Reg [in]
The 32-bit offset of the register in the camera's address space
Value [in]
The value to be written in the register

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This C++ example changes the value of the heartbeat timeout using the corresponding USB3 Vision™ register.

```
value=ActiveUSB.WriteRegister(0x0938,5000);
```

This VB example modifies a value of the custom camera feature.

```
ActiveUSB.WriteRegister &HA080,512
```

Remarks

The list of standard bootstrap registers can be found in *"USB3 Vision Camera Interface Standard For Machine Vision"* published by the Automated Imaging Association.

This method can be used to control custom camera attributes not available through USB3 Vision™ feature interface. Refer to the camera documentation for the list of the manufacturer-specific registers.

3.3 Events

The following events are generated by *ActiveUSB* control:

FrameAcquired	ID: 1	Called after a frame has been acquired and decoded
FrameAcquiredX	ID:10	Called after a frame has been acquired and decoded (multithreaded version)
RawFrameAcquired	ID:19	Called after a raw frame has arrived from the camera
FrameReady	ID:20	Called after a frame is about to be submitted for display
FrameDropped	ID: 3	Called if a frame is dropped during the video acquisition
Timeout	ID: 2	Called if the acquisition timeout has expired
FormatChanged	ID: 8	Called if the video size or pixel format has changed
FrameRecorded	ID:16	Called when a frame is added to a video file, image file or memory sequence
FrameLoaded	ID:15	Called when a frame is loaded from a video file, image file or memory sequence
CaptureCompleted	ID:17	Called when the capture into a video file or memory sequence is stopped
PlayCompleted	ID:18	Called when the playback of a video file or memory sequence is stopped
EventMessage	ID:13	Called when an asynchronous event arrives from the camera
EventDataMessage	ID:14	Called when an asynchronous data event arrives from the camera
CameraPlugged	ID:11	Called if a camera gets connected to the system
CameraUnplugged	ID:12	Called if a camera gets disconnected from the system
MouseDown	ID: 4	Called when the mouse button is pressed inside the control window
MouseUp	ID: 5	Called when the mouse button is released inside the control window
MouseDownRight	ID:21	Called when the right mouse button is pressed inside the control window
MouseUpRight	ID:22	Called when the right mouse button is released inside the control window
MouseDownClick	ID: 9	Called when the mouse button is double-clicked inside the control window
MouseMove	ID: 6	Called when the mouse button has moved inside the control window
Scroll	ID: 7	Called when the live video display has been scrolled

3.3.1 CameraPlugged

Description

This event is fired each time a USB3 Vision™ camera is connected to the system .

Syntax

[VB]

```
Private Sub objActiveUSB_CameraPlugged(ByVal Camera As Integer)
```

[C/C++]

```
HRESULT Fire_CameraPlugged(SHORT Camera);
```

Data Types [VB]

Camera: Integer

Parameters [C/C++]

Camera

The index of the camera that has been connected.

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example uses the **CameraPlugged** and **CameraUnplugged** events to detect changes in the number of USB3 Vision™ cameras in the network:

```
Private Sub ActiveUSB1_CameraPlugged(ByVal Camera As Integer)
    Dim Msg As String
    Msg = "Camera #" + Str(Camera) + " has been connected"
    MsgBox Msg
End Sub
```

```
Private Sub ActiveUSB1_CameraUnplugged(ByVal Camera As Integer)
    Dim Msg As String
    Msg = "Camera #" + Str(Camera) + " has been disconnected"
    MsgBox Msg
End Sub
```

3.3.2 CameraUnplugged

Description

This event is fired each time a USB3 Vision™ camera gets disconnected from the system.

Syntax

[VB]

```
Private Sub objActiveUSB_CameraUnplugged(ByVal Camera As Integer)
```

[C/C++]

```
HRESULT Fire_CameraUnplugged(SHORT Camera);
```

Data Types [VB]

Camera: Integer

Parameters [C/C++]

Camera

The index of the camera that has been disconnected.

Return Values

S_OK

Success

E_FAIL

Failure.

Example

This VB example uses the **CameraUnplugged** to detect when the currently selected camera is unplugged:

```
Private Sub ActiveUSB1_CameraUnplugged(ByVal Camera As Integer)
    If ActiveUSB1.Camera = Camera Then
        MsgBox "Currently selected camera has been disconnected"
    End If
End Sub
```

3.3.3 CaptureCompleted

Description

This event is fired when the capture into the memory sequence or video file is stopped.

Syntax

[VB]

```
Private Sub objActiveUSB_CaptureCompleted(ByVal Frames As Long)
```

[C/C++]

```
HRESULT Fire_CaptureCompleted(long Frames);
```

Data Types [VB]

Frames: long

Parameters [C/C++]

Frames

The number of frames recorded.

Example

This VB example records 1000 frames into the memory sequence and uses the **CaptureCompleted** event to save it in an AVI file.

```
Private Sub StartButton_Click()  
ActiveUSB1.StartSequenceCapture 1000  
End Sub
```

```
Private Sub ActiveUSB1_CaptureCompleted(ByVal Frames As Long)  
SaveSequence "C:\\video.avi"  
End Sub
```

Remarks

One of the following conditions must be met, before the **CaptureCompleted** event will be fired:

Capture has stopped automatically, because the specified number of frames has been acquired.

Capture to a file has stopped automatically because the device is full.

Capture has been stopped explicitly by calling the [StopSequenceCapture](#) or [StopCapture](#) methods.

3.3.4 EventMessage

Description

This event is fired each time an asynchronous event arrives from the camera on the message channel.

Syntax

[VB]

```
Private Sub objActiveUSB_EventMessage(ByVal eventID as long, ByVal blockID  
as long, ByVal timestamp as double,  
ByVal channelIndex as long)
```

[C/C++]

```
HRESULT Fire_CameraPlugged(long eventId, long blockId, double timestamp,  
long channelIndex);
```

Data Types [VB]

eventID: long
blockID: long
timestamp: double
channelIndex: long

Parameters [C/C++]

eventID
The 16-bit even identifier.

blockID
The 16-bit identifier of the data block associated with the event. 0 if there is no data block association.

timestamp
The timestamp representing the time at which the event was fired by the camera.

channelIndex
The index of the stream channel associated with the event.

Return Values

S_OK
Success

E_FAIL
Failure.

Example

This VB example intercepts message events and displays the information associated with each event:

```
Private Sub Form_Load()  
ActiveUSB1.SetFeatureString("EventNotification", "On")  
ActiveUSB1.Acquire = True  
End Sub
```

```
Private Sub ActiveUSB1_EventMessage(ByVal eventID as long, ByVal blockID
as long, ByVal timestamp as double, ByVal channelIndex as long)
LabelEventID.Caption=eventID
LabelBlockID.Caption=blockID
LabelTimestamp.Caption=timestamp
End Sub
```

Remarks

The **EventMessage** events are generated only for the EVENT type messages. For the EVENTDATA messages *ActiveUSB* will fire [EventDataMessage](#) events.

If several events are encapsulated into one USB3 Vision message, the **EventMessage** event will fire several times.

For more information on the USB3 Vision messaging refer to "*USB3 Vision Camera Interface Standard For Machine Vision*" published by the Automated Imaging Association.

3.3.5 EventDataMessage

Description

This event is fired each time an asynchronous data event arrives from the camera on the message channel.

Syntax

[VB]

```
Private Sub objActiveUSB_EventDataMessage(ByVal eventID as long, ByVal  
blockID as long, ByVal data as Variant,           ByVal timestamp as double, ByVal  
channelIndex as long)
```

[C/C++]

```
HRESULT Fire_EventDataMessage(SHORT Camera);
```

Data Types [VB]

eventID: Long
blockID: Long
data: Variant (Array)
timestamp: Double
channelIndex: Long

Parameters [C/C++]

eventID
The 16-bit even identifier.

blockID
The 16-bit identifier of the data block associated with the event. 0 if there is no data block association.

data
Pointer to SAFEARRAY of bytes contating the data encapsulated in the message.

timestamp
The timestamp representing the time at which the event was fired by the camera.

channelIndex
The index of the stream channel associated with the event.

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example intercepts message events and displays the information associated with each event:

```
Private Sub Form_Load()
```

```
ActiveUSB1.SetFeatureString("EventNotification","On")
ActiveUSB1.Acquire = True
End Sub
```

```
Private Sub ActiveUSB1_EventMessage(ByVal eventID as long, ByVal blockID
as long, ByVal A as Variant,
                                ByVal timestamp as double, ByVal
channelIndex as long)
LabelEventID.Caption=eventID
LabelBlockID.Caption=blockID
LabelA0.Caption=A(0)
LabelA0.Caption=A(1)
LabelA0.Caption=A(2)
LabelA0.Caption=A(3)
LabelTimestamp.Caption=timestamp
End Sub
```

Remarks

The **EventMessage** events are generated only for the EVENTDATA type messages. For the EVENT messages *ActiveUSB* will fire [EventMessage](#) events.

For more information on the USB3 Vision messaging refer to "*USB3 Vision Camera Interface Standard For Machine Vision*" published by the Automated Imaging Association.

3.3.6 FormatChanged

Description

This event is fired each time the frame size or pixel format of the camera changes.

Syntax

```
[VB]  
Private Sub objActiveUSB_FormatChanged()
```

```
[C/C++]  
HRESULT Fire_FormatChanged();
```

Parameters [C/C++]

None

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example uses the **FormatChanged** event to generate a sound signal:

```
Private Sub ActiveUSB1_FormatChanged()  
Beep  
End Sub
```

Remarks

The **FrameDropped** event is raised when the frame size or pixel format of the camera changes. Your application may use this event to perform certain actions when the video format is changed through the Property Pages of ActiveUSB. See [ShowProperties](#) for more details.

3.3.7 FrameAcquired

Description

This event is fired each time a frame has been acquired, decoded and processed.

Syntax

```
[VB]  
Private Sub objActiveUSB_FrameAcquired()
```

```
[C/C++]  
HRESULT Fire_FrameAcquired();
```

Parameters [C/C++]

None

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example uses the **FrameAcquired** event to access and display a pixel value in real time:

```
Private Sub ActiveUSB1_FrameAcquired()  
Label1.Caption = ActiveUSB1.GetPixel(16, 32)  
End Sub
```

Remarks

One of the following conditions must be met, before the FrameAcquired event will be fired:

- The [Acquire](#) property has been set to TRUE
- The [Grab](#) method has been called.

The **FrameAcquired** event is fired from the interface thread to provide compatibility with all types of ActiveX containers. For applications created in VB.NET, C# and C++ it is recommended to use the [FrameAcquiredX](#) event.

3.3.8 FrameAcquiredX

Description

This event is fired each time a frame has been acquired, decoded and processed. Unlike [FrameAcquired](#) event it is fired from a processing thread providing a higher efficiency. May not work in certain containers.

Syntax

```
[VB]
Private Sub objActiveUSB_FrameAcquiredX()
```

```
[C/C++]
HRESULT Fire_FrameAcquiredX();
```

Parameters

None

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB.NET example uses the **FrameAcquiredX** event to access and display a pixel value in real time:

```
Public Class Form1

    Delegate Sub UpdateFPSLabelCallback([text] As String)

    Private Sub AxActiveUSB1_FrameAcquiredX(sender As Object, e As
System.EventArgs) Handles AxActiveUSB1.FrameAcquiredX
        UpdateLabel(Format(AxActiveUSB1.GetPixel(16, 32)))
    End Sub

    Private Sub UpdateLabel(ByVal [text] As String)
        If Me.Label1.InvokeRequired Then
            Dim d As New UpdateLabelCallback(AddressOf UpdateLabel)
            Me.Invoke(d, New Object() {[text]})
        Else
            Me.Label1.Text = [text]
        End If
    End Sub

End Class
```

Remarks

This event is provided for applications that can process events fired from a processing thread (VB.NET, C#, C++). For applications created in VB6, VBA, Delphi and Matlab use [FrameAcquired](#)

event instead.

One of the following conditions must be met, before the **FrameAcquiredX** event will be fired:

- The [Acquire](#) property has been set to TRUE

- The [Grab](#) method has been called.

3.3.9 FrameDropped

Description

This event is fired each time a frame is dropped during the video acquisition.

Syntax

```
[VB]  
Private Sub objActiveUSB_FrameDropped()
```

```
[C/C++]  
HRESULT Fire_FrameDropped();
```

Parameters [C/C++]

None

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example uses the **FrameDropped** event to generate a sound signal:

```
Private Sub ActiveUSB1_FrameDropped()  
Beep  
End Sub
```

Remarks

The **FrameDropped** event is raised if an image frame is dropped during the data transmission. The most common reasons for this are the loss of UDP packets due to the network overload, extensive image processing of each frame by an application and limited bandwidth during the AVI recording.

3.3.10 FrameLoaded

Description

This event is fired each time a frame has been loaded from a video file or memory sequence during the playback ([PlayVideo](#)). It is also fired when the image is loaded from an image file ([LoadImage](#)).

Syntax

[VB]

```
Private Sub objActiveUSB_FrameLoaded(ByVal Frame As Long)
```

[C/C++]

```
HRESULT Fire_FrameLoaded();
```

Data Types [VB]

Frame: long

Parameters [C/C++]

Frame

The zero-based index of the last loaded frame.

Return Values

S_OK

Success

E_FAIL

Failure.

Example

This VB example uses the **FrameLoaded** event to update a sequence frame counter:

```
Private Sub ActiveUSB1_FrameLoaded((ByVal Frame As Long))  
    Label1.Caption = Frame  
End Sub
```

Remarks

3.3.11 FrameRecorded

Description

This event is fired each time a frame has been added to a memory sequence ([StartSequenceCapture](#)) or video file ([StartCapture](#)). It is also fired when an image file has been saved ([SaveImage](#), [SaveBkg](#)).

Syntax

[VB]

```
Private Sub objActiveUSB_FrameRecorded(ByVal Frame As Long)
```

[C/C++]

```
HRESULT Fire_FrameRecorded();
```

Data Types [VB]

Frame: long

Parameters [C/C++]

Frame

The zero-based index of the last recorded frame.

Return Values

S_OK

Success

E_FAIL

Failure.

