ActiveBF

User Guide

Version 4.0

Fifth Edition

January 2012

A&B Software LLC
New London, CT 06320
USA

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

Table of Contents

1 Introduction

ActiveBF is an SDK and ActiveX control built around BitFlow driver and designed for rapid application development tools, such as Visual Basic, VB.NET, Visual C++, C#, Java, Delphi, Python, etc. Provided is BFViewer application allowing customers to operate multiple BitFlow boards and save images in a number of formats. With *ActiveBF* control your application immediately supports all of BitFlow's framegrabbers: *Road Runner*, *Raven*, *R3*, *R64, Karbon, Neon, Alta*.

In general, with *ActiveBF* you can :

- Acquire and display live video from one or several BitFlow boards.
- Select among multiple camera sources.
- Set a desired video format and triggering mode.
- Use hardware and software triggers.
- Choose between synchronous and asynchronous acquisition modes.
- Adjust hardware brightness, contrast, and gamma.
- Flip and rotate the live image.
- Choose among several palettes for pseudo-color display.
- Grab 8-, 10-, 12-, 14-, 16-bit monochrome images or 24-, 30-, 32-, 36-, 48-bit color images.
- Perform automatic color interpolation of a monochrome video generated by Bayer cameras.
- Get an instant access to pixel values and pixel arrays.
- Retrieve individual color planes from RGB images.
- 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.
- Apply custom LUTs (lookup tables) to incoming video frames.
- Perform manual and automatic Window/Level processing (brightness, contrast, white balance).
- Save images in BMP, TIF and JPEG formats with adjustable compression.
- Perform time-lapse capture to an AVI file or series of sequentially-named images.
- 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.
- Communicate with cameras via Camera Link serial port.
- Control digital input/output on the board.
- Interface to DirectShow-based applications via the included Video Capture Source filter.

With the extended *ActiveBF 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 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.

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

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

1.1 License Agreement

This legal document is an agreement between you, the Licensee, and A&B Software LLC. By installing *ActiveBF 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 *ActiveBF SDK* related files.

1. Subject of agreement
The subject of this agreement is the software *ActiveBF SDK*, the operating manuals, and all other accompanying material. It will be referred to henceforth as *ActiveBF 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 *ActiveBF 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 *ActiveBF 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 *ActiveBF SDK* based programs. Likewise, the end user must not be given the *ActiveBF SDK* serial number.

4. Royalties
The *ActiveBF 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 *ActiveBF SDK*.

5. Redistributable files
The redistributable components of *ActiveBF SDK* are those files specifically designated as being distributable in the distributing your application section of ActiveBF User's Guide.

6. Trial version
A&B Software LLC grants the Licensee a non-exclusive license to test *ActiveBF 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
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 *ActiveBF 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, alter, translate, decompile, or disassemble *ActiveBF SDK*

- copy *ActiveBF SDK's* accompanying written documentation
- lend, hire out or lease *ActiveBF SDK*.

A permanent transference of *ActiveBF SDK* is only permitted when the Licensee retains no copies and the recipient declares his acceptance of the conditions of this agreement.

9. Exclusion of warranties
A&B Software LLC offers and the Licensee accepts the product 'as is'. A&B Software LLC does not warrant *ActiveBF 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 *ActiveBF 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 *ActiveBF SDK* are to be destroyed.

1.2 System Requirements

Hardware requirements:
- Pentium 1.5 GHz or better CPU recommended.
- 1G or more RAM recommended.
- A graphic card supporting 65535 colors or higher
- One or more BitFlow framegrabbers installed on the system.

Software requirements:
- Windows 2000, XP, Vista, Windows 7, XP 64-bit, Vista 64-bit, Windows 7 64-bit
- BitFlow SDK 5.30 or higher (free driver-only version).

1.3 Installation

Before installing *ActiveBF*, make sure that you have installed the latest version of *BitFlow SDK* (free driver-only version). If you are installing *ActiveBF* on a Windows Vista/7, you are recommended having administrator-level access.

To install *ActiveBF*, perform the following steps:

1. Save and exit out of all currently running applications.

2. Insert *ActiveBF* CD into the CD-ROM drive. The setup program should start automatically. If not, navigate to the following path (assuming D: is the drive letter of your CD-ROM drive): D:\setup.exe, and then double click the name of the file. The InstallShield Wizard 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 *ActiveBF.* The **Welcome** dialog box will appear. After reading the preliminary information, click **Next**. Note that if you select the complete type of setup, *ActiveBF's* location will be C:\Program Files\ActiveBF

4. The **License Agreement** dialog box will appear. To accept the license and continue, click **I accept the terms in the license agreement**, and then click **Next**. Note that the Next button is not available until you click "I accept". If you do not accept the license, click **I do not accept the terms in the license agreement**, and setup will terminate.

5. The **Setup Type** dialog box will appear. Select one of the following types of setup:

   · **Typical**: Installs all *ActiveBF* components, including the main files, sample applications, Help and documentation.

   · **Custom**: Allows you to select which component you want installed and to change *ActiveBF* default installation location.

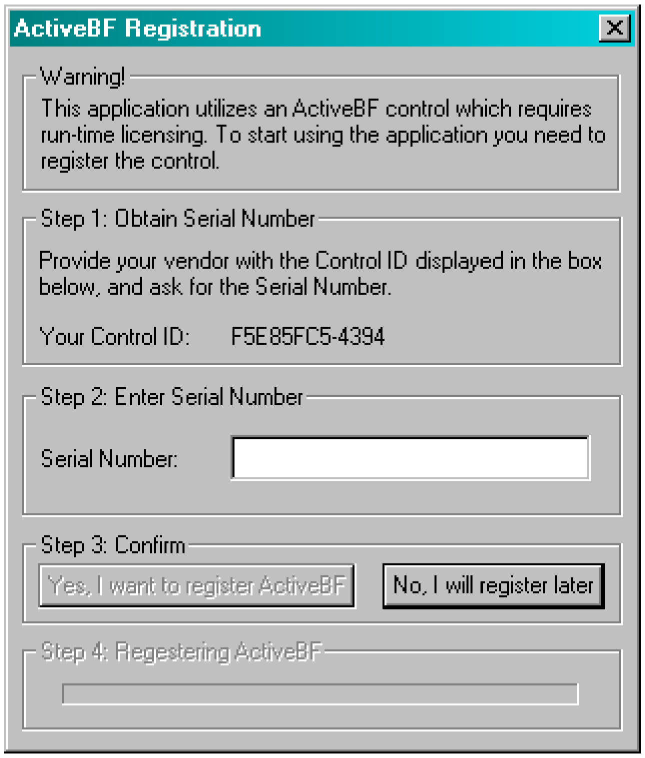
6. When **Ready to Install** dialog box appears, click Install. *ActiveBF* will begin installing on your system.

7. Once installation is finished, the **Completed** dialog box will appear. Click **Finish** to exit the **InstallShield Wizard**. 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.

The *ActiveBF* setup includes the demonstration license that allows you to use the control in both design and run-time mode for a period of 14 days.

If you wish to continue using *ActiveBF*, you must acquire a design-time license from your distributor. The design-time license is provided in form of a license file *activebf.lic* that must be saved in the ActiveBF folder (typically, C:\Program Files\ActiveBF) replacing the existing file. In addition, if you need to execute ActiveBF based applications outside of your 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 *ActiveBF* control, the registration dialog box will appear.



ActiveBF Registration

Warning!
This application utilizes an ActiveBF control which requires run-time licensing. To start using the application you need to register the control.

Step 1: Obtain Serial Number
Provide your vendor with the Control ID displayed in the box below, and ask for the Serial Number.

Your Control ID: F5E85FC5-4394

Step 2: Enter Serial Number
Serial Number:

Step 3: Confirm

Step 4: Regestering ActiveBF

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

-

1.4 Distributing your application

If you create an application using *ActiveBF* control, your distribution package should include the file *activebf.dll*. There is no need to provide a user of your application with the license file *activebf.lic.* The control must be registered on the end-user's system before the application can be used. Therefore, you may want your setup program to register the control when the application is installed. You can do it by passing the complete path to *activebf.dll* as an argument to *regsvr32.exe*, or you can write your own setup program to register the control directly instead. You might also want to include *BitFlow SDK* (free driver-only version) in your distribution package and make sure the user installs it before starting using your application.

In addition, you must ensure that the following files exit on the end-user's system:

mfc42.dll, v.6.0	MFCDLL shared 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.

When an ActiveBF based application is executed for the first time on an end-user's system, *ActiveBF* control 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 *ActiveBF* control using various development platforms.

Visual Basic
Visual C++
Visual C#

2.1 Visual Basic

This chapter shows you how to get started with *ActiveBF* 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

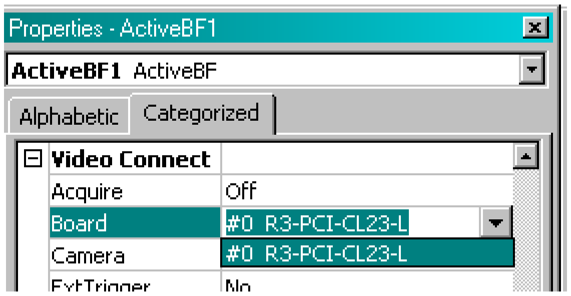
Assuming that you have already run the *ActiveBF* installation program and started Visual Basic, the next step is to create a project. Begin by selecting the *New Project* command from the file menu, and select *Standard EXE* as your project type. Then use the *Project / Components...* command and select *ActiveBF 1.5 type library* from the list. You will see *ActiveBF* icon appear at the bottom of the toolbox. You will see *ActiveBF* icon appear at the bottom of the toolbox:

Creating the Control

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

Selecting the Board

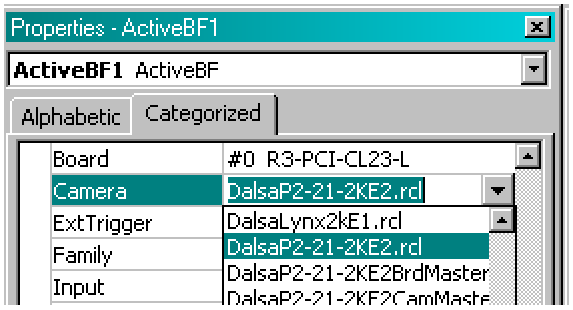
In the properties window, click the *Board* property. The list box will display BitFlow boards installed on your system. Select the one you intend to use:



Video Connect	
Acquire	Off
Board	#0 R3-PCI-CL23-L
Camera	#0 R3-PCI-CL23-L
ExtTrigger	No

Selecting the Camera

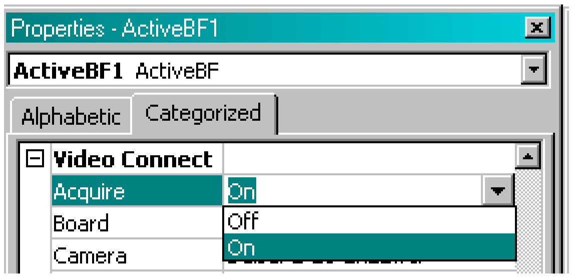
In the properties window, click the *Camera* property. The list box will display all the camera configuration files available for the chosen board. Select the one you intend to use:



Board	#0 R3-PCI-CL23-L
Camera	DalsaP2-21-2KE2.rcl
ExtTrigger	DalsaLynx2kE1.rcl
Family	DalsaP2-21-2KE2.rcl
Input	DalsaP2-21-2KE2BrdMaster
	DalsaP2-21-2KE2CamMaste

Activating continuous acquisition

In the properties window, click the *Acquire* property and set it to "On". If the selected camera is connected to the board, the live video will appear on the form in the *ActiveBF* 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 *ActiveBF* 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 ActiveBF1_FrameAcquired(ByVal Lines As Integer)
   'the following line needs to be added to the code
   Label1.Caption = ActiveBF1.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 *ActiveBF* 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 *MyActiveBF* and click *OK*. When *MFC AppWizard* appears, select *Dialog based* radio button and click *Finish*. The project will be created, and the program dialog *MyActiveBF* will be displayed for editing.

VC++ 2005, 2008, 2010

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* filed below type the name of your application, for instance *MyActiveBF* 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 *MyActiveBF* 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 *ActiveBF class* and press *OK*.
A white rectangle with the text "ActiveBF Control" will appear on the dialog.

Generating the class for the Control

VC++ 6.0

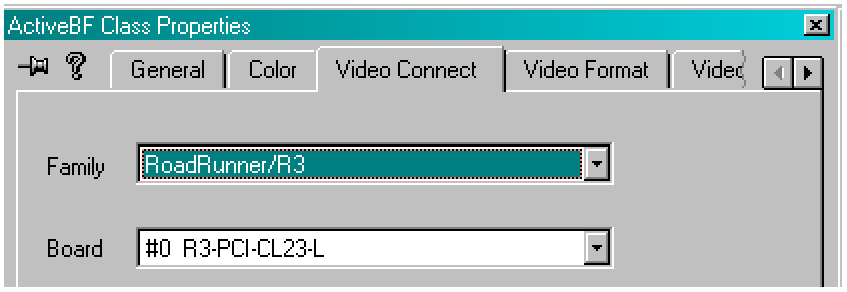
Right click in the dialog and select *Class Wizard* from the shortcut menu. When *MFC Class Wizard* appears, click the *Member Variables* tab. In the *Control ID* window click IDC_ACTIVEBF1 and then click *Add Variable.* Confirm generating the new CActiveBF class. When the *Add Member Variable* dialog appears, enter the name for the ActiveBF object, for instance *m_ActiveBF*. Click *OK* to close the Wizard.

VC++ 2005, 2008, 2010

Right click on the *ActiveBF* 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 ActiveBF object, for instance *m_ActiveBF*, and click finish. The Wizard will generate a wrapper class for ActiveBF control and add a corresponding member variable to the main dialog class.

Selecting the Board

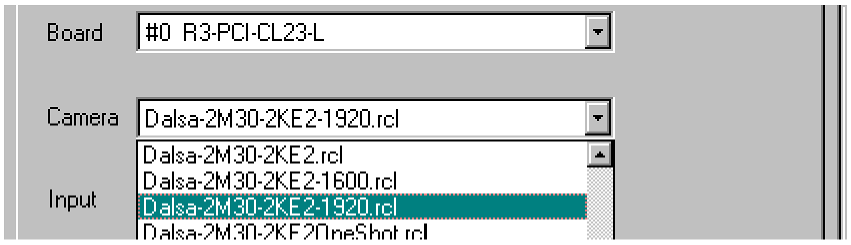
Right click on the *ActiveBF* control in the program dialog and select *Properties* from the shortcut menu. (In .NET select *Property pages* from the *View* menu). This will display the *ActiveBF Class Properties* tab dialog. Click the *Video Connect* tab. The *Board* list box will contain the names of the boards installed on your system. Select the one you intend to use.



Family: RoadRunner/R3
Board: #0 R3-PCI-CL23-L

Selecting the Camera

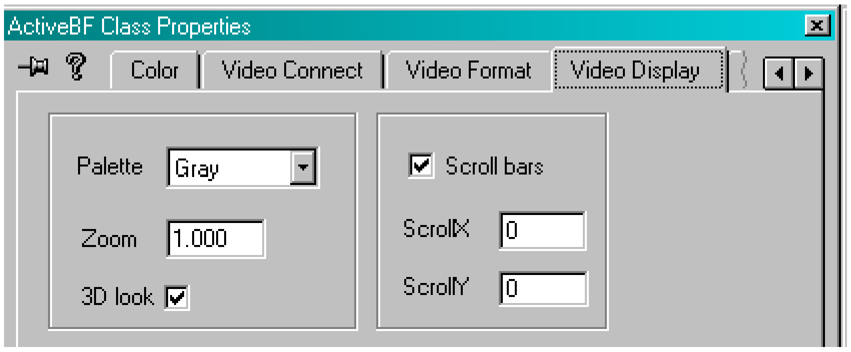
Click the arrow next to the *Camera* box. The list box will display all the camera configuration files available for the chosen board. Select the one you intend to use:


Board: #0 R3-PCI-CL23-L
Camera: Dalsa-2M30-2KE2-1920.rcl

```
Dalsa-2M30-2KE2.rcl
Dalsa-2M30-2KE2-1600.rcl
Dalsa-2M30-2KE2-1920.rcl
Dalsa-2M30-2KE2OneShot.rcl
```

Modifying the Control's appearance

Switch to the *Video Display* tab. Select *3D look* and *Scroll bars* check boxes:



Palette: Gray
Zoom: 1.000
3D look: ✓
Scroll bars: ✓
ScrollX: 0
ScrollY: 0

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

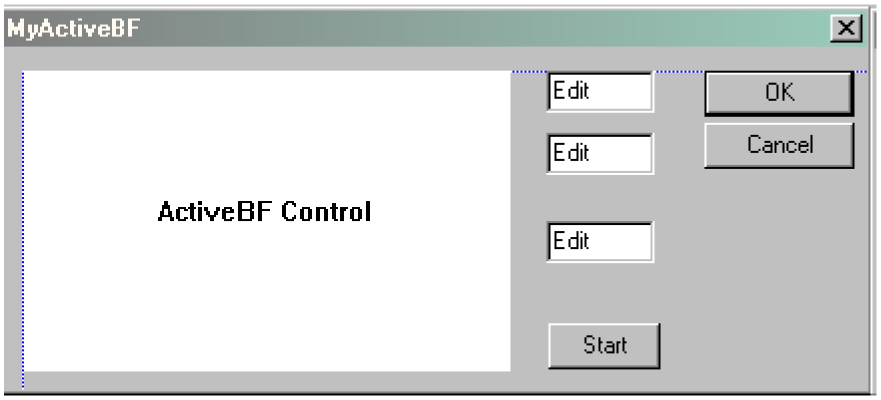
VC++ 2005, 2008, 2010

```
void CMyActiveBFDlg::OnBnClickedButton1()
{
  // TODO: Add your control notification handler code here
  m_ActiveBF.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 ActiveBF 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 *OnMouseMoveActivebf1* will be added to the source code. Add the following lines to the body of the function:

```
void CMyActiveBFDlg::OnMouseMoveActivebf1(short x, short y)
{
  // TODO: Add your control notification handler code here
  SetDlgItemInt(IDC_EDIT1,x);
  SetDlgItemInt(IDC_EDIT2,y);
  SetDlgItemInt(IDC_EDIT3,m_ActiveBF.GetPixel(x,y));
}
```

Running the application

From the *Build* menu of the development environment select *Execute MyActiveBF.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!

-

2.3 VB.NET

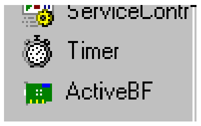
This chapter shows you how to get started with *ActiveBF* control in VB.NET. With just a few mouse clicks and a few lines of code, you will be able to display a live video image in your VB.NET program, access the array of pixel values and display them in a table in real time.
.

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* filed below type the name of your application, for instance *MyActiveBF* and click *OK*. The project will be created, and the main application form will be displayed for editing.

Creating the Control

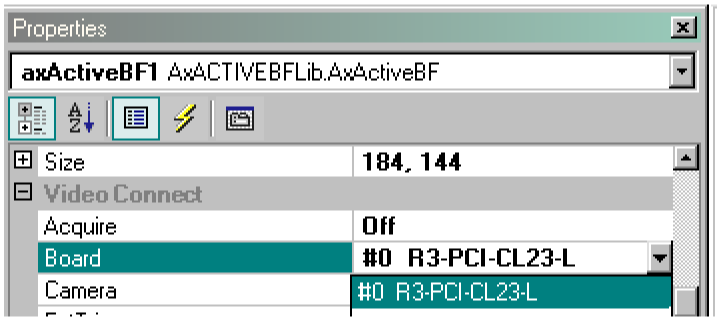
In the *Toolbox* select *Components.* From the *Tools* menu select *Customize Toolbox* and then select *ActiveBF Class* from the list. You will see *ActiveBF* icon appear at the bottom of the toolbox.



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

Selecting the Board

In the properties window, click the *Board* property. The list box will display BitFlow boards installed on your system. Select the one you intend to use:

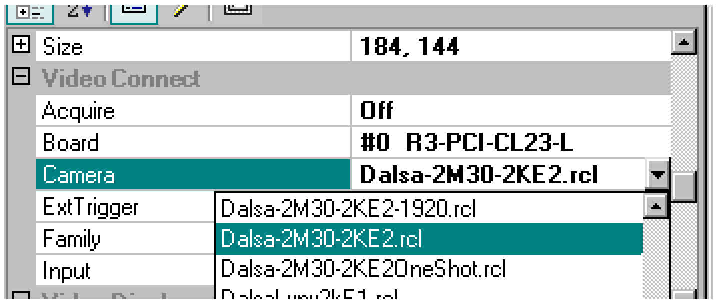


axActiveBF1 AxACTIVEBFLib.AxActiveBF

Size	184, 144
Video Connect	
Acquire	Off
Board	#0 R3-PCI-CL23-L
Camera	#0 R3-PCI-CL23-L

Selecting the Camera

In the *Properties* window, click the *Camera* property. The list box will display all the camera configuration files available for the chosen board. Select the one you intend to use:



Size	184, 144
Video Connect	
Acquire	Off
Board	#0 R3-PCI-CL23-L
Camera	Dalsa-2M30-2KE2.rcl
ExtTrigger	Dalsa-2M30-2KE2-1920.rcl
Family	Dalsa-2M30-2KE2.rcl
Input	Dalsa-2M30-2KE2OneShot.rcl

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
        AxActiveBF1.Acquire = True
End Sub
```

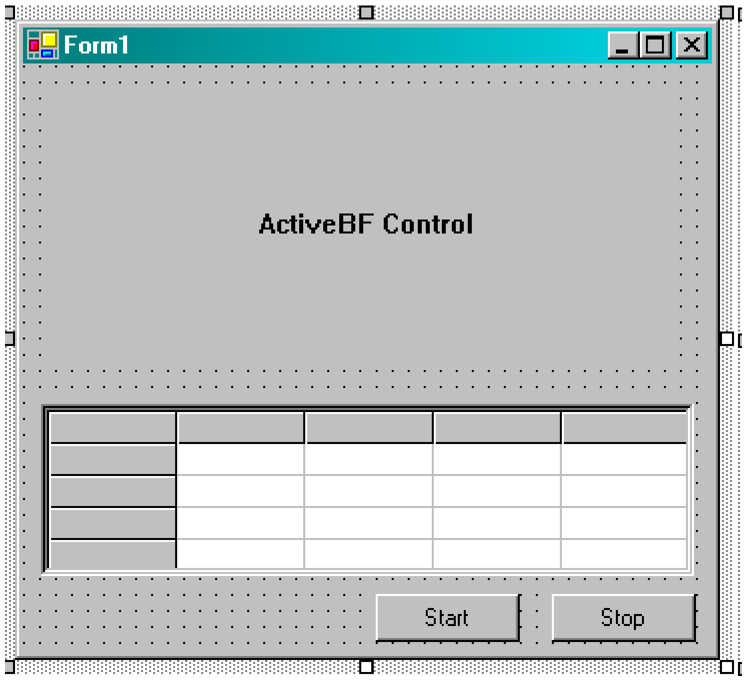
Repeat the same procedure for the *Stop* button adding the following line to the *Button2_Click* function:

```
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
Button1.Click
        AxActiveBF1.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 raws 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 ActiveBF control window. The code window will appear with an empty *AxActiveBF1_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 AxActiveBF1_FrameAcquired(ByVal sender As System.Object, ByVal e As
      AxACTIVEBFLib._IActiveBFEvents_FrameAcquiredEvent) Handles AxActiveBF1.FrameAcquired
       Dim A As Array
       Dim X As Integer
       Dim Y As Integer
       A = AxActiveBF1.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

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.

2.4 Visual C#

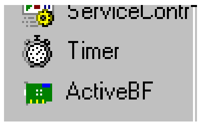
This chapter shows you how to get started with *ActiveBF* control in Visual C#. With just a few mouse clicks and a few lines of code, you will be able to display a live video image in your C# program and report a value of a pixel pointed by a mouse cursor in real time.
.

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* filed below type the name of your application, for instance *MyActiveBF* and click *OK*. The project will be created, and the main application form will be displayed for editing.

Creating the Control

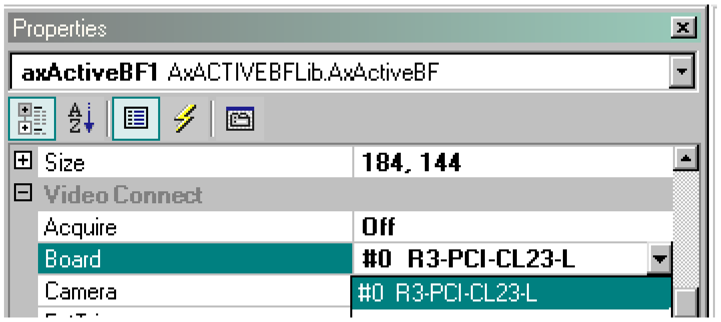
In the *Toolbox* select *Components.* From the *Tools* menu select *Customize Toolbox* and then select *ActiveBF Class* from the list. You will see *ActiveBF* icon appear at the bottom of the toolbox.



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

Selecting the Board

In the properties window, click the *Board* property. The list box will display BitFlow boards installed on your system. Select the one you intend to use:

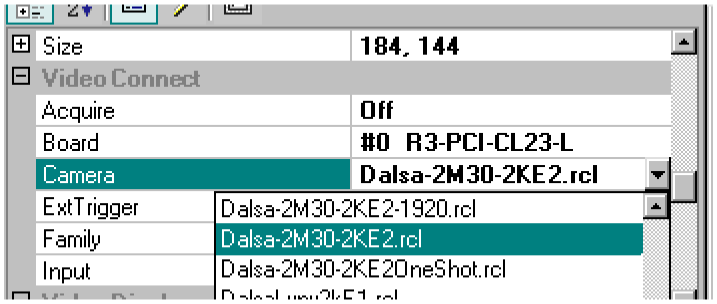


axActiveBF1 AxACTIVEBFLib.AxActiveBF

Size	184, 144
Video Connect	
Acquire	Off
Board	#0 R3-PCI-CL23-L
Camera	#0 R3-PCI-CL23-L

Selecting the Camera

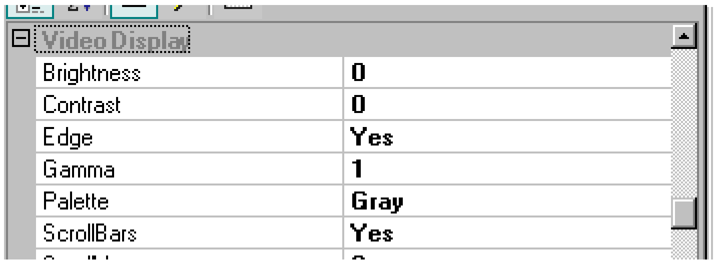
In the *Properties* window, click the *Camera* property. The list box will display all the camera configuration files available for the chosen board. Select the one you intend to use:



Size	184, 144
Video Connect	
Acquire	Off
Board	#0 R3-PCI-CL23-L
Camera	Dalsa-2M30-2KE2.rcl
ExtTrigger	Dalsa-2M30-2KE2-1920.rcl
Family	Dalsa-2M30-2KE2.rcl
Input	Dalsa-2M30-2KE2OneShot.rcl

Modifying the Control's appearance

In the Properties window scroll down to the *Video Display* category and set the *Edge* and *ScrollBars* fields to "Yes"



Video Display	
Brightness	0
Contrast	0
Edge	Yes
Gamma	1
Palette	Gray
ScrollBars	Yes

Adding the Start button

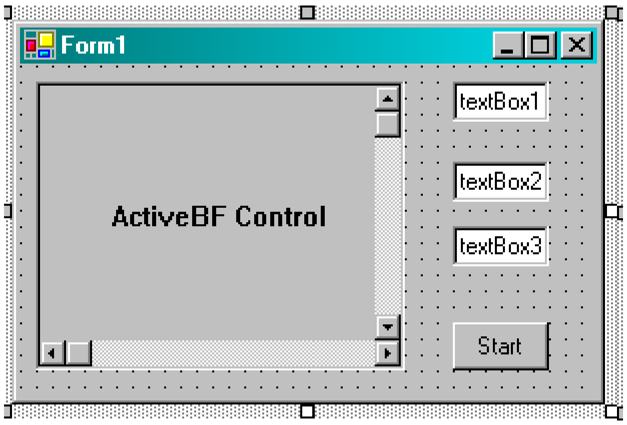
In the *Toolbox* select *Windows Form*, click on the *Button* icon and then draw a rectangular area on the program dialog. A button "Button1" will appear. Go to the *Property* window and change the *Text* property to "Start". Double click on the button. A new member function *button1_Click* will be added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