Example

This VB example uses the **FrameRecorded** event to update a sequence frame counter:

```
Private Sub ActiveUSB1_FrameRecorded(ByVal Frame As Long)
    Labell.Caption = Frame
End Sub
```

Remarks

3.3.12 FrameReady

Description

This event is fired each time a decoded frame is submitted for display.

Syntax

[VB]

```
Private Sub objActiveUSB_FrameReady()
```

[C/C++]

```
HRESULT Fire_FrameReady();
```

Parameters [C/C++]

None

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This MFC fragment uses the **FrameReady** event to perform a post-processing of the color video:

```
void CUcamProDlg::OnFrameReadyActiveUSB1()  
{  
    BYTE *ptr=m_ActiveUSB.GetImagePointer(0, m_ActiveUSB.GetSizeY()-1);  
    int nSize=m_ActiveUSB.GetSizeX()*m_ActiveUSB.GetSizeY();  
    int max=255;  
    for(int y=0;y<nSize;y++,ptr++)  
        *ptr=( *ptr+16)*2;  
}
```

Remarks

The **FrameReady** event is fired from *ActiveUSB*'s display thread which is running separately from the acquisition thread. This allows your post-processing routine to take advantage of a multi-processor or multi-core architecture. Note that If the post-processing procedure is not fast enough, it will only affect the display frame rate and will not result in missed frames in the acquisition thread.

In order to synchronize the post-processing with image display, the [Display](#) property should be set to False and [Draw](#) method used in the end of the event handler.

One of the following conditions must be met, before the **FrameReady** event will be fired:

The [Acquire](#) property has been set to TRUE

The [Grab](#) method has been called.

Note - this event may not work in older development environments such as VB6

3.3.13 MouseDblClick

Description

This event is fired each time the mouse button is double-clicked inside the control window. Returns the coordinates of a pixel pointed by the cursor.

Syntax

[VB]

```
Private Sub objActiveUSB_MouseDblClick( ByVal X As Integer, ByVal Y As Integer )
```

[C/C++]

```
HRESULT Fire_MouseDblClick( SHORT X, SHORT Y );
```

Data Types [VB]

X: Integer

Y: Integer

Parameters [C/C++]

X [in]

The X-coordinate of the pixel pointed by the cursor.

Y [in]

The Y-coordinate of the pixel pointed by the cursor.

Return Values

S_OK

Success

E_FAIL

Failure.

Example

This VB example uses the **MouseDblClick** event to open the *Properties* dialog:

```
Private Sub ActiveUSB1_MouseDblClick(ByVal X As Integer, ByVal Y As Integer)
    Dim Value As Integer
    Value = ActiveUSB1.ShowProperties
End Sub
```

Remarks

Note that the coordinates returned by this event refer to the image coordinate system, not to the screen coordinates.

Set the [Acquire](#) and [ScrollBars](#) properties to TRUE in order to display a scrollable live video and get a visual access to all parts of the image.

3.3.14 MouseDown

Description

This event is fired each time the mouse button is pressed inside the control window. Returns the coordinates of a pixel pointed by the cursor.

Syntax

[VB]

```
Private Sub objActiveUSB_MouseDown( ByVal X As Integer, ByVal Y As Integer )
```

[C/C++]

```
HRESULT Fire_MouseDown( SHORT X, SHORT Y );
```

Data Types [VB]

X: Integer

Y: Integer

Parameters [C/C++]

X [in]

The X-coordinate of the pixel pointed by the cursor.

Y [in]

The Y-coordinate of the pixel pointed by the cursor.

Return Values

S_OK

Success

E_FAIL

Failure.

Example

This VB example uses the **MouseDown** event to display the value of the pixel pointed by the cursor:

```
Private Sub ActiveUSB1_MouseDown(ByVal X As Integer, ByVal Y As Integer)
    Dim Value As Integer
    Value = ActiveUSB1.GetPixel(X, Y)
    MsgBox Value
End Sub
```

Remarks

Note that the coordinates returned by this event refer to the image coordinate system, not to the screen coordinates.

Set the [Acquire](#) and [ScrollBars](#) properties to TRUE in order to display a scrollable live video and get a visual access to all parts of the image.

3.3.15 MouseDownRight

Description

This event is fired each time the right mouse button is pressed inside the control window. Returns the coordinates of a pixel pointed by the cursor.

Syntax

[VB]

```
Private Sub objActiveUSB_MouseDownRight( ByVal X As Integer, ByVal Y As Integer )
```

[C/C++]

```
HRESULT Fire_MouseDownRight( SHORT X, SHORT Y );
```

Data Types [VB]

X: Integer

Y: Integer

Parameters [C/C++]

X [in]

The X-coordinate of the pixel pointed by the cursor.

Y [in]

The Y-coordinate of the pixel pointed by the cursor.

Return Values

S_OK

Success

E_FAIL

Failure.

Example

This VB example uses the **MouseDownRight** event to display the value of the pixel pointed by the cursor:

```
Private Sub ActiveUSB1_MouseDownRight(ByVal X As Integer, ByVal Y As Integer)
    Dim Value As Integer
    Value = ActiveUSB1.GetPixel(X, Y)
    MsgBox Value
End Sub
```

Remarks

Note that the coordinates returned by this event refer to the image coordinate system, not to the screen coordinates.

Set the [Acquire](#) and [ScrollBars](#) properties to TRUE in order to display a scrollable live video and get a

visual access to all parts of the image.

3.3.16 MouseMove

Description

This event is fired each time the mouse has moved inside the control window. Returns the coordinates of a pixel pointed by the cursor.

Syntax

[VB]

```
Private Sub objActiveUSB_MouseMove( ByVal X As Integer, ByVal Y As Integer )
```

[C/C++]

```
HRESULT Fire_MouseMove( SHORT X, SHORT Y );
```

Data Types [VB]

X: Integer

Y: Integer

Parameters [C/C++]

X [in]

The X-coordinate of the pixel pointed by the cursor.

Y [in]

The Y-coordinate of the pixel pointed by the cursor.

Return Values

S_OK

Success

E_FAIL

Failure.

Example

This VB example uses the **MouseMove** event to display the value of the pixel pointed by the cursor:

```
Private Sub ActiveUSB1_MouseMove(ByVal X As Integer, ByVal Y As Integer)
    Dim Value As Integer
    Value = ActiveUSB1.GetPixel(X, Y)
    Label1.Caption = Value
End Sub
```

Remarks

Note that the coordinates returned by this event refer to the image coordinate system, not to the screen coordinates.

Set the [Acquire](#) and [ScrollBars](#) properties to TRUE in order to display a scrollable live video and get a visual access to all parts of the image.

3.3.17 MouseUp

Description

This event is fired each time the mouse button is released inside the control window. Returns the coordinates of a pixel pointed by the cursor.

Syntax

[VB]

```
Private Sub objActiveUSB_MouseUp( ByVal X As Integer, ByVal Y As Integer )
```

[C/C++]

```
HRESULT Fire_MouseUp( SHORT X, SHORT Y );
```

Data Types [VB]

X: Integer

Y: Integer

Parameters [C/C++]

X [in]

The X-coordinate of the pixel pointed by the cursor.

Y [in]

The Y-coordinate of the pixel pointed by the cursor.

Return Values

S_OK

Success

E_FAIL

Failure.

Example

This VB example uses the **MouseUp** event to display the value of the pixel pointed by the cursor:

```
Private Sub ActiveUSB1_MouseUp(ByVal X As Integer, ByVal Y As Integer)
    Dim Value As Integer
    Value = ActiveUSB1.GetPixel(X, Y)
    MsgBox Value
End Sub
```

Remarks

Note that the coordinates returned by this event refer to the image coordinate system, not to the screen coordinates.

Set the [Acquire](#) and [ScrollBars](#) properties to TRUE in order to display a scrollable live video and get a visual access to all parts of the image.

3.3.18 MouseUpRight

Description

This event is fired each time the rightmouse button is released inside the control window. Returns the coordinates of a pixel pointed by the cursor.

Syntax

[VB]

```
Private Sub objActiveUSB_MouseUpRight( ByVal X As Integer, ByVal Y As Integer )
```

[C/C++]

```
HRESULT Fire_MouseUpRight( SHORT X, SHORT Y );
```

Data Types [VB]

X: Integer

Y: Integer

Parameters [C/C++]

X [in]

The X-coordinate of the pixel pointed by the cursor.

Y [in]

The Y-coordinate of the pixel pointed by the cursor.

Return Values

S_OK

Success

E_FAIL

Failure.

Example

This VB example uses the **MouseUpRight** event to display the value of the pixel pointed by the cursor:

```
Private Sub ActiveUSB1_MouseUpRight(ByVal X As Integer, ByVal Y As Integer)
    Dim Value As Integer
    Value = ActiveUSB1.GetPixel(X, Y)
    MsgBox Value
End Sub
```

Remarks

Note that the coordinates returned by this event refer to the image coordinate system, not to the screen coordinates.

Set the [Acquire](#) and [ScrollBars](#) properties to TRUE in order to display a scrollable live video and get a visual access to all parts of the image.

3.3.19 PlayCompleted

Description

This event is fired when the playback of the memory sequence or video file is stopped.

Syntax

[VB]

```
Private Sub objActiveUSB_PlayCompleted(ByVal Frames As Long)
```

[C/C++]

```
HRESULT Fire_PlayCompleted(long Frames);
```

Data Types [VB]

Frames: long

Parameters [C/C++]

Frames

The number of frames recorded.

Example

This VB example opens the memory sequence, plays it and uses the **PlayCompleted** event to turn the live video on.

```
Private Sub Play_Click()  
ActiveUSB1.OpenVideo "ram"  
ActiveUSB1.PlayVideo  
End Sub
```

```
Private Sub ActiveUSB1_PlayCompleted(ByVal Frames As Long)  
ActiveUSB1.CloseVideo  
ActiveUSB1.Acquire=True  
End Sub
```

Remarks

One of the following conditions must be met, before the **PlayCompleted** event will be fired:

Playback has stopped automatically, because the specified number of frames has been reached.

Playback has been stopped explicitly by calling [StopVideo](#).

3.3.20 RawFrameAcquired

Description

This event is fired each time a raw frame has arrived from the camera, before it is submitted to *ActiveUSB* decoding and processing chain. Can be used in combination with [GetRawData](#) to perform a preprocessing on raw images.

Syntax

[VB]

```
Private Sub objActiveUSB_RawFrameAcquired()
```

[C/C++]

```
HRESULT Fire_RawFrameAcquired();
```

Parameters [C/C++]

None

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This MFC fragment uses the **RawFrameAcquired** event to perform a preprocessing (invert operation) of the raw video:

```
void CUcamProDlg::OnRawFrameAcquiredActiveUSB1()
{
    VARIANT v=m_ActiveUSB.GetRawData(true);
    int nSize=m_ActiveUSB.GetSizeX()*m_ActiveUSB.GetSizeY();
    BYTE* ptr=(BYTE*)v.pvData;
    int max=255;
    for(int y=0;y<nSize;y++,ptr++)
        *ptr=max-*ptr;
}
```

Remarks

One of the following conditions must be met, before the **RawFrameAcquired** event will be fired:

- The [Acquire](#) property has been set to TRUE
- The [Grab](#) method has been called.

The internal ActiveUSB processing chain will be blocked until the container application releases the event handler. Therefore the displayed frame rate can drop and frames can get missed, if the external pre-processing procedure is not fast enough.

3.3.21 Scroll

Description

This event is fired each time the live video display has been scrolled. Returns the horizontal and vertical scroll positions.

Syntax

[VB]

```
Private Sub objActiveUSB_Scroll( ByVal X As Integer, ByVal Y As Integer )
```

[C/C++]

```
HRESULT Fire_Scroll( SHORT ScrollX, SHORT ScrollY );
```

Data Types [VB]

ScrollX: Integer

ScrollY: Integer

Parameters [C/C++]

ScrollX [in]

The X-coordinate of the pixel pointed by the cursor.

ScrollY [in]

The Y-coordinate of the pixel pointed by the cursor.

Return Values

S_OK

Success

E_FAIL

Failure.

Example

This VB example uses the **Scroll** event to display the position of the live video in the control window:

```
Private Sub ActiveUSB1_Scroll(ByVal ScrollX As Integer, ByVal ScrollY As Integer)
    Label1.Caption = ScrollX
    Label2.Caption = ScrollY
End Sub
```

Remarks

Note that the scroll positions returned by this event refer to the image coordinate system, not to the screen coordinates.

The [ScrollBars](#) properties to TRUE in order for the **Scroll** event to be fired.

3.3.22 Timeout

Description

This event is fired each time a timeout occurs during the acquisition of a frame.

Syntax

[VB]

```
Private Sub objActiveUSB_Timeout()
```

[C/C++]

```
HRESULT Fire_Timeout();
```

Parameters [C/C++]

None

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example uses the **Timeout** event to generate a sound signal:

```
Private Sub ActiveUSB1_Timeout()  
Beep  
End Sub
```

Remarks

The **Timeout** event is raised when the amount of seconds specified by the [Timeout](#) property passes after the [Grab](#) method has been called and no frame has been acquired. The event will be raised repeatedly if the control is set in the continuous acquisition mode ([Acquire](#) is TRUE). The most common reason for a timeout is a missing trigger signal in the [Trigger](#) mode.

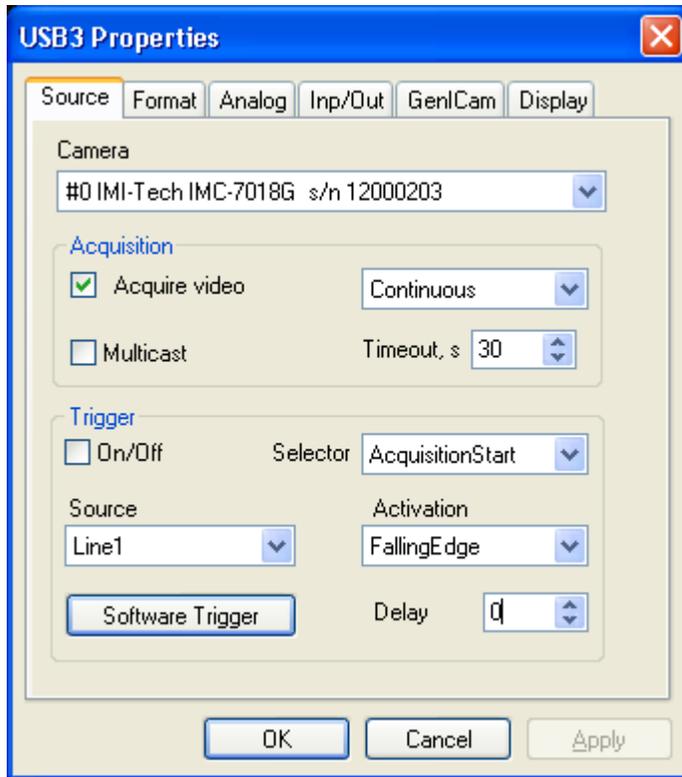
3.4 Property Pages

The following property pages are available in *ActiveUSB* control:

Source	Used to select the properties specifying the source for the video input
Format	Used to select the properties specifying the format of the video
Analog	Used to select the properties specifying the analog features of the video
Input/Output	Used to select the properties specifying the input/output features of the video
GenICam	Provides an access to the complete GenICam feature set of the camera
Display	Used to select the properties specifying the display settings

3.4.1 Source

This property page is used to select the properties specifying the source for the video input.



Select from the following options:

Camera

Displays the vendor's name and model of the currently selected camera. If you have more than one USB3 Vision™ camera connected to your system, you can switch to another camera by choosing the corresponding camera name in the list. Equivalent to the [Camera](#) property.

Acquire

Lets you enable the continuous acquisition mode. If this box is checked, the camera will continuously acquire the video into the internal image memory. If the control is visible, the live video will be displayed in the control window. Equivalent to the [Acquire](#) property.

Timeout

Use this option to set the number of seconds to wait for a frame to be acquired. Typically used to assign the timeout when the [Trigger](#) mode is active. If the timeout expires, the [Timeout event](#) event will be raised. Equivalent to the [Timeout](#) property.

Acquisition mode

Lets you select the desired acquisition mode from the list. The acquisition mode defines how many frames will be captured when the **Acquire** option is enabled. Equivalent to the [AcquisitionMode](#) property.

Trigger Source

Lets you select the configuration of a trigger signal. The rest of the options in the Trigger group are dependent on this selection. Equivalent to the [TriggerSelector](#) property.

Trigger

Check this box to enable the selected trigger. This mode is typically used with an asynchronously resettable camera. An acquisition will occur upon receiving a signal from an external hardware **Trigger Source**. Equivalent to the [Trigger](#) property.

Trigger Source

Lets you select the source for the selected trigger. Depending on a camera, there may be one or more hardware trigger inputs as well as the software trigger. If the software trigger is available and selected, use the **Software Trigger** button to simulate the trigger event. If the camera doesn't support trigger source selection, this option will be unavailable. Equivalent to the [TriggerSource](#) property and [SoftTrigger](#) method.

Trigger activation

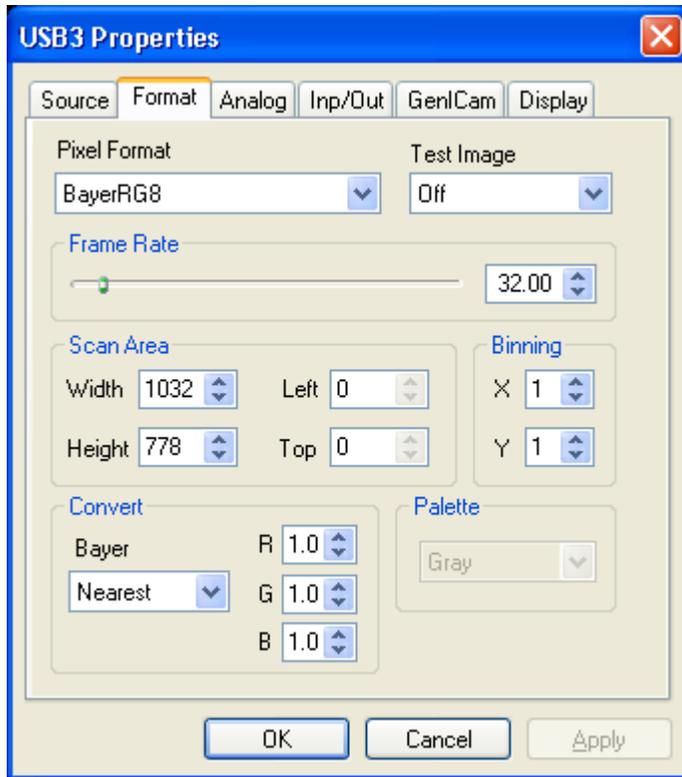
Lets you change the activation mode (trigger signal polarity) for the selected trigger. Equivalent to the [TriggerActivation](#) property.

Trigger delay

Lets you select the value for the trigger delay which defines the time between the arrival of the trigger signal and its effective activation. Equivalent to the [TriggerDelay](#) property.

3.4.2 Format

This property page is used to select the properties specifying the format of the video stream.



Select from the following options:

Pixel Format

Use this option to select the desired pixel format from the list of modes available for the current camera. Equivalent to the [Format](#) property.

Frame rate

Use this option to set the acquisition frame rate in frames per seconds. Note that the actual frame rate can also depend on the exposure time. Equivalent to the [AcquisitionFrameRate](#) property. If the camera doesn't support this property, the **Frame rate** option will be unavailable.

Scan Area

Lets you change the size and position of the image window on the camera's sensor. To modify the size and position of the window, enter the desired values for the image width, height, left coordinate and top coordinate in pixels. Equivalent to the [SizeX](#), [SizeY](#), [OffsetX](#), [OffsetY](#) properties.

Binning

Lets you change the number of horizontal and vertical photo-sensitive cells that must be combined together. This property has a net effect of increasing the intensity (or signal to noise ratio) of the pixel value and reducing the horizontal size of the image. If the camera doesn't support the binning, this option will be unavailable. Equivalent to the [BinningX](#) and [BinningY](#) properties.

Bayer

Select this option to activate the real-time color conversion of a monochrome raw video generated by a Bayer camera and select the specific Bayer conversion algorithm. Select one of the following

options:

None - Bayer conversion is disabled. The camera will output a monochrome raw image.

Nearest - Nearest Neighbour filter. Missing pixels are substituted with adjacent pixels of the same color.

Bilinear - Bilinear filter. Calculates the values of missing pixels by performing bilinear interpolation of the adjacent pixels.

Bilinear HQ - High Quality Linear filter. Calculates the values of missing pixels based on the Malvar, He and Cutler algorithm.

See the [Bayer](#) property for more information.

R, G, B, Y

Lets you adjust the gain factors for individual color channels or the intensity factor for a monochrome video. Equivalent to the [SetGains](#) property.

Palette

Lets you select one of a few predefined palette to be applied to a grayscale live video. The palettes represent choices that may be useful in viewing different kinds of video in pseudo-colors. Choose among the following palettes:

Gray

Applies the standard 256-level grayscale palette. This is a regular mode of viewing a grayscale video.

Inverse

Applies the inverted 256-level grayscale palette. The video will be displayed in the negative format.

Saturated

Applies the grayscale palette with colored upper entries. The saturated palette allows you to control the dynamic range of the video signal by bringing it slightly below the saturation level of the video camera or video amplifier. To achieve the maximum dynamic range, adjust the intensity of the light source and/or the gain and zero level of the video amplifier so that the red color corresponding to the brightest pixel values just barely shows up.

Rainbow

Applies a color palette where the entries are evenly distributed along the Hue axis. This allows for assigning different color pigments to different levels of intensity.

Spectra

Applies a color palette where the entries are distributed along the Hue and Luminance axes. That allows for assigning different color pigments to different levels of intensity while preserving the luminance scale.

Isodense

Applies the 256-level grayscale palette, each 8-th entry of which is colored. The isodense palette allows you to clearly see transitions between different levels of intensities as isolines on a topographic map.

Multiphase

Applies the multiphase palette. Entries in the multiphase palette are at opposite ends of the color model so even small changes in gray levels are highlighted.

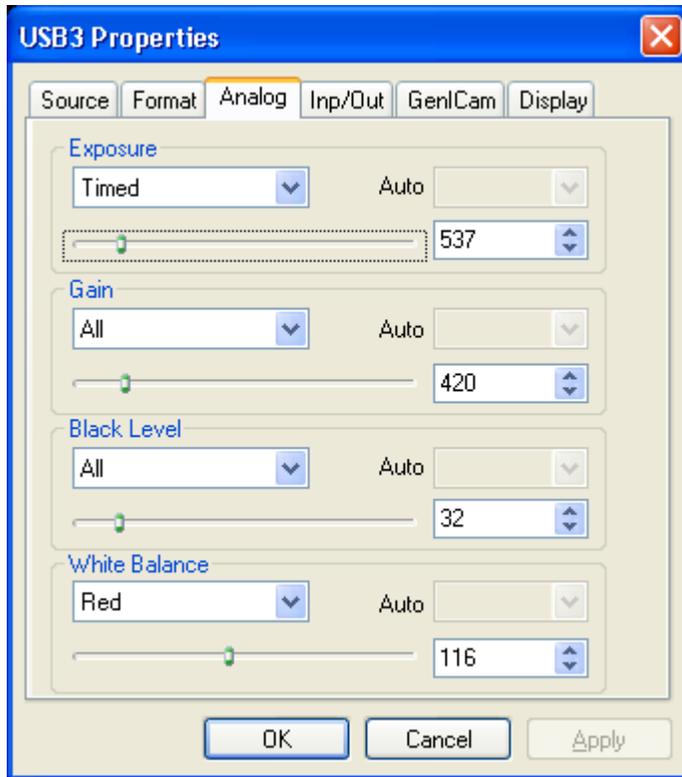
Random

Applies the random color palette whose entries are filled with random values each time you select it from the list.

This option is equivalent to the [Palette](#) property.

3.4.3 Analog

This property page is used to select the properties specifying the analog features of the camera.



Select from the following options:

Exposure Mode

Lets you select the operation mode of the exposure (shutter). See [ExposureMode](#) for more details.

Exposure Auto

Lets you select between the automatic (AE) or manual exposure control mode. Depending on the camera, the following selections can be available:

- Off* - exposure is manually controlled using **Exposure Value** option.
- Once* - the camera sets the optimal exposure level and returns to the "Off" state
- Continuous* - the camera constantly adjusts the exposure level

The availability and possible values of this option depend on the [ExposureAuto](#) property.

Exposure Value

Use the slider and spin controls to adjust the integration time of the incoming light. You can also enter the desired exposure value in the corresponding text box. Note that this option is available only for the cameras that support the manual exposure control. Equivalent to the [ExposureTime](#) property.

Gain Selector

Lets you select the color channel to be controlled by the **Gain Value** and **Gain Auto** options. See [GainSelector](#) for more details.

Gain Auto

Lets you select between the automatic (AGC) or manual gain control mode. Depending on the camera, the following selections can be available:

- Off* - gain is manually controlled using **Gain Value** option
- Once* - the camera sets the optimal gain level and returns to the "Off" state
- Continuous* - the camera constantly adjusts the gain level

The availability and possible values of this option depend on the [GainAuto](#) property.

Gain Value

Use the slider and spin controls to adjust the camera's video signal amplification. You can also enter the desired gain value in the corresponding text box. Note that this option is available only for the cameras that support the manual gain control. Equivalent to the [Gain](#) property.

Black Level Selector

Lets you select the color channel to be controlled by the **Black Level Value** and **Black Level Auto** options. See [BlackLevelSelector](#) for more details.

Black Level Auto

Lets you select between the automatic or manual black level control mode. Depending on the camera, the following selections can be available:

- Off* - black level is manually controlled using **Black Level Value** option
- Once* - the camera sets the optimal black level and returns to the "Off" state
- Continuous* - the camera constantly adjusts the black level

The availability and possible values of this option depend on the [BlackLevelAuto](#) property.

Black Level Value

Use the slider and spin controls to adjust the black level (brightness) of the video signal. You can also enter the desired black level value in the corresponding text box. Note that this option is available only for the cameras that support the manual black level control. Equivalent to the [BlackLevel](#) property.

White Balance Selector

Lets you select the color channel to be controlled by the **White Balance Ratio** option. See [BalanceRatioSelector](#) for more details.

White Balance Auto

Lets you select between the automatic (AWB) or manual white balance control mode. Depending on the camera, the following selections can be available:

- Off* - balance ration for a selected color channel is manually controlled using **White Balance Ratio** option.
- Once* - the camera sets the optimal white balance level and returns to the "Off" state
- Continuous* - the camera constantly adjusts the white balance level

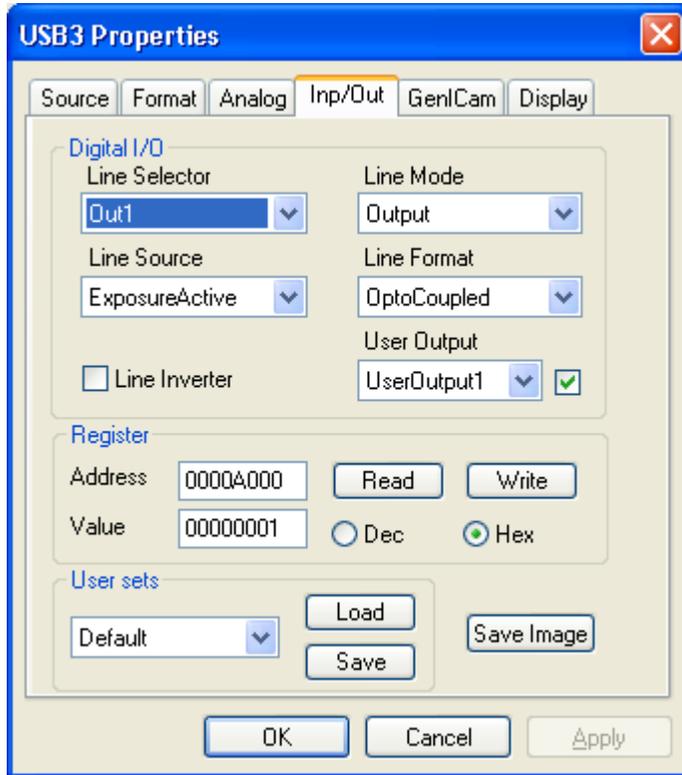
The availability and possible values of this option depend on the [BalanceWhiteAuto](#) property.