```
private void button1_Click(object sender, System.EventArgs e)
    {
      axActiveBF1.Acquire=true;
    }
```

This will activate continuous acquisition when the button is clicked.

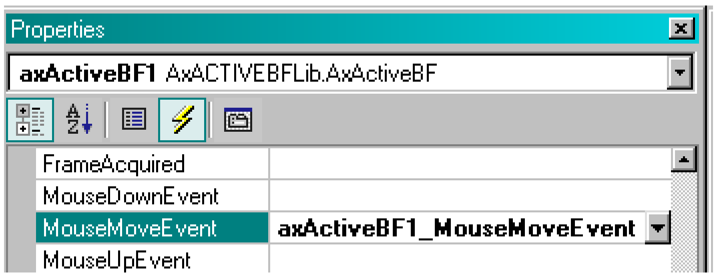
Adding text boxes

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



axActiveBF1 AxACTIVEBFLib.AxActiveBF

FrameAcquired	
MouseDownEvent	
MouseMoveEvent	axActiveBF1_MouseMoveEvent
MouseUpEvent	

The new event handler *axActiveBF1_MouseMoveEvent* will be added to the source code. Add the following lines to the body of the function:

```
private void axActiveBF1_MouseMoveEvent(object sender,
AxACTIVEBFLib._IActiveBFEvents_MouseMoveEvent e)
    {
      textBox1.Text=ToString(e.x);
      textBox2.Text=e.y;
      textBox3.Text=axActiveBF1.GetPixel(e.x,e.y);

    }
```

Running the application

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!

2.5 Using ActiveBF API at runtime

Most applications would use *ActiveBF* 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 *ActiveBF* SDK dynamically at runtime. This can be useful if you want a user of your application to decide if he wants to use IIDC-1394 cameras or if you are creating a dynamic link library that has no GUI. ActiveBF easily allows you to do it via its COM programming interface.

C/C++

1. Copy ActiveBF_i.c and ActiveBF.h files into your project folder and include them into your project.

2. Add the following defines to a mudule which will be using *ActiveBF* functions:

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

3. Initialize COM library and instantiate an ActiveBF object:

```
IActiveBF *pActiveBF;
HRESULT hr = CoInitialize(0);
if (hr==S_OK)
{
   hr = CoCreateInstance(CLSID_ActiveBF, NULL, CLSCTX_INPROC_SERVER, IID_IActiveBF,
(void**) &pActiveBF);
}
```

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

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

Refer to *BFConsole* and *BFWin* sample applications for more details.

VB6

1. Add the *ActiveBF* component to the Toolbox as described in Getting started in VB.

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

```
Dim WithEvents AB As ActiveBF
```

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

```
Set AB = New ActiveBF
```

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

```
CameraList=AB.GetCameraList         'retreiving the list of names of connected cameras
AB.Camera=0                         'selecting camera #0
AB.SetFeature "GainRaw", 150        'setting Gain value
AB.Acquire=True                     'initiating the acquisition
```

5. To add an ActiveBF event handler to your code, select the AB 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 AB_FrameAcquired()
End Sub
```

Refer to *BFByRef* sample application for more details.

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

```
Set AB = Nothing
```

VB.NET

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

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

```
Dim WithEvents AB As ActiveBFLib.ActiveBF
```

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

```
AB = New ActiveBFLib.ActiveBF
```

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

```
AB.Family = "R64"
BrdLst = AB.GetBoardList                'retreiving board list
BrdNumber = UBound(BrdLst)                      'number of boards found
For i = 0 To BrdNumber                  'filling out combo box with the names of boards
   ComboBox1.Items.Add(BrdLst(i))
Next
 .....
 AB.Board = 0
 AB.Acquire = True
```

5. To add an ActiveBF event handler to your code, select the AG 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 AB_FrameAcquired(ByVal Lines As Short) Handles AB.FrameAcquired
End Sub
```

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

```
AB = Nothing
```

Refer to *BFNetRef* sample application for more details.

C#

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

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

```
ActiveBFLib.ActiveBF AB;
```

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

```
AB = new ActiveBFLib.ActiveBF();
```

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

```
AB.Board = 0
AB.Acquire=true;                  'starting automatic acquisition
AB.ShowProperties (true, 0);   'displaying buit-in property pages
```

2.6 Working with multiple cameras

In general, interfacing to multiple BitFlow boards is as easy as dropping a few *ActiveBF* 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 *ActiveBF* object for a different board by clicking a corresponding *ActiveBF* window and modifying the *Board* property. *Do not configure different ActiveBF objects for the same board. Multiple instances of ActiveBF cannot acquire video from the same board.* For the same reason, if you run several ActiveBF-based applications, make sure that each of them connects to a different board.

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

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

```
AxActiveBF1.Acquire = True
AxActiveBF2.Acquire = True
AxActiveBF3.Acquire = True
```

5. For more information refer to the code of the *TwoBoards* 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 board. Refer to DirectShow Programming Reference for more information.

3 ActiveX Reference

Properties
Methods
Events
Property Pages

3.1 Properties

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

Appearance Category

BackColor	Returns or sets the background color of the control window
Display	Enables/disables automatic live display in the control window.
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

Video Connect Category

Family	Returns or sets the family (product line) of BitFlow boards selected for the video capture
Board	Returns or sets the ordinal number of the currently selected BitFlow board
Camera	Returns or sets the camera connected to the current board
Input	Returns or sets the number of the analog video input for the Raven board
ExtTrigger	Enables/disables the external hardware trigger connected to the acquisition circuitry
Acquire	Enables/disables the continuous acquisition mode.
LineScan	Enables/disables the video update per each captured line

Video Format Category

Format	Returns or sets the depth of pixel values
SizeX	Returns or sets the width of the video frame
SizeY	Returns or sets the height of the video frame
Encoder	Enables/disables the external horizontal synchronization mode for line-scan cameras
EncoderInput	Returns or sets the encoder input for the R64 board.
Trigger	Enables/disables the trigger mode.
TriggerInput	Returns or sets the hardware trigger input for the R64 board
StartStop	Enables/disables the start-stop mode for line-scan cameras
Timeout	Returns or sets the current acquisition timeout in milliseconds
Asynch	Enables/disables the asynchronous mode for video acquisition
Bayer	Enables/disables the Bayer conversion mode and selects conversion method
BayerLayout	Returns or sets the pixel layout in the Bayer array

Video Display Category

Palette	Returns or sets the number of the current live video palette
Zoom	Returns or sets the zoom factor for the live video display
Display	Enables/disables automatic live display in the control window.
AntiTearing	Enables/disables the anti-tearing feature for the live video display.
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 live video
ScrollY	Returns or sets the current vertical scroll position for the live video
Brightness	Returns or sets the brightness of the current board's look-up table
Contrast	Returns or sets the contrast of the current board's look-up table
Gamma	Returns or sets the gamma of the current board's look-up table
Flip	Flips the image horizontally or/and vertically
Rotate	Rotates the image at the specified angle
BkgName	Returns or sets the name prefix for background data
BkgCorrect	Resturns or sets the background correction mode
HotPixelCorrect	Enables/disables the hot pixel correction mode
HotPixelLevel	Returns or sets the threshold level to be used in the hot pixel correction mode
Integrate	Enables/disables the frame integration operation and selects the integration mode
IntegrateWnd	Returns or sets the number of frames to be used in the Integrate operation
LUTMode	Enables/disables the lookup table operation on the video frames

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]
```
objActiveBF.Acquire [= Value]
```

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

Data Type [VB]

Boolean

Parameters [C/C++]

pAcquire [out,retval]
Pointer to the Boolean that is TRUE if the continuous acquisition is enable, or FALSE otherwise
Acquire [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
    ActiveBF1.Acquire=TRUE
    Exit Sub
err_cmdStart:
    MsgBox Err.Description
End Sub
```

Remarks

If this property is set to TRUE, the board 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, and GetImageData methods.

3.1.2 Alpha

Description

Shows/hides the alpha plane over the live video.

Syntax

[VB]
```
objActiveBF.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()
ActiveBF1.Acquire=True
ActiveBF1.Alpha=True
End Sub

Private Sub ActiveBF1_FrameAcquired()
ActiveBF1.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
```

```
ActiveBF1.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 ActiveBF 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.

3.1.3 AntiTearing

Description

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

Syntax

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

[C/C++]
```
HRESULT get_AntiTearing ( bool *pAntiTearing );
HRESULT put_AntiTearing( bool AntiTearing );
```

Data Type [VB]

Boolean

Parameters [C/C++]

pAntiTearing [out,retval]
Pointer to the Boolean that is TRUE if the anti-tearing is enable, or FALSE otherwise
AntiTearing [in]
Set to TRUE to enable the 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:

```
ActiveBF1.AntiTearing = True
MsgBox ActiveBF1.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 higly 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.

3.1.4 Asynch

Description

Enables/disables the asynchronous mode for video acquisition.