White Balance Ratio

Use the slider and spin controls to adjust the ratio (amplification factor) of the selected color component. You can also enter the desired balance ratio in the corresponding text box. Note that this option is available only for the cameras that support the manual balance ratio control. Equivalent to the [BalanceRatio](#) property.

3.4.4 Inp/Out

This property page is used to select the properties specifying input/output operations.



Select from the following options:

Line Selector

Lets you select the physical line (or pin) of the external device connector to configure. Equivalent to the [LineSelector](#) property.

Line Mode

Lets you select the input or output mode for the currently selected line. Equivalent to the [LineMode](#) property.

Line Source

Lets you select the source of the signal to output on the selected line when the line mode is *Output*. Equivalent to the [LineSource](#) property.

Line Format

Lets you select the electrical format (TTL, LVDS, OptoCoupled etc) of the selected line. Equivalent to the [LineFormat](#) property.

Line Inverter

Select this option to have the electrical signal on the selected line inverted. Equivalent to the [LineInverter](#) property.

User Output

Lets you configure the bits of the User Output register. Checking/unchecking the box will set the selected bit to the High or Low state respectively. Equivalent to the [UserOutputSelector](#) and [UserOutputValue](#) properties.

Register

Lets you perform reads and writes to a selected register in the camera bootstrap address space. To perform the read operation, enter the desired hexadecimal address to the **Address** field and click the **Read** button. The result will be displayed in the **Value** box. Depending on the **Dec/Hex** radio boxes, the result will be displayed either in decimal or hexadecimal form. To perform the write operation, enter a desired address and value to the **Address** and **Value** fields respectively, and then press the **Write** button. Equivalent to the [ReadRegister](#), [WriteRegister](#) methods.

User set

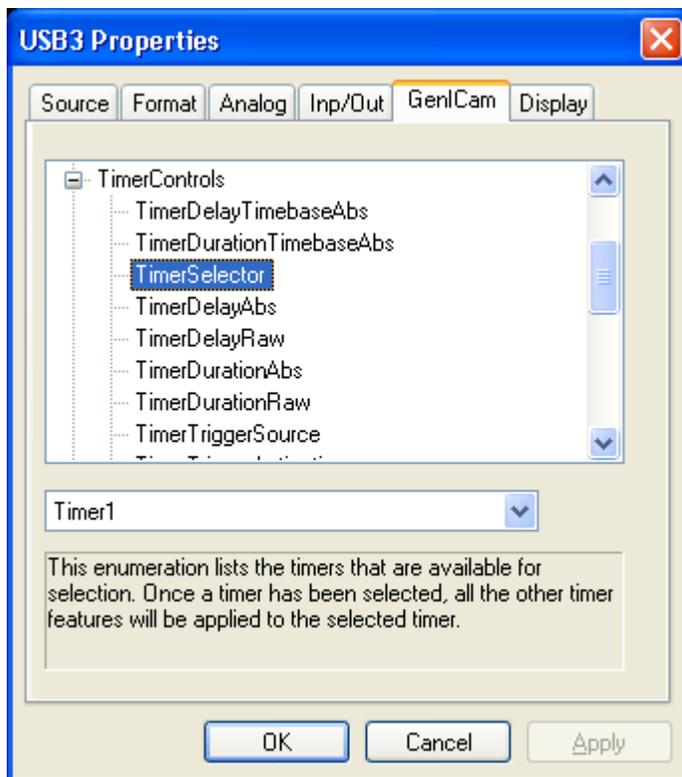
Lets you load or store camera settings under the specified user set. Use the list box to select the desired user set. Use the Save button to store the current camera settings in the selected set. Use the Load button to load the setting from the selected set into the camera. See [UserSetSelector](#) for details.

Save image

Lets you save the current frame buffer in an image file. When you click this button, the **Save As** dialog box will appear where you can select the file name and one of the image file formats: RAW, BMP, TIF, JPEG or DPX. Note that BMP and TIF files will be recorded with no compression while JPEG files will be recorded with quality 75. DPX files are recorded in the 10-bit packed RGB or mono format. Equivalent to the [SaveImage](#) method.

3.4.5 GenICam

This property page is used to access all the camera features per GenICam standard.



Select from the following options:

GenICam Tree

Shows all the camera features sorted by categories as reported by the camera's XML file. USB3 Vision™ standard, which is a subset of the GenICam standard, uses XML files to expose all the commands and features available for a camera. To access a specific feature, browse through the categories and highlight the desired feature.

Feature Control

Depending on the type of the feature highlighted in the tree, the following options will become available:

Feature type	GUI control	Action
Integer	Text box + spin control	Adjust the feature value or enter the desired value in the text box
Float	Text box + spin control	Adjust the feature value or enter the desired value in the text box
Boolean	List box	Select "On" or "Off" value
Enumerated	List box	Select among available feature values
Command	Push button	Click the button to execute the command

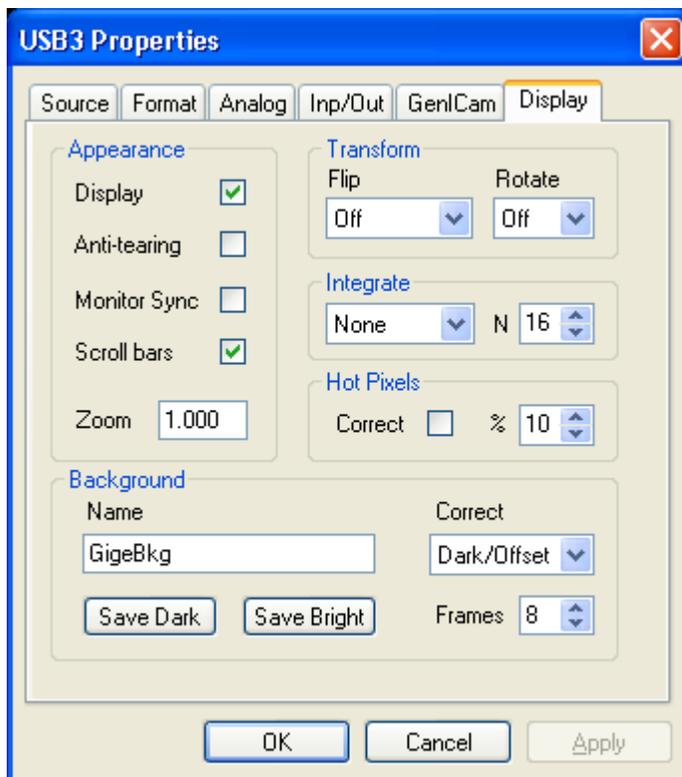
If the currently selected feature is read-only, the corresponding control will be disabled. If the feature is currently unavailable, the text box will display "Unavailable".

Feature Description

Shows the description of the currently selected feature. The availability of the feature description depends on the camera XML implementation.

3.4.6 Display

This property page is used to select the properties specifying the display settings of *ActiveUSB*.



Select from the following options:

Display

Lets you enable or disable live display in the control window. You might want to disable the display option if you want to render captured frames by other means such as [Picture box](#). When using [DirectShow Video Capture Filter](#), setting this property to TRUE will activate the internal RGB24 conversion which is required by image data access and image analysis methods. Equivalent to the [Display](#) property.

Anti-tearing

Lets you enable or disable the anti-tearing feature. Anti-tearing removes horizontal tears in the live video caused by a difference between the camera frame rate and refresh rate of the monitor. Equivalent to the [AntiTearing](#) property.

Monitor Sync

Lets you enable or disable the monitor synchronization mode. If this box is checked, the camera frame rate will exactly match the refresh rate of the system monitor thus eliminating all the display artifacts related to the digital image transfer. Equivalent to the [MonitorSync](#) property.

Scroll bars

Lets you enable or disable the scroll bars in the control window. If this box is checked and the video width or/and height exceed the size of the control window, the scroll bar(s) will be displayed on the border of the control window allowing you to pan the live video. Equivalent to the [ScrollBars](#) property.

Digital zoom

Lets you adjust the magnification of the live video display. This option doesn't change the content of the image data, but only its appearance in the control window. If it is set to zero, the image will be fit to the size of the control window. In this case the display might not retain the original proportions of the video frame. Equivalent to the [Magnification](#) property. *Note - if the Display option is unchecked and Digital zoom is set to zero, ActiveUSB will enter a raw image transfer mode with the minimal CPU usage.*

Flip

Lets you select one of the flipping modes. Flipping affects the live video display as well as actual order of pixels in the frame buffer. Select among the following modes:

Off

No image flipping is performed.

Horizontal

The image is flipped horizontally.

Vertical

The image is flipped vertically.

Diagonal

The image is flipped horizontally and vertically.

This option is equivalent to the [Flip](#) property.

Rotate

Lets you rotate the image. Rotation affects the live video display as well as actual order of pixels in the frame buffer. Select one of the following modes:

Off

No image rotation is performed.

90°

The image is rotated counterclockwise.

180°

The image is rotated 180 degrees.

270°

The image is rotated clockwise.

This option is equivalent to the [Rotate](#) property.

Integrate Mode

Lets you select the frame integration operation mode. The frame integration allows you to average or add frames "on the fly" without sacrificing the frame rate. Equivalent to the [Integrate](#) property. Select one of the following modes:

None

Frame integration is disabled.

Average

Running Average mode. Each output frame is the result of averaging a selected number of previously captured frames.

Add

Running Accumulation mode. Each output frame is the sum of a selected number of previously captured frames.

Integrate Window (N)

Sets the number of frames for the integration. Equivalent to the [IntegrateWnd](#) property.

Hot Pixel Correct

Lets you enable or disable the hot pixel correction mode. If this box is checked, unusually bright

pixels will be effectively removed from the image. Hot pixels are associated with elements on a camera sensor that have higher than normal rates of charge leakage. For more details on hot pixel correction refer to [HotPixelCorrect](#).

Hot Pixel Level (%)

Sets the hot pixel correction level, in percent. Equivalent to the [HotPixelLevel](#) property.

Background name

Lets you enter the name prefix under which the background files are stored. See [BkgName](#) for more details.

Save Dark

Click this button to store a dark field background image on the hard drive. Dark field should be saved when no light transmitted through the camera lens. The dark field will be calculated by averaging the number of consecutive frames specified by the **Frames** option. Equivalent to the [SaveBkg](#) method.

Save Bright

Click this button to store a bright field background image on the hard drive. Bright field should be saved with the maximum light transmitted and no objects in the field of view. To achieve the best dynamic range, the light intensity should be adjusted so that it stays just below the saturation level of the camera. To control the saturation for monochrome cameras, use the **Saturated Palette**. The bright field will be calculated by averaging the number of consecutive frames specified by the **Frames** option.

Correct

Lets you select a background correction mode. Select *None* if you do not want the background correction to be performed. Select *Dark/Offset* to apply the dark-field background correction to each frame captured. Select *Flat/Gain* to apply the flat-field background correction to each frame captured. Make sure to save the bright and dark fields before using this option, otherwise certain background correction modes will not be available. For more details on background correction refer to [BkgCorrect](#).

4 DirectShow

ActiveUSB SDK includes a *DirectShow Video Capture Filter* which allows Windows users to view/capture images from USB3 cameras and control camera parameters in DirectShow-compliant applications such as *AmCap*, *VirtualDub*, *Video Capturix* and others. The filter is automatically installed on your system during *ActiveUSB* [Installation](#).

The filter has an output *Capture Pin* which can be connected to other DirectShow filters or graph. The pin supports all video modes provided by USB3 Vision compliant cameras, with high-depth pixel modes automatically converted to 8-bit monochrome or 24-bit RGB format.

ActiveUSB Video Capture Filter provides a programmatic access to the video acquisition and camera properties through a set of standard and custom DirectShow interfaces. For more information on DirectShow programming refer to [DirectShow Quick Reference Guide](#) and [DirectShow Interfaces](#).

4.1 Quick Reference Guide

About DirectShow

Microsoft® DirectShow® application programming interface (API) is a media-streaming architecture for the Microsoft Windows® platform. It is embedded into the set of DirectX APIs. The purpose of this API is to capture, render and playback streaming media as Video or Audio streams. It manages both devices or files for capturing and rendering.

Filters and Filter Graphs:

The building block of DirectShow is a software component called a *filter*. A filter is a software component that performs some operation on a multimedia stream. For example, DirectShow filters can

- read files
- get video from a video capture device
- decode various stream formats, such as MPEG-1 video
- pass data to the graphics or sound card

Filters receive input and produce output through their pin(s). *ActiveUSB Video Capture Filter* provides a DirectShow interface to USB3 Vision compliant cameras. It has one output Video Capture pin which can be connected to other filters. The pin supports all video modes provided by a camera, with high-depth pixel modes automatically converted to 8-bit monochrome or 24-bit RGB format.

One or more of the pins may be connected in a chain, so that the output from one filter becomes the input for another. A set of connected filters is called a *filter graph*.

Filters expose various interfaces and media type. Interfaces are a filter's set of method for a specific task, for example the CaptureGraphBuilder is an interface of GraphBuilder filter. Media type identifies what kind of data the upstream filter will deliver to the downstream filter, and the physical layout of the data.

Depending on the camera video mode, *ActiveUSB Video Capture Filter* delivers the following media types:

If *Display* property of *IActiveUSB* interface is set to FALSE:

Mono 8:	MEDIASUBTYPE_RGB8
Mono 16:	MEDIASUBTYPE_RGB8
RGB 24:	MEDIASUBTYPE_RGB24
Mono 8 debayered:	MEDIASUBTYPE_RGB24
RGB 48:	MEDIASUBTYPE_RGB24
Mono 16 debayered:	MEDIASUBTYPE_RGB24
YUV422:	MEDIASUBTYPE_UYVU
YUV411:	MEDIASUBTYPE_Y411
YUV444:	MEDIASUBTYPE_AYUV

If *Display* property of *IActiveUSB* interface is set to TRUE:

For all video modes MEDIASUBTYPE_RGB24

Writing a DirectShow Application:

A limited set of DirectShow interfaces are compatible with Visual Basic, but primarily DirectShow is a C/C++ COM interface. In order to build applications that use *ActiveUSB Video Source Filter*, you will

need to install DirectX SDK and have a general understanding of COM client programming. A good starting point is Microsoft's Amcap sample application included in the DirectX SDK.

There are several tasks that any DirectShow application must perform.

- The application creates an instance of the Filter Graph Manager.
- The application uses the Filter Graph Manager to build a filter graph. The exact set of filters in the graph will depend on the application.
- The application uses the Filter Graph Manager to control the filter graph and stream data through the filters. Throughout this process, the application will also respond to events from the Filter Graph Manager.
- The application queries the interfaces exposed by the filters and uses their methods to control the properties of the filters.
- When processing is completed, the application releases the Filter Graph Manager and all of the filters.

Note that *ActiveUSB Video Source Capture Filter* provides two alternative ways of working with multiple cameras. By default, ActiveUSB installs one DirectShow device called "GigE Vision compliant camera". You can select a specific camera via the [Source](#) property page or through the [IActiveUSB](#) interface. Alternatively, you can use the [FilterConfig](#) utility to register several GigE Vision devices in the system, each one associated with a specific camera. Refer to [Working with multiple cameras](#) for more details.

For more information refer to the following topics:

[Retrieving the Filter](#)

[Building the Graph](#)

[Displaying the Preview](#)

[Capturing to AVI](#)

[Getting the Image Data](#)

[Displaying Property Pages](#)

4.1.1 FilterConfig utility

Provided with *ActiveUSB Video Capture Source Filter* is FilterConfig console utility located in the Driver subfolder of *ActiveUSB* folder (typically C:/Program Files/ActiveUSB/Driver). Depending on the platform on your DirectShow application, you should use either FilterConfig.exe or FilterConfig64.exe. The utility allows you to register several DirectShow devices when multiple GigE Vision cameras are used.

By default ActiveUSB registers a single DirectShow device called "*GigE Vision compliant camera*", in which case the selection of a specific camera is available via the [Source](#) property page or through the [ActiveUSB](#) interface. You can run several instances of this device in your system or in your application provided each instance is configured for a different camera. If you execute several copies of a DirectShow-based video capture application such as Microsoft's Amcap, each of them will attempt to automatically configure itself for a different camera and memorize corresponding camera settings in the system registry upon exiting. The same automatic configuration routine will apply to multiple instances of the "*GigE Vision compliant camera*" device running in one application.

In certain cases however you may want to have each camera associated with a separate DirectShow device in the system. This may be necessary when a third-party DirectShow application has to be used with several GigE Vision cameras. To register several GigE Vision devices in the system, run the FilterConfig executable with a numerical argument indicating the amount of GigE Vision cameras in the system. For example, the following command will register three DirectShow devices "*GigE Vision Device 1*", "*GigE Vision Device 2*", "*GigE Vision Device 3*":

```
C:\Program Files\ActiveUSB\Driver> FilterConfig.exe 3
```

You can change the number of GigE Vision DirectShow devices listed in the system by calling FilterConfig with a different parameter. To reset the filter to a default single "*GigE Vision compliant camera*" device, run FilterConfig with the zero argument:

```
C:\Program Files\ActiveUSB\Driver> FilterConfig.exe 0
```

Note: Before using FilterConfig to register multiple devices, you must enumerate your cameras by opening several instances of Microsoft's AmCap application. Each instance will automatically connect to the next available camera and will memorize its device index upon exiting. Alternatively the GcamCap application (located in the ActiveUSB/Bin folder) can be used for the same purpose.

4.1.2 Retrieving the Filter

To instantiate *ActiveUSB Video Capture Filter*, you need to enumerate all video capture filters using the system device enumerator and match the filter name with "USB3 Vision compliant camera":

```
IBaseFilter* pFilter;
HRESULT hr;

//locate the camera filter using system device enumerator
CComPtr< ICreateDevEnum > pCreateDevEnum;
pCreateDevEnum.CoCreateInstance( CLSID_SystemDeviceEnum );
if( !pCreateDevEnum )
return E_FAIL;
// enumerate video capture devices
CComPtr< IEnumMoniker > pEm;
pCreateDevEnum->CreateClassEnumerator( CLSID_VideoInputDeviceCategory, &pEm, 0 );
if( !pEm )
return E_FAIL;

pCreateDevEnum.Release();
pEm->Reset();
int noCapDevice=0;

while(true)
{
ULONG ulFetched = 0;
CComPtr< IMoniker > pM;
hr = pEm->Next( 1, &pM, &ulFetched );
if( hr != S_OK )
break;
//get the property bag interface from the moniker
CComPtr< IPropertyBag > pBag;
hr = pM->BindToStorage( 0, 0, IID_IPropertyBag, (void**)&pBag );
if( hr != S_OK )
{
pBag.Release();
continue;
}

//retrieve the name of the device
CComVariant var;
var.vt = VT_BSTR;
hr = pBag->Read(L"FriendlyName", &var, NULL );
if( hr != S_OK )
{
SysFreeString(var.bstrVal);
pBag.Release();
continue;
}

//is this ActiveUSB filter?
if( !memcmp( W2CA(var.bstrVal), "USB3 Vision compliant camera", 26 ) )
{
hr = pM->BindToObject( 0, 0, IID_IBaseFilter, (void**)pFilter );
}
noCapDevice++;
pM.Release();
}
```

```
pBag.Release();  
SysFreeString( var.bstrVal );  
}  
pEm.Release();
```

4.1.3 Building the Graph

DirectShow filters run in the context of the filter graph. The graph manages connections between filters from a source, such as a video capture filter, to a renderer, such as a video window, and any transformation filters in between. DirectShow provides **ICaptureGraphBuilder** and **ICaptureGraphBuilder2** interfaces containing methods for building and controlling a capture graph.

```
CComPtr< IGraphBuilder > pGraph;
CComPtr< ICaptureGraphBuilder2 > pBuilder;
HRESULT hr = S_OK;

//create filter graph
hr = pGraph.CoCreateInstance( CLSID_FilterGraph );
if( FAILED(hr) )
{
    Error( "Failed to create a filter graph." );
    return FALSE;
}

//create a capture graph builder
hr = pBuilder.CoCreateInstance( CLSID_CaptureGraphBuilder2 );
if( FAILED(hr) )
{
    Error( "Failed to create a capture graph builder." );
    return FALSE;
}

//add ActiveUSB filter to the graph
hr = pGraph->AddFilter(pFilter, L"USB3 Vision camera" );
if( FAILED(hr) )
{
    Error( "Failed to add the camera to graph." );
    return FALSE;
}

//specify the filter graph to the graph builder
hr = pBuilder->SetFiltergraph (pGraph);
```

4.1.4 Displaying the Preview

To build a video preview graph, call the *RenderStream* method of **ICaptureGraphBuilder2** interface.

The first parameter to the *RenderStream* method specifies a pin category; for a preview graph use `PIN_CATEGORY_PREVIEW`. The second parameter specifies a media type, as a major type GUID. For video, use `MEDIATYPE_Video`. The third parameter is a pointer to the capture filter's `IBaseFilter` interface. The next two parameters are not needed in this example. They are used to specify additional filters that might be needed to render the stream.

Setting the last parameter to `NULL` causes the Capture Graph Builder to select a default renderer for the stream, based on the media type. For video, the Capture Graph Builder always uses the Video Renderer filter as the default renderer.

```
hr = pBuilder->RenderStream(&PIN_CATEGORY_PREVIEW, &MEDIATYPE_Video,  
pFilter, NULL, NULL);
```

To start/stop the graph, use the **IMediaControl** interface:

```
CComQIPtr <IMediaControl, &IID_IMediaControl> control = pGraph;
```

```
hr = control->Run();    // start the graph  
////  
//// .....  
hr=control->Stop();    //stop the graph
```

4.1.5 Capturing to AVI

To capture the video stream to an AVI file use the AVI MUX filter as follows.

```
IBaseFilter *pMux;  
  
//specify the output media type, file name and get the resulting MUX  
hr = pBuild->SetOutputFileName(&MEDIASUBTYPE_Avi, L"C:\\test.avi",  
    &pMux, NULL);  
  
// Render the capture pin to the multiplexer  
hr = pBuild->RenderStream(&PIN_CATEGORY_CAPTURE, &MEDIATYPE_Video,  
    pCap, NULL, pMux);  
  
// Release the mux filter.  
pMux->Release();
```

You can encode the video stream by inserting an encoder filter between the capture filter and the AVI Mux filter. Use the System Device Enumerator or the Filter Mapper to select an encoder filter. Specify the encoder filter as the fourth parameter to *RenderStream*.

If you want to capture a video while you are previewing the video call the *RenderStream* method successively. The capture graph builder will automatically add a Smart Tee to split the capture stream from the preview stream.

4.1.6 Getting the Image Data

The Sample Grabber filter provides a way to retrieve samples as they pass through the filter graph. It is a transform filter with one input pin and one output pin. It passes all samples downstream unchanged, so you can insert it into a filter graph without altering the data stream. Your application can then retrieve individual samples from the filter by calling methods on the **ISampleGrabber** interface. If you want to retrieve samples without rendering the data, connect the Sample Grabber filter to the Null Renderer filter.

Before building the rest of the graph, you should set a media type for the Sample Grabber by calling the *SetMediaType* method. When the Sample Grabber connects, it will compare this media type against the media type offered by the other filter.

If you want to discard the samples after you are done with them connect the Sample Grabber to the Null Renderer Filter. The sample grabber operates either in buffering mode (makes a copy of each sample before delivering the sample downstream) or in callback mode (invokes an application-defined callback function on each sample).

```
// Create a Sample Grabber.
IBaseFilter *pGrabberF = NULL;
hr = CoCreateInstance(CLSID_SampleGrabber, NULL, CLSCTX_INPROC_SERVER,
    IID_IBaseFilter, (void**)&pGrabberF);
if (FAILED(hr))
{
    Error( "Failed to create a grabber" );
    return FALSE;
}
hr = pGraph->AddFilter(pGrabberF, L"Sample Grabber");
if (FAILED(hr))
{
    Error( "Failed to add the grabber to graph" );
    return FALSE;
}

// Query the Sample Grabber for the ISampleGrabber interface.
ISampleGrabber *pGrabber;
pGrabberF->QueryInterface(IID_ISampleGrabber, (void**)&pGrabber);

//specify RGB24 uncompressed video
AM_MEDIA_TYPE mt;
ZeroMemory(&mt, sizeof(AM_MEDIA_TYPE));
mt.majorType = MEDIATYPE_Video;
mt.subtype = MEDIASUBTYPE_RGB24;
hr = pGrabber->SetMediaType(&mt);

// Activate buffering mode
hr = pGrabber->SetBufferSamples(TRUE);

// Run the graph.
pControl->Run();
pEvent->WaitForCompletion(INFINITE, &evCode); // Wait till it's done.

// Find the required buffer size.
long cbBuffer = 0;
hr = pGrabber->GetCurrentBuffer(&cbBuffer, NULL);

// Allocate the array and call the method a second time to copy the buffer:
```

```
char *pBuffer = new char[cbBuffer];  
//get the image data  
hr = pGrabber->GetCurrentBuffer(&cbBuffer, (long*)pBuffer);
```

Note that images DirectShow are converted into standard video types (RGB8, RGB24, YUV411, YUV422, YUV444) . If your work with high-bit depth video formats and want to access original pixel values, use the [GetImageData](#) or [GetImagePointer](#) methods of [IActiveUSB](#) interface.

4.1.7 Displaying Property Pages

ActiveUSB Video Capture Filter provides a pass-through access to *ActiveUSB* [Property Pages](#). The filter and its output pin expose [ISpecifyPropertyPages](#) interface for retrieving a list of CLSIDs which may then be passed to the *OLECreatePropertyFrame* API function for display and control. The following code shows how to display the filter's property pages:

```
//get the property page interface
ISpecifyPropertyPages *pProp;
HRESULT hr = pFilter->QueryInterface(IID_ISpecifyPropertyPages, (void **)&pProp);

//get the filter's name and Unknown pointer
if (SUCCEEDED(hr))
{
    IUnknown *pFilterUnk;
    pFilter->QueryInterface(IID_IUnknown, (void **)&pFilterUnk);

//show the property pages
CAUUID caGUID;
pProp->GetPages(&caGUID);
pProp->Release();
OleCreatePropertyFrame(
    hwnd,                // parent window handle
    0, 0, NULL,         // caption for the dialog box
    1,                  // number of objects
    &pFilterUnk,         // array of object pointers
    caGUID.cElems,     // number of property pages
    caGUID.pElems,     // array of property pages CLSIDs
    0, 0, NULL
);

//release objects
pFilterUnk->Release();
CoTaskMemFree(caGUID.pElems);
}
```

For an example on how to display the pin's property pages refer to [ISpecifyPropertyPages](#).

The filter's property pages provides an access to those camera properties that can be modified while the graph is running, while the pin's property pages control the camera settings that cannot be accepted dynamically such as video mode and format change. Therefore you must stop the graph before displaying the pin's property pages and rebuild it afterwards.

4.2 Interfaces

The *ActiveUSB Video Capture Filter* provides a programmatic access to video acquisition and camera properties via the following COM-interfaces:

Video Capture Filter

IAMCameraControl	Provides programmatic control over shutter speed, iris, pan, tilt, zoom, focus, optical filter
IAMVideoProcAmp	Provides programmatic control over brightness, gain, gamma, sharpness, auto exposure, hue, saturation, white balance
IAMVideoControl	Provides programmatic control over image flipping and camera triggering
IActiveUSB	Provides a pass-through interface to <i>ActiveUSB</i> COM object
ISpecifyPropertyPages	Returns a list of <i>ActiveUSB</i> property pages supported by the Filter

Video Capture Pin

IAMStreamConfig	Provides an ability to retrieve and set video modes
ISpecifyPropertyPages	Provides a list of <i>ActiveUSB</i> property pages supported by the Pin
IAMVideoCompression	Provides the name and serial number of a camera
IAMDroppedFrames	Provides information about dropped and non-dropped frames

4.2.1 IAMCameraControl

Description

Provides programmatic control over the camera exposure.