Syntax

[VB]
```
objActiveBF.Asynch [= Value]
```

[C/C++]
```
HRESULT get_Asynch ( bool *pAsynch );
HRESULT put_Asynch( bool pAsynch );
```

Data Type [VB]

Boolean

Parameters [C/C++]

pAsynch [out,retval]
Pointer to the Boolean that is TRUE if the asynchronous acquisition is enable, or FALSE otherwise
Asynch [in]
Set to TRUE to enable the asynchronous acquisition, or set to FALSE for synchronous acquisition

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example sets the acquisition in the asynchronous mode:

```
ActiveBF1.Asynch = TRUE
MsgBox ActiveBF1.Asynch
```

Remarks

If this property is set to TRUE, the board will continuously transfer pixels into the internal image memory, allowing your application to perform other tasks while the acquisition of the next frame occurs. The asynchronous mode provides the fastest and most efficient setup for real-time image processing. However, if processing occurs at a slower rate than pixels are acquired, using this mode may result in the decomposition of images and loss of data.

Setting the **Asynch** property to FALSE will set the board to the synchronous mode. In this mode the acquisition of the next frame will be initiated upon calling the Grab method. The synchronous mode is slower, but it guarantees the wholeness of images during real-time processing.

3.1.5 BackColor

Description

Returns or sets the background color of the control window.

Syntax

[VB]
```
objActiveBF.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:

```
ActiveBF1.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 *ActiveBF*.

3.1.6 Bayer

Description

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

Syntax

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

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

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 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 missign 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:

```
ActiveBF1.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 (demosaicing). The layout of the Bayer array can be selected with the BayerLayout property.

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.

Note that this property is available only if the current video Format is a monochrome one.

3.1.7 BayerLayout

Description

Returns or sets the layout of pixels in the CCD array of a Bayer camera. The values from 0 to 3 correspond to the following layouts:

```
0 - GB
  G  B  G  B  G
  R  G  R  G  R

1 - GR
  G  R  G  R  G
  B  G  B  G  B

2 - BG
  B  G  B  G  B
  G  R  G  R  G

0 - RG
  R  G  R  G  R
  G  B  G  B  R
```

Syntax

[VB]
```
objActiveBF.BayerLayout [= Value]
```

[C/C++]
```
HRESULT get_BayerLayout( long *pBayerLayout );
HRESULT put_BayerLayout( long BayerLayout );
```

Data Type [VB]

Long

Parameters [C/C++]

pBayerLayout [out,retval]
  Pointer to the ordinal number of the currently selected layout
BayerLayout [in]
  The number of the layout 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 the Bayer layout:

```
Private Sub Form_Load()
ActiveBF1.Acquire = True
ActiveBF1.Bayer = True
Combo1.AddItem ("GB")
Combo1.AddItem ("GR")
Combo1.AddItem ("BG")
Combo1.AddItem ("RG")
Combo1.ListIndex = 1
End Sub

Private Sub Combo1_Click()
ActiveBF.BayerLayout = Combo1.ListIndex
End Sub
```

Remarks

This property works in combination with the Bayer filter. The layout of the CCD array is defined by your Bayer camera specifications and the setting of the camera file used. Depending on the start of the horizontal scan you might need to select a different Bayer layout, as the position of the top left corner of the video window relative to the CCD mask can change.

Note that this property is available only if the current video Format is a monochrome one.

3.1.8 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]
```
objActiveBF.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
BitShift [in]
Background correction mode to be set

Return Values

S_OK
Success

Example

The following VB example sets the dark field correction mode:

```
ActiveBF1.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

$I_{x.v}$	is a pixel value of the original image
$B_{x.v}$	is a pixel value of the dark field image
$W_{x.v}$	is a pixel value of the bright field image
W_{max}	is a maximum value of the bright field image
$C_{x.v}$	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.

3.1.9 BkgName

Description

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

Syntax

[VB]
```
objActiveBF.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:

```
ActiveBF1.BkgName="MyBackground"
ActiveBF1.SaveBkg (1)
```

Remarks

Bakground data are stored as raw image files in the Bkgnd subfolder of ActiveBF program directory (typically C:\Program Files\ActiveBF\Bkgnd). The full name of a background file is composed of the **BkgName** prefix and substrings indicating the board index, video format and background type. E.g., if the background name is assinged as in the example above and the board operates in the 12-bit format, the bakground image file will be stored under the name "MyBackground_board0_12bit_dark.raw"

3.1.10 Board

Description

Returns or sets the ordinal number of the currently selected BitFlow board (physical or virtual). Boards of the selected Family are numbered sequentially starting from zero as they are found when the system boots.

Syntax

[VB]
```
objActiveBF.Board [= Value]
```

[C/C++]
```
HRESULT get_Board( long *pBoard );
HRESULT put_Board( long Board );
```

Data Type [VB]

Long

Parameters [C/C++]

pBoard [out,retval]
Pointer to the number of the currently selected board
Board [in]
The number of the board to be selected

Return Values

S_OK
Success
E_FAIL
Failure.
E_INVALIDARG
Invalid property value.

Example

This VB example sets the current board number to 1:

```
ActiveBF1.Family = "R64"
ActiveBF1.Board = 1
MsgBox ActiveBF1.Board
```

Remarks

The valid **Board** property values are 0-15. A given board will be the same number every time the system boots, as long as the quantity of boards remains the same and they remain in the same PCI slots. Note that certain BitFlow framegrabbers (such as Karbon) are treated by the system as several virtual boards.

A proper framegrabber Family must be set before selecting the **Board**.

If you use property pages in your development environment, the **Board** property field will be presented as a list box containing all the boards of the current Family that are installed on your system. A typical selection in the Board field of a Property window will read #0 R64-PCI-CL which indicates that the first R64 board is currently selected for the video capture. If you have more than one board installed on your system, you can switch to another board of the same family (in the above example, another R64) by choosing the corresponding board from the list.

If you use more than one *ActiveBF* control in your application, make sure each of them is connected to a different board, or errors will occur.

3.1.11 Brightness

Description

Returns or sets the brightness of the current board's look-up table.

Syntax

[VB]
```
objActiveBF.Brightness [= Value]
```

[C/C++]
```
HRESULT get_Brightness( long *pBrightness );
HRESULT put_Brightness( long Brightness );
```

Data Type [VB]

Long

Parameters [C/C++]

pBrightness [out,retval]
Pointer to the current brightness
Brightness [in]
The brightness to be set.

Return Values

S_OK
Success
E_FAIL
Failure.
E_INVALIDARG
Invalid property value.

Example

This VB example sets the hardware brightness to 32:

```
ActiveBF1.Brightness = 32
MsgBox ActiveBF1.Brightness
```

Remarks

This property changes the brightness of the video by modifying the hardware look-up table. The valid property values are from -100 to +100. If the current Board does not have a look-up table, the **Brightness** property will be unavailable.

3.1.12 Camera

Description

Returns or sets the name of the camera configuration file for the current board.

Syntax

[VB]
```
objActiveBF.Camera [= strCamera]
```

[C/C++]
```
HRESULT get_Family( BSTR *pstrCamera );
HRESULT put_Family( BSTR strCamera );
```

Data Type [VB]

String

Parameters [C/C++]

pstrpCamera [out,retval]
Pointer to the string containing the currently selected camera file
strCamera [in]
The string containing the camera file to be selected

Return Values

S_OK
Success
E_FAIL
Failure.
E_INVALIDARG
Invalid property value.

Example

This VB example sets the synthetic camera file for the R64 board.

```
ActiveBF1.Camera = "Bitflow-Synthetic-1024x1024-E1.r64"
MsgBox ActiveBF1.Camera
```

Remarks

The valid **Camera** property values for the RoadRunner, Raven, R3-CL, and R3 boards must be the respective names of the camera files from CAM, RVC, RCL and R64 subfolders of the BitFlow SDK camera configuration folder (defined in SysReg application as the Camera configuration file path). In addition, you can select the default camera file by using the string "_default_" as a property value. In this case *ActiveBF* control will use the first camera from the list maintained by the SysReg application. See *BitFlow SDK User Guide* for more details.

If you use property pages in your development environment, the **Camera** property field will be presented as a list box containing all the camera configuration files available for the currently selected Board with the default camera string on top of the list.

3.1.13 Contrast

Description

Returns or sets the contrast of the current board's look-up table.

Syntax

[VB]
```
objActiveBF.Contrast [= Value]
```

[C/C++]
```
HRESULT get_Contrast( long *pContrast );
HRESULT put_Contrast( long Contrast );
```

Data Type [VB]

Long

Parameters [C/C++]

pContrast [out,retval]
Pointer to the current contrast
Contrast [in]
The contrast to be set.

Return Values

S_OK
Success
E_FAIL
Failure.
E_INVALIDARG
Invalid property value.

Example

This VB example sets the hardware Contrast to 32:

```
ActiveBF1.Contrast = 32
MsgBox ActiveBF1.Contrast
```

Remarks

This property changes the contrast of the video (the difference between the dark and light areas) by modifying the hardware look-up table. The valid property values are from -100 to +100. If the current Board does not have a look-up table, the **Contrast** property will be unavailable.

3.1.14 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]
```
objActiveBF.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()
ActiveBF1.Display = False
ActiveBF1.Acquire = True
End Sub

Private Sub ActiveBF1_FrameAcquired(ByVal Lines As Integer)
a = ActiveBF1.GetImageData
For x = 0 To 200
For y = 0 To 200
a(x, y) = 255 - a(x, y)
Next
Next
ActiveBF1.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.

3.1.15 Edge

Description

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

Syntax

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

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

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:

```
ActiveBF1.Edge = TRUE
MsgBox ActiveBF1.Edge
```

Remarks

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

3.1.16 Encoder

Description

Enables/disables the external horizontal synchronization mode.

Syntax

[VB]
```
objActiveBF.Encoder [= Value]
```

[C/C++]
```
HRESULT get_Encoder ( bool *pEncoder );
HRESULT put_Encoder( bool pEncoder );
```

Data Type [VB]

Boolean

Parameters [C/C++]

pEncoder [out,retval]
Pointer to the Boolean that is TRUE if the encoder is enable, or FALSE otherwise
Encoder [in]
Set to TRUE to enable the encoder, or set to FALSE otherwise

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example activates the encoder circuitry:

```
ActiveBF1.Encoder = TRUE
MsgBox ActiveBF1.Encoder
```

Remarks

The encoder mode is used with a line scan camera, the horizontal synchronization for which is provided by a motion encoder. The encoder generates a trigger signal at regular spatial intervals, so that the lines captured are synchronous with movement of an object in the field of view. To set the board to the internal horizontal synchronization mode, set the **Encoder** property to FALSE. Note that this property is available only for Camera configuration files that have an encoder support.

3.1.17 EncoderInput

Description

Returns or sets the encoder input for the R64 board.

Syntax

[VB]

```
objActiveBF.EncoderInput [= Value]
```

[C/C++]

```
HRESULT get_EncoderInput( long *pEncoderInput );
HRESULT put_EncoderInput( long EncoderInput );
```

Data Type [VB]

Long

Parameters [C/C++]

pEncoderInput [out,retval]
Pointer to the number of the currently selected encoder input
EncoderInput [in]
The number of the encoder input to be selected

Return Values

S_OK
Success
E_FAIL
Failure.
E_INVALIDARG
Invalid property value.

Example

This VB example sets the current encoder input to 1:

```
ActiveBF1.EncoderInput = 1
MsgBox ActiveBF1.EncoderInput
```

Remarks

The **EncoderInput** property is only available if the currently selected Family is R64. The allowable property values are 0-3. You can use this property to switch between several encoders connected to the R64 board.

3.1.18 ExtTrigger

Description

Enables/disables the external hardware trigger connected to the acquisition circuitry.

Syntax

[VB]
```
objActiveBF.ExtTrigger [= Value]
```

[C/C++]
```
HRESULT get_ExtTrigger ( bool *pExtTrigger );
HRESULT put_ExtTrigger( bool pExtTrigger );
```

Data Type [VB]

Boolean

Parameters [C/C++]

pExtTrigger [out,retval]
Pointer to the Boolean that is TRUE if the external trigger circuitry is enable, or FALSE otherwise
ExtTrigger [in]
Set to TRUE to enable the external trigger, or set to FALSE otherwise

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example activates the external trigger circuitry:

```
ActiveBF1.ExtTrigger = TRUE
MsgBox ActiveBF1.ExtTrigger
```

Remarks

If you do not have a hardware trigger and want to use the software trigger, set the **ExtTrigger** property to FALSE to disable the trigger circuitry. If this property is set to TRUE and no trigger is connected, the board may randomly self-trigger from the electric noise on the unconnected input.

3.1.19 Family

Description

Returns or sets the family (product line) of the BitFlow board selected for the video capture. If boards of different families are installed on the system, use this option to select the active family. Currently supported families are Raven, Road Runner/R3, and R64/Neon/Karbon/Alta.

Syntax

[VB]
```
objActiveBF.Family [= strValue]
```

[C/C++]
```
HRESULT get_Family( BSTR *pstrFamily );
HRESULT put_Family( BSTR strFamily );
```

Data Type [VB]

String

Parameters [C/C++]

pstrpFamily [out,retval]
Pointer to the string containing the name of the currently selected family
strFamily [in]
The string containing the name of the board family to be set

Return Values

S_OK
Success
E_FAIL
Failure.
E_INVALIDARG
Invalid property value.

Example

This VB example sets the current board family to the R64:

```
ActiveBF1.Family = "R64"
MsgBox ActiveBF.Family
```

Remarks

The valid **Family** property values (non case-sensitive) are:

For the Raven family: "Raven"
For the RoadRunner/R3 family: "Road Runner" or "R3"
For the R64/Neon/Karbon/Alta family: "R64", "Neon", "Karbon" or "Alta"

If you try to set an invalid value, an error will occur.

3.1.20 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]
```
objActiveBF.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()
ActiveBF1.Acquire = True
Combo1.AddItem ("None")
Combo1.AddItem ("Horizontal")
```

```
Combo1.AddItem ("Vertical")
Combo1.AddItem ("Diagonal")
Combo1.ListIndex = 0
End Sub

Private Sub Combo1_Click()
ActiveBF1.Flip = Combo1.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.

3.1.21 Font

Description

Returns or sets the font for DrawText.

Syntax

[VB]
```
objActiveBF.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 ActiveBF1_FrameAcquired()
Dim Font1 As New StdFont
Font1.Name = "Arial"
Font1.Size = 30
Font1.Bold = True
ActiveBF1.Font = Font1
ActiveBF1.DrawText 10, 100, "ActiveBF", 255, 0, 0
Font2.Name = "Arial"
Font2.Size = 20
Font2.Italic = True
ActiveBF1.Font = Font2
ActiveBF1.DrawText 10, 200, "ActiveBF Control", 0, 0, 255
End Sub
```

Remarks

Also see DrawText.

3.1.22 Format

Description

Returns or sets the depth of pixel values returned by image access methods.

Syntax

[VB]
```
objActiveBF.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 currently selected pixel depth
Format [in]
The pixel depth to be selected

Return Values

S_OK
Success
E_FAIL
Failure.
E_INVALIDARG
Invalid property value.

Example

This VB example sets the current pixel depth to 10:

```
ActiveBF1.Format = 10
MsgBox ActiveBF1.Format
```

Remarks

The **Format** property can be changed only for high bit depth images (10-, 12-, 14-, 16-bit grayscale and 30-, 36-, 42-, 48-bit color images). Changing the value of the property does not affect the internal image memory, but it will make the image access methods (GetPixel, GetLine, GetImageData, etc.) rescale the output data so that they match the selected format. For example, if the camera delivers 10-bit images and the Format property is set to 16, the data will be remapped from the 0-1023 range to 0-65535.

If you use property pages in your development environment, the **Format** property field will be presented as a list box containing all the formats available for the current Camera configuration file.

3.1.23 HotPixelCorrect

Description

Enables/disables the hot pixel correction mode. Used in combination with HotPixelLevel.

Syntax

[VB]
```
objActiveBF.HotPixelCorrect [= 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:

```
ActiveBF1.HotPixelLevel = 15
ActiveBF1.HotPixelCorrect = True
```

Remarks

Hot pixels are individual pixels that appear much brighter than the rest of an image. They are assosiated with elements on a camera sensor that have higher than normal rates of charge leakage. Hot pixels are especially noticable 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.

3.1.24 HotPixelLevel

Description

Returns or sets the threshold level to be used in the hot pixel correction mode, in percents.

Syntax

[VB]
```
objActiveBF.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:

```
ActiveBF1.HotPixelLevel = 15
ActiveBF1.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.

3.1.25 Gamma

Description

Returns or sets the gamma level of the current board's look-up table.

Syntax

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

[C/C++]
```
HRESULT get_Gamma( float *pGamma );
HRESULT put_Gamma( float Gamma );
```

Data Type [VB]

Float

Parameters [C/C++]

pGamma [out,retval]
Pointer to the current gamma level
Gamma [in]
The gamma level to be set.

Return Values

S_OK
Success
E_FAIL
Failure.
E_INVALIDARG
Invalid property value.

Example

This VB example sets the hardware gamma level to 2.4:

```
ActiveBF1.Gamma = 2.4
MsgBox ActiveBF1.Gamma
```

Remarks

This property changes the gamma level of the video by modifying the hardware look-up table. The gamma correction modifies an image by applying standard, nonlinear gamma curves to the intensity scale. A gamma value of 1 is equivalent to the identity curve that does not affect the image. To lighten the video and increase the contrast in its darker areas, set the **Gamma** to a value greater than 1. To darken the video and emphasize the contrast in the lighter areas, use a value less than 1. The valid property values are from 0 to 100.0 If the current Board does not have a look-up table, the **Gamma** property will be unavailable.

3.1.26 Input

Description

Returns or sets the number of the analog video input for the Raven board.

Syntax

[VB]
```
objActiveBF.Input [= Value]
```

[C/C++]
```
HRESULT get_Input( long *pInput );
HRESULT put_Input( long Input );
```

Data Type [VB]

Long

Parameters [C/C++]

pInput [out,retval]
Pointer to the number of the currently selected input
Input [in]
The number of the input to be selected

Return Values

S_OK
Success
E_FAIL
Failure.
E_INVALIDARG
Invalid property value.

Example

This VB example sets the current input number to 1:

```
ActiveBF1.Input = 1
MsgBox ActiveBF1.Input
```

Remarks

The **Input** property is only valid if the currently selected Family is Raven. The allowable property values are 0-3. You can use this property to switch between several analog cameras connected to the Raven board.

3.1.27 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 IntegrateWnd.

Syntax

[VB]
```
objActiveBF.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:

```
ActiveBF1.IntegrateWnd=16
ActiveBF1.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.

3.1.28 IntegrateWnd

Description

Returns or sets the number of frames to be used in the Integrate operation.

Syntax

[VB]
```
objActiveBF.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:

```
ActiveBF1.IntegrateWnd=16
ActiveBF1.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.

3.1.29 LineScan

Description

Enables/disables the continuous update of the video window per each captured horizontal line.