Methods

HRESULT Get (long *Property*, long **pValue*, long **Flags*)

Retrieves the value of a specific camera control property.

HRESULT Set (long *Property*, long *IValue*, long *Flags*);

Sets the value of a specific camera control property.

HRESULT GetRange (long *Property*, long **pMin*, long **pMax*, long **pSteppingDelta*, long **pDefault*, long **pFlags*);

Retrieves values associated with the range of a specified camera property.

Parameters

Property

[in] Value that specifies the camera property. Use the following value:
CameraControl_Exposure - to control the camera's shutter.

pValue / IValue

[in/out] New value or pointer to the value of the specified property.

pValue

[in] New value of the specified property.

pMin

[out] Pointer to the minimum range for the specified property.

pMax

[out] Pointer to the maximum range for the specified property.

pSteppingDelta

[out] Pointer to the step size for the specified property.

pDefault

[out] Pointer to the default value of the specified property.

Flags / pFlags

[in/out] Value or pointer to the value indicating whether the camera property is automatic or manual:

CameraControl_Flags_Auto - sets the property's automatic flag.

CameraControl_Flags_Manual - sets the property's manual flag.

Return Values

S_OK

Success

E_FAIL

Failure.

E_INVALIDARG

Invalid argument.

Example

This C++ code request a pointer to **IAMCameraControl** interface from the video capture filter and sets the shutter value to 512:

```
IAMCameraControl *pCameraControl;  
HRESULT hr;  
hr = pFilter->QueryInterface(IID_IAMCameraControl, (void **)&pCameraControl);  
if(hr==S_OK)  
{  
    hr=pCameraControl->Set(CameraControl_Exposure,512,Camera Control_Flags_Manual);  
}
```

Remarks

IAMCameraControl is a standard DirectShow interface. Refer to *Microsoft DirectX SDK* documentation for more details.

4.2.2 IAMVideoProcAmp

Description

Provides programmatic control over the following camera properties: black level, gain, gamma, white balance.

Methods

HRESULT Get (long *Property*, long **pValue*, long **Flags*)

Retrieves the value of a specific camera control property.

HRESULT Set (long *Property*, long *IValue*, long *Flags*);

Sets the value of a specific camera control property.

HRESULT GetRange (long *Property*, long **pMin*, long **pMax*, long **pSteppingDelta*, long **pDefault*, long **pFlags*);

Retrieves values associated with the range of a specified camera property.

Parameters

Property

[in] Value that specifies the camera property. Use one of the following values:

VideoProcAmp_Brightness - to set and retrieve the camera's black level setting.

VideoProcAmp_Gain - to control the camera's gain.

VideoProcAmp_Gamma - to control the camera's gamma setting.

VideoProcAmp_WhiteBalance - to control the camera's white balance.

pValue / IValue

[in/out] New value or pointer to the value of the specified property.

pValue

[in] New value of the specified property.

pMin

[out] Pointer to the minimum range for the specified property.

pMax

[out] Pointer to the maximum range for the specified property.

pSteppingDelta

[out] Pointer to the step size for the specified property.

pDefault

[out] Pointer to the default value of the specified property.

Flags / pFlags

[in/out] Value or pointer to the value indicating whether the camera property is automatic or manual:

VideoProcAmp_Flags_Auto - sets the property's automatic flag.

VideoProcAmp_Flags_Manual - sets the property's manual flag.

Return Values

S_OK

Success

E_FAIL

Failure.

E_INVALIDARG

Invalid argument.

Example

This C++ code request a pointer to **IAMVideoProcAmp** interface from the video capture filter, initiates the automatic gain control and sets the black level value to 64:

```
IAMVideoProcAmp *pVideoProcAmp;
HRESULT hr;
hr = pFilter->QueryInterface(IID_IAMVideoProcAmp, (void **)&pVideoProcAmp);
if(hr==S_OK)
{
    hr=pVideoProcAmp->Set(VideoProcAmp_Gain,0,VideoProcAmp_Flags_Auto);
    hr=pVideoProcAmp->Set(VideoProcAmp_Brightness,64,VideoProcAmp_Flags_Manual);
}
```

Remarks

IAMVideoProcAmp is a standard DirectShow interface. Refer to *Microsoft DirectX SDK* documentation for more details.

4.2.3 IAMVideoControl

Description

Provides an ability to flip a picture horizontally and/or vertically, set up a trigger mode, issue the software trigger, and list the available frame rates.

Methods

HRESULT GetMode (IPin *pPin, long *pFlags)

Retrieves the video control properties.

HRESULT SetMode (IPin *pPin, long Flags)

Sets the video control properties.

HRESULT GetCaps (IPin *pPin, long *pFlags)

Requests availability of the video control properties.

HRESULT GetCurrentActualFrameRate (IPin *pPin, LONGLONG *pFrameRate)

Retrieves the actual frame rate, expressed as a frame duration in 100-nanosecond units.

HRESULT GetFrameRateList (IPin *pPin, long iIndex, SIZE Dimensions, long *ListSize, LONGLONG **FrameRates)

Retrieves a list of available frame rates for the specified video mode.

HRESULT GetMaxAvailableFrameRate(IPin *pPin, long iIndex, SIZE Dimensions, LONGLONG *pFrameRate)

Retrieves the maximum frame rate available for the specified video mode.

Parameters

pPin

[in] Pointer to the video capture pin.

pFlags / Flags

[in/out] New value or pointer to the value specifying a combination of the video control flags:

VideoControlFlag_FlipHorizontal - specifies that the picture is flipped horizontally.

VideoControlFlag_FlipVertical - specifies that the picture is flipped vertically.

VideoControlFlag_ExternalTriggerEnable - specifies that the camera is set to the trigger mode.

VideoControlFlag_Trigger - issues a software trigger signal.

pFrameRate

[in/out] Pointer to the frame rate. The frame rate is expressed as frame duration in 100-nanosecond units.

iIndex

[in] Index of the video mode to query for the frame rates.

Dimensions

[in] Frame image size (width and height) in pixels

ListSize

[out] Pointer to the number of elements in the list of frame rates.

FrameRates

[in] Address of a pointer to an array of frame rates in 100-nanosecond units.

Return Values

S_OK

Success
E_FAIL
Failure.
E_INVALIDARG
Invalid argument.

Example

This C++ code request a pointer to **IAMVideoControl** interface from the video capture filter, and sets the camera into the trigger mode with the horizontal image flipping:

```
IAMVideoProcAmp *pVideoControl;  
HRESULT hr;  
hr = pFilter->QueryInterface(IID_IAMVideoControl, (void **)&pVideoControl);  
if(hr==S_OK)  
{  
    hr=pVideoControl->Set(VideoProcAmp_Gain,0,VideoProcAmp_Flags_Auto);  
    hr=pVideoControl->Set(VideoProcAmp_Brightness,64,VideoProcAmp_Flags_Manual);  
}
```

Remarks

IAMVideoControl is a standard DirectShow interface. Refer to *Microsoft DirectX SDK* documentation for more details.

4.2.4 IActiveGige

Description

Provides a pass-through access to an *ActiveUSB* COM object associated with the video capture filter.

Methods

For a detailed description of methods refer to [Properties](#) and [Methods](#) in the *ActiveX Reference* chapter.

Example

This C++ code request a pointer to **IActiveUSB** interface from the video capture filter and sets up a bilinear Bayer interpolation:

```
IActiveUSB *pActiveUSB;  
HRESULT hr;  
hr = pFilter->QueryInterface(IID_IActiveUSB, (void **)&pCameraControl);  
if(hr==S_OK)  
{  
    hr=pActiveUSB->put_Bayer(2);  
}
```

Remarks

Use **IActiveUSB** interface to access the camera properties and methods not available via standard DirectShow interfaces. Since DirectShow does not support 16 bits per channel media types, you might want to use *IActiveUSB*'s image access methods for retrieving high-depth pixel data.

Note that for YUV video modes the [Display](#) property must be set to TRUE in order to use image access and image analysis functions such as [GetImageData](#), [GetComponentData](#), [GetImageLine](#), [GetImageWindow](#), [GetHistogram](#), [GetImageStat](#) etc. These functions require internal RGB conversion which by default is turned off by DirectShow filter to reduce the CPU load.

Refer to [ActiveX Reference](#) chapter for more details.

4.2.5 IAMStreamConfig

Description

Provides an ability to set stream formats and to find out what types of formats the Capture Pin can be connected to.

Methods

HRESULT GetFormat(AM_MEDIA_TYPE **pmt)

Retrieves the video stream's format.

HRESULT SetFormat(AM_MEDIA_TYPE *pmt)

Sets the video stream's format.

HRESULT GetNumberOfCapabilities(int *piCount, int *piSize)

Retrieves the number of stream capabilities (video modes) available for the current camera.

HRESULT GetStreamCaps(int iIndex, AM_MEDIA_TYPE **pmt, BYTE *pSCC)

Obtains video capabilities of a stream.

Parameters

pmt
[in/out] Pointer to a AM_MEDIA_TYPE structure.

piCount
[out] Pointer to the number of VIDEO_STREAM_CONFIG_CAPS (video modes) supported.

piSize
[out] Pointer to the size of the VIDEO_STREAM_CONFIG_CAPS structure.

iIndex
[in] Index to the desired media type and capability pair.

pSCC
[out] Pointer to a stream configuration structure. The structure is defined as follows:

```
typedef struct _VIDEO_STREAM_CONFIG_CAPS
{
    GUID guid; // set to MEDIATYPE_Video
    ULONG VideoStandard; // set to AnalogVideo_None
    SIZE InputSize; // maximum image size
    SIZE MinCroppingSize; // minimum ROI size (Format 7)
    SIZE MaxCroppingSize; // maximum ROI size (Format 7)
    int CropGranularityX; // horizontal size granularity (Format 7)
    int CropGranularityY; // vertical size granularity (Format 7)
    int CropAlignX; // horizontal offset granularity (Format 7)
    int CropAlignY; // vertical offset granularity (Format 7)
    SIZE MinOutputSize; // same as MinCroppingSize
    SIZE MaxOutputSize; // same as MaxCroppingSize
    int OutputGranularityX; // set to zero
    int OutputGranularityY; // set to zero
    int StretchTapsX; // set to zero
    int StretchTapsY; // set to zero
    int ShrinkTapsX; // set to zero
    int ShrinkTapsY; // set to zero
    LONGLONG MinFrameInterval; // minimum frame interval in 100 nanoseconds
    LONGLONG MaxFrameInterval; // maximum frame interval in 100 nanoseconds
    LONG MinBitsPerSecond; // minimum bandwidth
    LONG MaxBitsPerSecond; // maximum bandwidth
} VIDEO_STREAM_CONFIG_CAPS;
```

Return Values

S_OK
Success

E_FAIL
Failure.

E_INVALIDARG
Invalid argument.

Example

This C++ code request a pointer to **IAMStreamConfig** interface, collects available video modes from the video capture filter, finds a Format 7 mode with a horizontal resolution of 1024, sets ROI to 600x400 and switches the camera to this mode.

```

IAMStreamConfig* pSC;
if( pFilter->QueryInterface( IID_IAMStreamConfig, (void **)&pSC ) == S_OK )
{
    int iCount, iSize;
    VIDEO_STREAM_CONFIG_CAPS caps;
    pSC->GetNumberOfCapabilities(&iCount, &iSize);
    for(int i=0;i<iCount;i++)
    {
        AM_MEDIA_TYPE *pmt = NULL;
        if( pSC->GetStreamCaps(i, &pmt, (BYTE*)&caps) == S_OK )
        {
            if( caps.MaxOutputSize.cx == 1024 && caps.MaxOutputSize.cx!=caps.MinOutputSize)
            {
                ((VIDEOINFOHEADER *)(pmt->pbFormat))->rcSource.left=0;
                ((VIDEOINFOHEADER *)(pmt->pbFormat))->rcSource.top=0;
                ((VIDEOINFOHEADER *)(pmt->pbFormat))->rcSource.right=600;
                ((VIDEOINFOHEADER *)(pmt->pbFormat))->rcSource.bottom=400;
                pConfig->SetFormat(pmt);
            }
            DeleteMediaType (pmt);
            break;
        }
        DeleteMediaType (pmt);
    }
}

```

Remarks

Use the **GetNumberOfCapabilities** and **GetStreamCaps** methods to collect the information about available video modes. The information in VIDEO_STREAM_CONFIG_CAPS structures is particularly useful for specifying a Region of Interest (ROI) in partial scan modes (Format 7). In this case, *MinCroppingSize* and *MaxCroppingSize* fields define the range of possible ROI sizes, while *CropGranularity* and *CropAlign* define the ROI size and offset granularity. Modify the *rcSource* and *AvgTimePerFrame* fields of the VIDEOINFOHEADER block of a AM_MEDIA_TYPE structure in the **SetFormat** method to set up a specific ROI and frame rate. Refer to *Microsoft DirectX SDK* documentation for more details.

4.2.6 IAMVideoCompression

Description

Provides the name and serial number of the currently selected camera.

Methods

HRESULT GetInfo (*WCHAR *pszVersion*, *int *pcbVersion*, *LPWSTR pszDescription*, *int *pcbDescription*, *NULL, NULL, NULL, NULL*)

Parameters

pszVersion

[out] Pointer to a version string, such as "Version 2.1.0".

pcbVersion

[in, out] Size needed for a version string. Pointer to the number of bytes in the Unicode™ string, not the number of characters, so it must be twice the number of characters the string can hold. Call with this set to NULL to retrieve the current size.

pszDescription

[out] Pointer to a camera description.

pcbDescription

[in, out] Size needed for a description string. Pointer to the number of bytes in the Unicode string, not the number of characters, so it must be twice the number of characters the string can hold. Call with this set to NULL to retrieve the current size.

Return Values

S_OK

Success

E_FAIL

Failure.

E_INVALIDARG

Invalid argument.

Example

This C++ code request a pointer to **IAMVideoCompression** interface from the video capture filter and retrieves the information about the current camera:

```
IAMVideoCompression *pVideoCompression;
HRESULT hr;
int versize = VERSIZE;
int descsize = DESCSize;
WCHAR wachVer[VERSIZE]={0}, wachDesc[DESCSize]={0};
TCHAR camName[VERSIZE + DESCSize + 5]={0};
hr = pBuilder->FindInterface(&PIN_CATEGORY_CAPTURE,
    &MEDIATYPE_Video, pFilter,
    IID_IAMVideoCompression, (void **)&pVideoCompression);
if(hr==S_OK)
{
    hr = pVideoCompression->GetInfo(wachVer, &versize, wachDesc, &descsize, NULL, NULL, NULL, NULL);
    if(hr==S_OK)
        if(wcslen(wachDesc) && wcslen(wachVer))
            wprintf(camName, TEXT("%s - %s0"), W2T(wachDesc), W2T(wachVer));
}
```

Remarks

IAMVideoCompression is a standard DirectShow interface. Refer to *Microsoft DirectX SDK* documentation for

more details.

4.2.7 IAMDroppedFrames

Description

Provides information about frames that the filter dropped (that is, did not send), the frame rate achieved, and the data rate achieved.

Methods

HRESULT GetNumDropped(long *pIDropped)

Retrieves the total number of frames that the pin dropped since it last started streaming.

HRESULT GetNumNotDropped(long *pINotDropped)

Retrieves the total number of frames that the pin delivered downstream (did not drop).

HRESULT GetAverageFrameSize(long *pIAverageSize)

Retrieves the average size of frames that the pin dropped.

Parameters

pIDropped

[out] Pointer to the total number of dropped frames.

pINotDropped

[out] Pointer to the total number of frames that were not dropped

pIAverageSize

[out, retval] Pointer to the average size of frames sent out by the pin since the pin started streaming, in bytes.

Return Values

S_OK

Success

E_FAIL

Failure.

E_INVALIDARG

Invalid argument.

Example

This C++ code request a pointer to **IAMDroppedFrame** interface from the video capture pin and retrieves the number of dropped and non-dropped frames:

```
IAMDroppedFrames *pDroppedFrames;
HRESULT hr;
long nDropped, nNonDropped;
hr = pBuilder->FindInterface(&PIN_CATEGORY_CAPTURE,
    &MEDIATYPE_Video, pFilter,
    IID_IAMDroppedFrames, (void **)&pDroppedFrames);
if(hr)
{
    pDroppedFrames->GetNumDropped(&nDropped);
    pDroppedFrames->GetNumNotDropped(&nNonDropped);
}
```

Remarks

IAMDroppedFrames is a standard DirectShow interface. Refer to *Microsoft DirectX SDK* documentation for more details.

4.2.8 ISpecifyPropertyPages

Description

Provides a list of *ActiveUSB* property pages supported by the Video Capture Filter and Video Capture Pin. Property pages provide a built-in camera control GUI to DirectShow compatible applications.

Methods

`HRESULT GetPages(CAUUID *pPages)`

Fills an array of GUID values where each GUID specifies the CLSID of a corresponding *ActiveUSB* property page.

Parameters

pPages

[out] Pointer to a caller-allocated **CAUUID** structure that must be initialized and filled before returning. The *pElems* field in the **CAUUID** structure is allocated by the callee with **CoTaskMemAlloc** and freed by the caller with **CoTaskMemFree**.

Return Values

S_OK

Success

E_POINTER

The address in *pPages* is not valid.

Example

This C++ code displays built-in property pages for the Video Capture pin:

```
ISpecifyPropertyPages *pSpec;
CAUUID cauuid;
IAMStreamConfig *pSC;
hr =pBuilder->FindInterface(&PIN_CATEGORY_CAPTURE, &MEDIATYPE_Video,
    pFilter, IID_IAMStreamConfig, (void **)&pSC);
if (FAILED(hr))
{
    Error( "Capture pin not found" );
    return FALSE;
}
hr = pFilter->QueryInterface(IID_ISpecifyPropertyPages, (void **)&pSpec);
if(hr == S_OK)
{
    hr = pSpec->GetPages(&cauuid);
    hr = OleCreatePropertyFrame(ghwndApp, 30, 30, NULL, 1,
        (IUnknown **)&pSC, cauuid.cElems,
        (GUID *)cauuid.pElems, 0, 0, NULL);
    CoTaskMemFree(cauuid.pElems);
    pSpec->Release();
}
```

Remarks

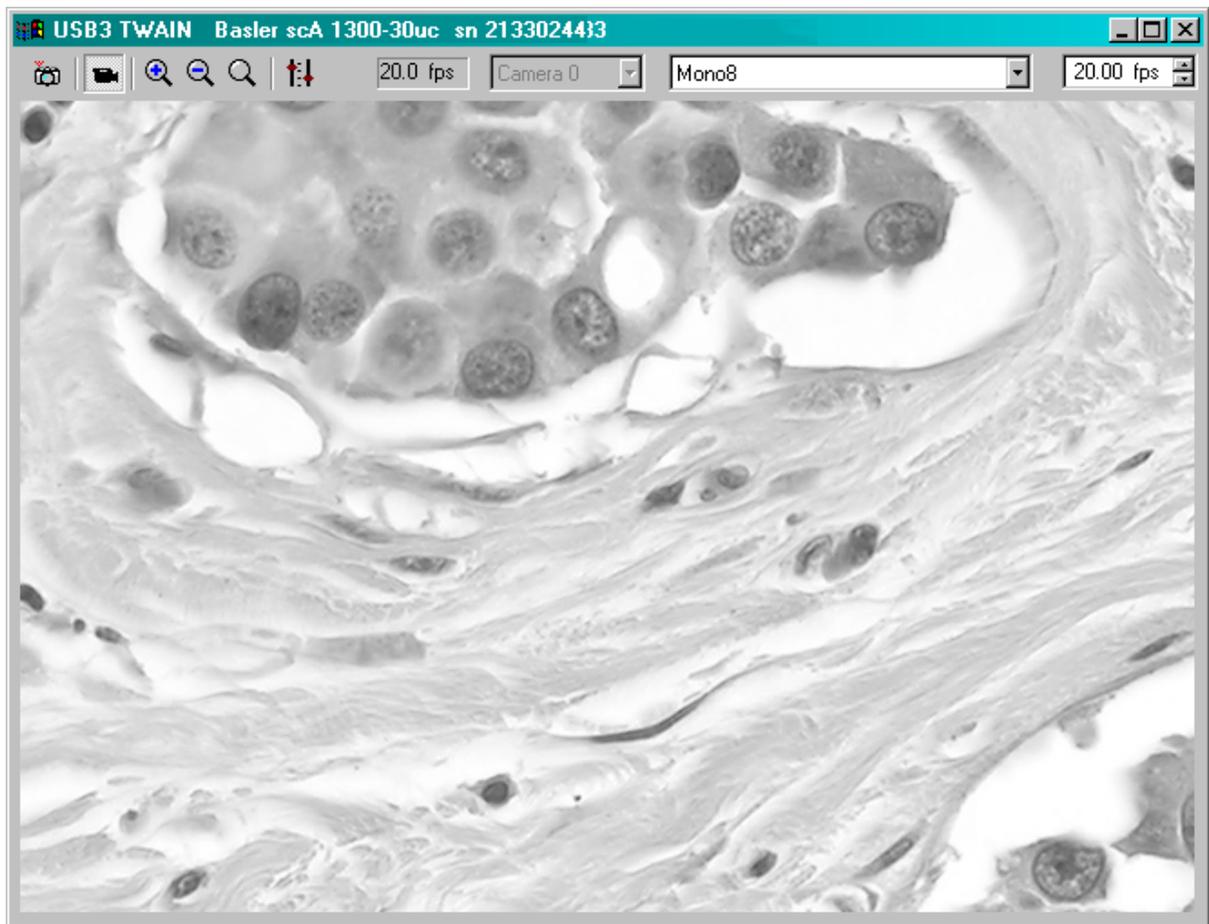
Video Capture Filter supports the [Exposure](#) and [Color](#) property pages. Video Capture Pin supports the [Source](#), [Format](#) and [Advanced](#) property pages. Refer to [ActiveX Reference](#) chapter for more information.

5 TWAIN

ActiveUSB TWAIN driver allows Windows users to view and capture images in TWAIN-compliant graphic editing and scientific imaging applications such as Photoshop, Image Pro and others. Depending on the selected video mode, the TWAIN driver can capture images in 8- or 16-bit monochrome and 24- or 48-bit RGB format.

The driver is automatically installed on your system during *ActiveUSB* [Installation](#). To use USB3 Vision™ cameras via a TWAIN compliant imaging application (e.g. Photoshop, Image Pro etc.), you must first set **USB3 Vision Compliant Camera** as the default TWAIN source.

- Open the TWAIN compliant application (e.g. Photoshop, Image Pro etc.)
- Specify the TWAIN source that you want to interface with. The procedure for this step will vary depending on the software you are using, but you should see a list of TWAIN devices.
- Chose **USB3 Vision Compliant Camera**.
- Open the TWAIN interface. This step also varies depending on the software. The USB3 Vision TWAIN window will appear as shown below.



The user interface of the TWAIN driver includes the following buttons and controls:

Capture

Captures the current video frame into the application.

Live

Switches the camera into the continuous acquisition mode and initiates live preview.

Freeze

Stops the acquisition from the camera and freezes the last video frame in the image window.

Zoom in

Click this button to increase the magnification of the image.

Zoom out

Click this button to decrease the magnification of the image.

Fit to screen

Click this button to change the magnification factor of the image so that it fits to the image window.

Settings

Click this button to display ActiveUSB [Property Pages](#).

Format

Use this option to select the desired pixel format from the list of available formats.

6 Samples

The ActiveUSB distribution package includes the following sample applications:

Programming language	Project name	Description
Visual Basic 6.0	UcamProfile	Live image preview, real-time line profile, drawing text, rectangles and ellipses
	MultiUcam	Dual camera viewer with custom exposure controls and pixel value display
	VBProcess	Live image processing with direct pixel manipulation in the frame buffer
	UcamStat	Real-time image statistics over an adjustable luminance range
	UcamEnhance	Real time frame averaging and integration, software gain control, hot pixel correction
	UcamAlpha	Animation effects in the alpha-plane over the live video, full screen mode
	UcamLUT	Selection among several user-defined look-up tables, color matrix correction
	UcamBarcode	1D and 2D barcode decoding
	UcamLensCorrect	Real-time lens distortion correction
	UcamByRef	Live image in PictureBox object, pixel values at cursor coordinates, fps display, using ActiveUSB by reference
	UcamAVI (DVR-version)	Video-capture and playback to/from AVI files with the sound recording
	UcamSequence (DVR-version)	Video capture and playback to/from memory sequence
	UcamReplay (DVR-version)	Live video with a delayed replay using memory loop recording
	UcamStreamer (DVR-version)	Live video with simultaneous web streaming to a specified ip address
Visual Basic 6.0, DirectShow	VBCap	Live image preview, built-in property pages. Based on ActiveUSB Video Capture Filter and DirectShow programming interface.

Visual C++, MFC	ActiveDemo PlugUnplug	Live image preview, real-time line profile, histogram and image statistics over ROI, saving image into file, continuous capture into multiple frames. Multiple camera viewer with plug/unplug event processing. Ability to disconnect and connect up to 4 cameras with streaming video.
Visual C++, Win32 API	UcamConsole UcamWin	Console application showing how to use ActiveUSB API to retrieve the list of connected cameras and video formats, capture a series of frames and modify camera properties Live image preview using DIB rendering, built-in and custom camera controls, saving image into file. Written in C with Win32 API (no MFC used)
Visual C++, DirectShow	UcamCap	Live image preview, built in and custom camera controls. Based on ActiveUSB Video Capture Filter and Microsoft DirectX SDK.
VB.NET	UcamOverlay UcamCapture	Flipped video preview, pseudo-coloring, pixel window extraction, overlay animation Time-lapse video capture to AVI files or series of images, overlaid frame counter
Visual C#	UcamSharp FilterSharp	Image and pixel viewer with custom camera controls: selection among multiple cameras and pixel formats, pseudo-color display, real-time exposure and gain control, mouse event handling Direct access to the frame buffer using pointers, real-time Emboss filter, non-destructive color overlay, interactive drawing
Delphi	UcamHist	Live video with real time 3-color histogram overlay

MATLAB	SingleCamera	Single camera application with custom exposure control.
	MultiCamera	Two-camera application with built-in property controls.
	MatlabViewer	Live image preview with real-time pixel values, line-profile and 3D-plot, mouse event handling
Python	ActiveUsb.py	Obtaining the camera and format list, modifying the gain value, retrieving pixel values
Adobe Flash	UcamFlash.swf	A mirrored live image preview, color picker, transparent bitmaps over the video
HTML	TryActiveUsb.html	Web page with an integrated viewer

Note that VB.NET and Visual C# samples utilize .NET Framework. Make sure it is installed on your system before using these samples. In addition UcamOverlay and UcamSharp samples assume the presence of MSCOMM32.ocx on the system.

All the samples include complete source code and executables which can be run with any USB3 Vision™ camera out of the box. Simply connect one or more cameras to your system, start an application of your choice and see how *ActiveUSB* performs in real world!

7 Camera list

ActiveUSB is designed to work with any USB3 Vision™ compatible camera. The following cameras have been tested with *ActiveUSB*:

Make	Models
Allied Vision Technologies	Mako USB3 series
Basler Vision Technologies	Ace USB3 series
Baumer Optronics	MX and PX USB 3.0 series
IDS	uEye U3V series
ISG	LightWise Allegro USB 3.0 series
JAI	GO and SP USB3 series
Lumenera	Lt USB 3.0 series
Matrix Vision	mvBlueFox series
Pixelink	Prometheus, Hyperion and Titan series
Pleora Technologies	iPORT CL-U3, SB-U3, NTx-U3 modules
Point Grey	Backfly, Chameleon3, Flea3 and Grasshopper3 series
Sensor to Image	FPGA U3V engines
Sentech	STC U3V series
Sumix	Hummingbird SMX-16xx series
Toshiba Teli	BU and DU series
Ximea	xiQ series

8 Troubleshooting

Below is the list of the most frequently encountered issues and remedies for their resolutions:

Problem description	Cause	Resolution
<p>The camera is not recognized by <i>ActiveUSB</i>. "No USB3 Vision camera found" message appears in the control window.</p>	<p>The camera is not receiving power.</p> <p><i>USB3 Vision Camera Driver</i> is not installed for the camera.</p> <p>The camera is not USB3 Vision™ compliant.</p>	<p>This is a typical issue for USB3 hubs that do not provide power to external devices. Use powered USB3 hubs or connect your camera directly to a USB3 port on your computer.</p> <p><i>ActiveUSB</i> will not work with a system driver provided by your camera manufacturer. Refer to Driver Setup for the USB3 Vision driver installation procedure.</p> <p><i>ActiveUSB</i> only works with USB3 Vision™ compliant cameras. Non-compliant USB3 cameras (without USB3 Vision™ logo) will not be recognized by <i>ActiveUSB</i>. Use the manufacturer's provided software.</p>
<p>Live video would not start or occasionally freezes.</p>	<p>The camera and USB3 adapter have a compatibility problem.</p> <p>The computer/USB3 adapter/cabling has an intermittent hardware issue.</p>	<p>Some USB3 Vision boards/chipsets are known not to perform well with certain models of USB3 cameras. Replace your board with a board recommended by your camera manufacturer.</p> <p>Replace the cable or try to run the camera/software on a different system.</p>
<p>The video is corrupted (frames are broken into parts, or synchronization is lost). Some video formats and frame rates do not work.</p>	<p>The camera and USB3 board have a compatibility problem.</p> <p>The processor has an extensive latency in the C3 power state transition.</p>	<p>See the solution above.</p> <p>This problem is common for notebook computers. To correct the latency setting, run C3State.reg file located in the Driver folder. Restart the system for the changes to take effect.</p>
<p>When I select certain Formats, I am getting a split or scrambled video</p>	<p>Some cameras offer manufacturer-defined formats not compatible with USB3 Vision specifications</p>	<p>Use one of the standard formats listed in the Format chapter.</p>

Problem description	Cause	Resolution
When switching to certain video modes, I get blank screen.	Some cameras require turning the acquisition off while switching between video modes.	Turn the acquisition off by setting the Acquire property to off, then select the desired video mode and turn the acquisition on.
When I try to compile sample projects from their default directories in C:\Program Files, I get an error message "Could not create an output directory"	You do not have the right permissions due to User Account Control (UAC).	Copy the Samples folder from C:\Program Files\ActiveUSB\ to one of your user directories (e.g. Desktop) and open sample projects from there. Alternatively, ask your system administrator to provide you with writing permissions for the ActiveUSB folder.
VB.NET and C# sample applications do not work.	.NET framework is not installed on the system	Install the latest .NET framework from Microsoft: http://msdn.microsoft.com/netframework/default.aspx
When I run a live video application from within the Visual Studio, the application shows "Camera in use by another process" message.	Visual Studio does not close the design view while starting the application.	The design window of the IDE maintains control over the camera blocking your application from initiating the acquisition. Make sure to close the design view before running your application.
I am trying to do real-time image processing in .NET, and I experience a significant drop in the frame rate.	VB.NET and C# have performance issues when working with large arrays. FrameAcquired event has an overhead that might affect the performance	For performance boost, use unsafe code and pointers to directly access <i>ActiveUSB</i> image buffer. A pointer to the image buffer is provided by GetImagePointer . For VB.NET, C# and C++ applications use FrameAcquiredX event.
AVI files are not recorded in real-time, many frames are being dropped.	Your system does not provide enough throughput or CPU power to keep up with the camera frame rate	Switch to a lower resolution mode or reduce the frame rate. If you are using a Bayer camera, consider recording the monochrome video instead of color one. Alternatively, upgrade your system to a faster hard drive (such as an SSD) and/or faster CPU.

Problem description	Cause	Resolution
<p><i>ActiveUSB</i> installation fails with the following error: "ActiveUSB.dll failed to register, HRESULT - 2147024770"</p>	<p>Runtimes components of Visual C++ 8.0 are not installed on your system</p>	<p>Install Microsoft's VC80 Redistributable Package, then run <i>ActiveUSB</i> installation again.</p> <p>For the 64-bit OS you may also need to install VC80 Redistributable Package 64-bit</p>
<p><i>ActiveUSB</i> installation fails with the following error: "ActiveUSB.dll failed to register, HRESULT - 2147220473"</p>	<p>This error is usually caused by a corrupted registration of atl.dll system file.</p>	<ol style="list-style-type: none"> 1. Locate atl.dll, generally found in "C:\WINNT\system32" or "C:\WINDOWS\system32". 2. Open the command-line prompt <ul style="list-style-type: none"> • o Start Menu/Run • o type "cmd" • o press "ENTER" 3. Using the atl.dll filename and path, call regsvr32, i.e. at the command-line prompt, type: <ul style="list-style-type: none"> • o regsvr32 "C:\WINNT\system32\atl.dll" • o press "ENTER" 4. You should see a regsvr32 window, confirming success: Close it. 5. Repeat <i>ActiveUSB</i> installation.

Index

- 1 -

1394 driver 15

- 3 -

3D look 93

- A -

Accumulate 116, 117
Acquire 54
Acquisition 54, 56, 59
AcquisitionFrameCount 56
AcquisitionFrameRate 58
AcquisitionMode 59
Affinity 61
Alpha 62
Analog property page 446
Anti-tearing 64
API 44
Average 116, 117
AVI 182, 185, 207, 208, 209, 229, 230, 231, 318, 319, 320, 321, 328, 336, 351, 352, 354, 356, 381, 382, 384, 386, 389, 390, 391, 396, 398, 402, 404, 405, 406, 416, 426, 427, 437

- B -

Background color 66
BalanceRatioAbs 67
BalanceRatioSelector 69
BalanceWhiteAuto 71
Bayer filter 73
BinningtX 75
BinningtY 76
BkgCorrect 77
BkgName 79
Black level 80, 82, 84, 216, 217
BlackLevel 80
BlackLevelAuto 82
BlackLevelSelector 84
bmp 324, 345

Building the Graph 461

- C -

C# 40
C++ 32
Camera 86
Camera list 221
CameraPlugged event 414
CameraUnplugged event 415
CaptureComplete 349
CaptureCompleted event 416
Capturing to AVI 463
Category list 224
CloseVideo 182
ColorCorrect 88
Compatibility 486
Configurator 22
CreateSequence 183
CreateVideo 185

- D -

DecimationX 89
DecimationY 90
Digital I/O 120, 122, 124, 126, 128, 163, 165
DirectShow 455, 458, 459, 461, 462, 463, 464, 466, 468, 470, 472, 474, 475, 477, 479, 480
DirectShow Interfaces 467
DirectShow Quick Reference Guide 456
Display 91
Displaying Property Pages 466
Displaying the Preview 462
Distributing 29
Draw 187
DrawAlphaClear 188
DrawAlphaEllipse 189
DrawAlphaLine 191
DrawAlphaPixel 193
DrawAlphaRectangle 194
DrawAlphaText 196
DrawEllipse 198
DrawLine 200
DrawPixel 201
DrawRectangle 202
DrawText 204

- E -

Edge 93
 EventDataMessage event 419
 EventMessage event 417
 Events 412
 Exposure 94, 96, 98, 240, 241
 ExposureAuto 94
 ExposureMode 96
 ExposureTime 98

- F -

Feature access 242, 244, 247, 253, 255, 257, 260, 271, 362, 364, 366, 368
 Feature information 238, 250, 262, 263
 Feature list 251
 File Access 266, 267, 269, 340
 FileAccess 409
 FilterConfig 458
 Flip 100
 Font 102
 Format list 273
 FormatChanged event 421
 FPS 58, 205, 206
 Frame rate 275, 276
 FrameAcquired event 422
 FrameAcquiredX event 423
 FrameDropped event 425
 FrameLoaded event 426
 FrameReady event 428
 FrameRecorded event 427
 Fromat 103

- G -

Gain 106, 108, 110, 277, 278
 GainAuto 108
 GainSelector 110
 Gamma 112
 GenlCam property page 450
 GetAcquisitionFrameRateMax 205
 GetAcquisitionFrameRateMin 206
 GetAudioLevel 207
 GetAudioList 208
 GetAudioSource 209

GetBalanceRatioMax 210
 GetBalanceRatioMin 211
 GetBarcode 212
 GetBitsPerChannel 214
 GetBlackLevelMax 216
 GetBlackLevelMin 217
 GetBlockId 218
 GetBytesPerPixel 219
 GetCameraIP 220
 GetCameraList 221
 GetCameraMAC 223
 GetChunkPointer 225
 GetChunkSize 227
 GetCodec 229
 GetCodecList 230
 GetCodecProperties 231
 GetComponentData 232
 GetComponentLine 234
 GetDIB 236
 GetEnumList 238
 GetExposureTimeMax 240
 GetExposureTimeMin 241
 GetFeature 242
 GetFeature64 244
 GetFeatureAccess 246
 GetFeatureArray 247
 GetFeatureDependents 249
 GetFeatureDescription 250
 GetFeatureIncrement 253
 GetFeatureList 251
 GetFeatureMax 257
 GetFeatureMin 255
 GetFeatureRepresentation 259
 GetFeatureString 260
 GetFeatureTip 262
 GetFeatureType 263
 GetFeatureVisibility 265
 GetFileAccessMode 266
 GetFileList 267
 GetFileSize 269
 GetFileTransferProgress 271
 GetFormatList 273
 GetFPS 276
 GetFPSAcquired 275
 GetGainMax 277
 GetGainMin 278
 GetHistogram 279
 GetImageData 281, 360

GetImageLine 283
GetImagePointer 285
GetImageStat 287
GetImageWindow 289
GetLevels 291
GetLUT 292
GetOptimalPacketSize 293
GetPicture 295
GetPixel 297
GetRawData 299
GetRGBPixel 301
GetROI 303
GetSequenceFrameCount 304
GetSequencePicture 305
GetSequencePixel 306
GetSequencePointer 308
GetSequenceRawData 310
GetSequenceTimestamp 312
GetSequenceWindow 313
GetTimestamp 315
Getting the Image Data 464
GetTriggerDelayMax 316
GetTriggerDelayMin 317
GetVideoFPS 318
GetVideoFrameCount 319
GetVideoPosition 320
GetVideoVolume 321
Grab 322
Guide 29

- H -

Height 147
Histogram 279
Horizontal binning 75
Horizontal decimation 89
Horizontal offset 148
HotPixelCorrect 114
HotPixelLevel 115

- I -

IActiveGige 474
IAmCameraControl 468
IAMDroppedFrames 479
IAMStreamConfig 475
IAMVideoCompression 477

IAMVideoControl 472
IAMVideoProcAmp 470
Input/Output property page 448
Installation 12
Integrate 116
IntegrateWnd 117
Introduction 7
IsFeatureAvailable 323
ISpecifyPropertyPages 480

- J -

jpeg 324, 345

- L -

LensCorrect 118
License 7, 9
LineFormat 120
LineInverter 122
LineMode 124
LineSelector 126
LineSource 128
LoadImage 324
LoadSequence 327
LoadSettings 325
Lookup table 130, 292, 370, 377
LUTMode 130

- M -

Magnification 131
Methods 170
MonitorSync 133
MouseDownClick event 430
MouseDown event 431
MouseDownRight event 432
MouseMove event 434
MouseUp event 435
MouseUpRight event 436
Multiple cameras 47

- O -

OffsetX 148
OffsetY 149
OpenVideo 328

Overlay 135
 OverlayClear 330
 OverlayColor 136
 OverlayEllipse 331
 OverlayFont 137
 OverlayLine 332
 OverlayPixel 333
 OverlayRectangle 334
 OverlayText 335
 Overview 7

- P -

Palette 138
 PalyVideo 336
 Pixel depth 214, 219
 PlayCompleted event 437
 Properties 48
 Property pages 393, 441
 Pseudo colors 138

- R -

RawFrameAcquired event 438
 Read block 338
 Read register 342
 ReadFile 340
 Reference 47
 Registration 21
 Requirements 11
 Retrieving the Filter 459
 ROI 303, 379
 Rotate 140

- S -

Samples 483
 SaveBkg 343
 SaveImage 345
 SaveSequence 349
 SaveSettings 347
 Scroll event 439
 ScrollBars 142
 ScrollX 144
 ScrollY 145

Sequence 183, 304, 305, 306, 308, 310, 312, 313,
 318, 320, 327, 328, 336, 349, 381, 382, 384, 399,
 401, 405, 406, 416, 426, 427, 437

SetAudioLevel 351
 SetAudioSource 352
 SetCodec 354
 SetCodecProperties 356
 SetColorMatrix 358
 SetFeature 362
 SetFeature64 364
 SetFeatureArray 366
 SetFeatureString 368
 SetGains 370
 SetLensDistortion 372
 SetLevels 374
 SetLUT 377
 SetROI 379
 SetVideoFPS 381
 SetVideoPosition 382
 SetVideoSync 384
 SetVideoVolume 386
 SetWebStreamer 387
 ShowAudioDlg 389
 ShowCodecDlg 390
 ShowCompressionDlg 391
 ShowProperties 393
 SizeX 146
 SizeY 147
 SoftTrigger 395
 Software trigger 395
 Source property page 442, 452
 StartCapture 396
 StartSequenceCapture 399
 StartVideoCapture 402
 StopCapture 398
 StopSequenceCapture 401
 StopVideo 405
 StopVideoCapture 404

- T -

TestImageSelector 151
 tiff 324, 345
 Timeout 150, 440
 Trigger 153
 TriggerActivation 155
 TriggerDelay 157
 Triggering 155, 157, 159, 161, 316, 317

TriggerSelector 159
TriggerSource 161
TriggerVideo 406
Troubleshooting 487
TWAIN 481

- U -

UcamViewer 25
UserOutputSelector 163
UserOutputValue 165
UserSetSelector 166

- V -

VB 30
VB.NET 36
Vertical binning 76
Vertical decimation 90
Vertical offset 149
Video Format 444
Visual Basic 30
Visual C# 40
Visual C++ 32

- W -

Web streaming 168
White balance 67, 69, 71, 210, 211
Width 146
Window/Level 291, 374
Write block 407
Write register 411
WriteFile 409