Syntax

[VB]
```
objActiveBF.LineScan [= Value]
```

[C/C++]
```
HRESULT get_LineScan ( bool *pLineScan );
HRESULT put_LineScan( bool pLineScan );
```

Data Type [VB]

Boolean

Parameters [C/C++]

pLineScan [out,retval]
Pointer to the Boolean that is TRUE if the LineScan mode is enable, or FALSE otherwise
Edge [in]
Set to TRUE to enable the LineScan mode, or set to FALSE to disable it

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example activates the LineScan mode:

```
ActiveBF1.LineScan = TRUE
MsgBox ActiveBF1.LineScan
```

Remarks

Normally when set to the **Acquire** mode, *ActiveBF* will update its window upon each frame captured into the internal memory. When this property is set to TRUE, the control will update its window per each captured horizontal line of pixels, thus allowing you to display live video acquired by line scan cameras set to a slow line rate. Enabling the **LineScan** mode will also make the control fire the LineAcquired events.

This property should be set to FALSE when working with area scan and fast-rate line scan devices. Enabling it in these cases can significantly reduce the performance of your application.

3.1.30 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]
```
objActiveBF.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 overlay, or set to FALSE to disable it.

Return Values

S_OK
Success
E_FAIL
Failure.

Example

The following VB example enables the LUT mode and assigns gain levels for color channels :

```
ActiveBF1.LUTMode = True
ActiveBF1.SetGains 2.5, 2.8, 2.4
```

Remarks

This property works in combination with the SetLUT or SetGains methods.

3.1.31 Overlay

Description

Enables/disables the overlay.

Syntax

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

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

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:

```
ActiveBF1.Acquire = True
ActiveBF1.OverlayEllipse 100,100,200,200
ActiveBF1.OverlayColor=RGB(255,0,0)
ActiveBF1.Overlay=  True
```

Remarks

Overlay feature allows you to display custom graphics and text on the live video image. Setting this property to true causes ActiveBF 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.

3.1.32 OverlayColor

Description

Returns or sets the color of the overlay graphics.

Syntax

[VB]
```
objActiveBF.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:

```
ActiveBF1.Acquire = True
ActiveBF1.OverlayEllipse 100,100,200,200
ActiveBF1.OverlayColor=RGB(255,0,0)
ActiveBF1.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.

3.1.33 OverlayFont

Description

Returns or sets the font for OverlayText.

Syntax

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

[C/C++]
```
HRESULT get_OverlayColor(IFontDisp* *pFont);
HRESULT put_OverlayColor(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
ActiveBF1.OverlayFont = Font
ActiveBF1.OverlayText 10, 100, "ActiveBF rules!"
ActiveBF1.OverlayColor = RGB(255, 0, 0)
ActiveBF1.Overlay = True
```

Remarks

Also see OverlayColor, OverlayText.

3.1.34 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 colorized 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 colorized. 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.

Syntax

[VB]
```
objActiveBF.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 selects the inverse palette by setting the property to 1

```
ActiveBF1.Palette = 1
MsgBox ActiveBF1.Palette
```

Remarks

The **Palette** property is used for viewing different kinds of video in pseudo-colors. The property is only valid for grayscale Camera configuration files. 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.

3.1.35 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° conunterclockwise
180 -	The image is rotated 180°
270 -	The image is rotated -90° clockwise

Syntax

[VB]
```
objActiveBF.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()
ActiveBF1.Acquire = True
Combo1.AddItem ("0°")
Combo1.AddItem ("90°")
Combo1.AddItem ("180°")
Combo1.AddItem ("270°")
Combo1.ListIndex = 0
End Sub

Private Sub Combo1_Click()
ActiveBF1.Rotate = Combo1.ListIndex * 90
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.

3.1.36 ScrollBars

Description

Enables/disables the scroll bars on the control window.

Syntax

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

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

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:

```
ActiveBF1.ScrollBars = TRUE
MsgBox ActiveBF1.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.

3.1.37 ScollX

Description

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

Syntax

[VB]
```
objActiveBF.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:

```
ActiveBF1.ScrollX = 512
MsgBox ActiveBF1.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.

3.1.38 ScrollY

Description

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

Syntax

[VB]
```
objActiveBF.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:

```
ActiveBF1.ScrollY = 512
MsgBox ActiveBF1.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.

3.1.39 SizeX

Description

Returns or sets the width of the video frame.

Syntax

[VB]
```
objActiveBF.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:

```
ActiveBF1.SizeX = 512
MsgBox ActiveBF1.SizeX
```

Remarks

The **SizeX** property lets you change the image width defined by the current Camera configuration file. The valid range of the image width depends on the camera associated with the board. For example, if you exceed the sensor size of the camera, you will end up with a scrambled or unusable image. Also, this property may not work with complex and/or multi-tap cameras.

3.1.40 SizeY

Description

Returns or sets the height of the video frame.

Syntax

[VB]
```
objActiveBF.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:

```
ActiveBF1.SizeY = 512
MsgBox ActiveBF1.SizeY
```

Remarks

The **SizeY** property lets you change the image height defined by the current Camera configuration file. The valid range of the image height depends on the camera associated with the board. For example, if you exceed the sensor size of the camera, you will end up with a scrambled or unusable image. Also, this property may not work with complex and/or multi-tap cameras.

3.1.41 StartStop

Description

Enables/disables the start-stop mode.

Syntax

[VB]
```
objActiveBF.StartStop [= Value]
```

[C/C++]
```
HRESULT get_StartStop ( bool *pStartStop );
HRESULT put_StartStop( bool pStartStop );
```

Data Type [VB]

Boolean

Parameters [C/C++]

pStartStop [out,retval]
Pointer to the Boolean that is TRUE if the start-stop mode is enable, or FALSE otherwise
StartStop [in]
Set to TRUE to enable the start-stop mode, or set to FALSE otherwise

Return Values

S_OK	Success
E_FAIL	Failure.

Example

This VB example activates the start-stop mode:

```
ActiveBF1.StartStop = TRUE
MsgBox ActiveBF1.StartStop
```

Remarks

The start-stop mode is typically used with a line scan camera in order to initiate and finish a capture of a variable size frame. The acquisition of a frame will start when the external trigger signal asserts and end when it de-asserts. The size of an image captured in the start/stop mode will depend on the time interval between two trigger events.

The trigger events can be simulated by calling the SwitchTrigger method twice.

To set the board to the trigger acquire mode, set the **StartStop** property to FALSE. Note that this property is available only for Camera configuration files designed to support the start-stop mode. See *BitFlow SDK Reference Manual* for more details.

3.1.42 Timeout

Description

Returns or sets the current acquisition timeout in milliseconds.

Syntax

[VB]
```
objActiveBF.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 4 sec:

```
ActiveBF1.Timeout = 4000
MsgBox ActiveBF1.Timeout
```

Remarks

The **Timeout** property lets you set the number of milliseconds to wait for a frame to be acquired. If the timeout expires, the Timeout event will be raised. If you acquire images with a slow frame rate, set this property to a higher value. If the property is set to zero, the timeout will never occur.

3.1.43 Trigger

Description

Enables/disables the trigger mode.

Syntax

[VB]
```
objActiveBF.Trigger [= Value]
```

[C/C++]
```
HRESULT get_Trigger ( bool *pTrigger );
HRESULT put_Trigger( bool pTrigger );
```

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_FAIL
Failure.

Example

This VB example activates the trigger mode:

```
ActiveBF1.Trigger = TRUE
MsgBox ActiveBF1.Trigger
```

Remarks

When the **Trigger** property is set to TRUE, the board will be set to either one shot or trigger acquire mode depending on the Camera configuration file. The one shot mode is typically used with an asynchronously resetable camera. The acquisition of a frame will occur upon receiving a signal from an external hardware trigger. The trigger acquire mode will be used for free-run camera configuration files. In this mode the trigger event will cause the board to acquire an image at the start of the next frame. Note that the trigger acquire mode will introduce a latency of up to a frame between the trigger and the beginning of the acquisition of the frame. For time critical applications it is recommended that a one shot camera file is used.

The trigger event can be simulated by calling the SwitchTrigger method.

To set the board to the free-run mode, set the **Trigger** property to FALSE. Note that not all camera files support switching between the trigger and free-run modes. It is recommended to select a camera configuration file specifically designed for the desired mode. See *BitFlow SDK Reference Manual* for more details.

3.1.44 TriggerInput

Description

Returns or sets the encoder input for the R64 board.

Syntax

[VB]
```
objActiveBF.TriggerInput [= Value]
```

[C/C++]
```
HRESULT get_TriggerInput( long *pTriggerInput );
HRESULT put_TriggerInput( long TriggerInput );
```

Data Type [VB]

Long

Parameters [C/C++]

pTriggerInput [out,retval]
Pointer to the number of the currently selected trigger input
TriggerInput [in]
The number of the trigger input to be selected

Return Values

S_OK
Success
E_FAIL
Failure.
E_INVALIDARG
Invalid property value.

Example

This VB example sets the current trigger input to 1:

```
ActiveBF1.TriggerInput = 1
MsgBox ActiveBF1.TriggerInput
```

Remarks

The **TriggerInput** property is only available if the currently selected Family is R64 and the ExtTrigger property set to TRUE. The allowable **TriggerInput** values are 0-3. You can use this property to switch between several hardware triggers connected to the R64 board.

3.1.45 Zoom

Description

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

Syntax

[VB]
```
objActiveBF.Zoom [= Value]
```

[C/C++]
```
HRESULT get_Zoom( float *pZoom );
HRESULT put_Zoom( float Zoom );
```

Data Type [VB]

Float

Parameters [C/C++]

pZoom [out,retval]
Pointer to the current zoom factor
Zoom [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.

Return Values

S_OK
Success
E_FAIL
Failure.
E_INVALIDARG
Invalid property value.

Example

This VB example sets the zoom factor to 0.25:

```
ActiveBF1.Zoom = 0.25
MsgBox ActiveBF1.Zoom
```

Remarks

This property adjusts the magnification of the life video display. It does not change the content of the image data, but only its appearance in the *ActiveBF* control window. The valid property values are from 0 to 100. If the **Zoom** 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 both Display and **Zoom** 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 *ActiveBF's* built-in image decoding/display and uses GetRawData to access the raw image data.

3.2 Methods

ActiveBF control provides the following methods to a container application:

Acquisition

Grab	Grabs a single frame into the internal memory
SwitchTrigger	Asserts/deasserts the software trigger signal

Image Access

GetBytesPerPixel	Returns the number of bytes per pixel in the video
GetPixel	Returns the pixel value at the specified coordinates
GetRGBPixel	Returns the array of RGB values at the specified coordinates
GetLine	Returns the array of pixel values in the specified horizontal line
GetComponentLine	Returns the array of pixel values of the color component in the specified horizontal line
GetImageWindow	Returns the 2D array of pixel values in the specified rectangular area of the current frame
SetImageWindow	Copies the 2D array of pixel values to the selected window of the current frame
GetImageData	Returns the two dimensional array of pixel values in the currently acquired frame
GetComponentData	Returns the two dimensional array of pixel values in the specified color component
GetImagePointer	Returns the memory pointer to the specified pixel
GetDIB	Returns the handler to a Device Independent Bitmap
GetPicture	Returns the Picture object corresponding to the currently acquired frame

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)

CreateVideo	Creates an AVI file for the video recording and preallocates its size
StartVideoCapture	Starts time-lapse video capture to the previously created AVI file
StopVideoCapture	Stops video capture to the AVI file
GetAudioList	Returns the names of audio recording devices available in the system
SetAudioSource	Sets the index of the audio recording device to be used during the AVI capture
GetAudioSource	Returns the index of the currently selected audio recording device for the AVI capture
ShowAudioDlg	Displays the audio Input dialog for adjusting the mixer properties of the audio source
SetAudioLevel	Sets the recording level of the currently selected audio device
GetAudioLevel	Returns the recording level of the currently selected audio device

Sequence Capture (DVR version only)

CreateSequence	Allocates memory for capturing a sequence of frames
StartSequenceCapture	Starts acquiring a sequence of frames into the memory
StopSequenceCapture	Stops acquiring a sequence of frames into the memory
GetSequenceFrameCount	Returns the current number of frames in the memory sequence
SaveSequence	Saves the memory sequence in the specified video file
LoadSequence	Loads the specified video file into the memory sequence
GetSequenceWindow	Returns 2D-array of pixel values of the selected window in a frame in the memory sequence
GetSequenceRawData	Returns 2D-array of raw pixel values in a frame in the memory sequence
GetSequencePixel	Returns the pixel value in the selected coordinates of the frame in the memory sequence
GetSequencePointer	Returns the memory pointer to a selected pixel of the frame in the memory sequence
GetSequencePicture	Returns the Picture object corresponding to a frame in the memory sequence
GetSequenceTimestamp	Returns the timestamp of the selected frame in the memory sequence

Video and Sequence Playback (DVR version only)

OpenVideo	Opens the specified video file or memory sequence for the playback
PlayVideo	Plays the currently open video file or memory sequence
StopVideo	Stops the playback of the video file or memory sequence
CloseVideo	Closes the currently open video file
GetVideoFrameCount	Returns the number of the frame in the currently open video file or memory sequence
GetVideoPosition	Returns the position of the current frame in the open video file or memory sequence
SetVideoPosition	Seeks, extracts and display the specified frame from the open video file or sequence
GetVideoFPS	Returns the frame rate of the currently opened video file or memory sequence
SetVideoFPS	Sets the frame rate of the currently opened video file or memory sequence
GetVideoVolume	Returns the playback volume level of the currently open AVI file
SetVideoVolume	Sets the playback volume level of the currently open AVI file
SetVideoSync	Sets the playback synchronization mode of the currently open AVI file
TriggerVideo	Advances the AVI playback to the next frame

Drawing

Draw	Displays the current frame in *ActiveBF* 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 srting of text in the alpha plane

Input/Output

SerialOpen	Opens the serial device on the Camera Link board
SerialConnect	Selects the number of the serial device on the Camera Link board
SerialRead	Reads a string of characters from the currently open Camera Link serial device
SerialWrite	Writes the string of characters to the currently open Camera Link serial device
SerialClose	Closes the currently open Camera Link serial device
GetInputBit	Reads the state of the digital input bit
SetOutputBit	Sets the state of the digital output bit

Image Processing

SetROI	Sets the rectangular region of interest and luminance range
GetROI	Returns the settings of the currently selected ROI
GetImageStat	Returns the array containing statistical data of the current image frame
GetHistogram	Returns the histogram of the current image frame
SaveBkg	Stores a dark or bright background image on the hard drive
SetLUT	Assigns the array of values for the software lookup table
GetLUT	Returns the array of values for the software lookup table
SetGains	Set the levels for the software gain control
SetLevels	Set the minimum and maximum levels for the Window/Level operation
GetLevels	Returns the minimum and maximum levels for the Window/Level operation

Utilities

ShowProperties	Displays property pages in run-time
GetFPS	Returns the acquired frame rate
GetBoardHandle	Returns the handle of the currenly selected board
GetBoardList	Returns the array of BitFlow boards installed on the system

3.2.1 CloseVideo

Description

Closes the video file currently opened by OpenVideo.

Syntax

[VB]
objActiveBF.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()
ActiveBF1.OpenVideo "C:\\video1.avi"
ActiveBF1.PlayVideo
End Sub

Private Sub ActiveBF1_PlayCompleted (ByVal Frames As Long)
ActiveBF1.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]
```
objActiveBF.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_INVALIDARG
Invalid file name.

Example

This VB example demonstrates how to allocate enough memory for the sequence capture:

```
Private Sub Form_Load()
ActiveBF1.StartSequenceCapture  1000
ActiveBF1.Acquire = True
End Sub

Private Sub StartButton_Click()
ActiveBF1.StartSequenceCapture  800
End Sub

Private Sub StopButton_Click()
ActiveBF1.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 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 StartSequenceCapture and StopSequenceCapture.

Syntax

[VB]
```
objActiveBF.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()
ActiveBF1.CreateVideo  "c:\\mycapture.avi",  80000000
ActiveBF1.Acquire = True
End Sub

Private Sub StartButton_Click()
ActiveBF1.StartVideoCapture
End Sub
```

```
Private Sub StopButton_Click()
ActiveBF1.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 *ActiveBF* window.

Syntax

[VB]
```
objActiveBF.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()
ActiveBF1.Display = False
ActiveBF1.Acquire = True
End Sub

Private Sub ActiveBF1_FrameAcquired(ByVal Lines As Integer)
a = ActiveBF1.GetImageData
For x = 0 To 200
For y = 0 To 200
a(x, y) = 255 - a(x, y)
Next
Next
ActiveBF1.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]
```
objActiveBF.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
ActiveBF1.Acquire = True
ActiveBF1.Alpha = True
End Sub

Private Sub ActiveBF1_FrameAcquired()
ActiveBF1.DrawAlphaClear
ActiveBF1.DrawAlphaRectangle x, 10, x + 50, 80, 0, 255,0,0,50
x = x + 2
If x = ActiveBF1.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 elliplse in the alpha plane over the video window.

Syntax

[VB]
```
objActiveBF.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: Intager

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-transparend filled red ellipse in the alpha-plane over the live video:

```
ActiveBF1.Acquire = True
ActiveBF1.DrawAlphaEllipse 100,100,200,200,0,255,0,0,50
```

```
ActiveBF1.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]
```
objActiveBF.DrawAlphaline X1, Y1, X2, Y2, Width, Red , Green, Blue [,
Opacity]
```

[C/C++]
```
HRESULT DrawAlphaline( 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: Intager

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]
Vaule 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:

```
ActiveBF1.Acquire = True
ActiveBF1.DrawAlphaLine 100,100,200,200,3,0,255,0
ActiveBF1.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]
```
objActiveBF.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: Intager

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]
Vaule 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:

```
ActiveBF1.Acquire = True
ActiveBF1.DrawAlphaPixel 100,100,200,200,0,0,255,80
ActiveBF1.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]
```
objActiveBF.DrawAlphaRectangle X1, Y1, X2, Y2, Width, Red , Green, Blue [,
Opacity]
```

[C/C++]
```
HRESULT DrawAlphaRectangle( 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: Intager

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-transparend filled red rectangle in the alpha-plane over the live video:

```
ActiveBF1.Acquire = True
ActiveBF1.DrawAlphaRectangle 100,100,200,200,0,255,0,0,50
```

```
ActiveBF1.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]
```
objActiveBF.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: Intager

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]
Vaule 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:

```
ActiveBF1.Acquire = True
ActiveBF1.DrawAlphaText 50,100, "ActiveBF SDK for 1394 cameras", 255,255,0,50
ActiveBF1.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 rectangle in the current image frame.

Syntax

[VB]
```
objActiveBF.DrawElliplse X1, Y1, X2, Y2, Width, Red [,Green, Blue, Opacity
]
```

[C/C++]
```
HRESULT DrawEllipse( 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: Intager

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 ActiveBF1_FrameAcquired()
    ActiveBF1.DrawRectangle  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]
```
objActiveBF.DrawLine X1, Y1, X2, Y2, Width, Red [,Green, Blue, Opacity ]
```

[C/C++]
```
HRESULT DrawLine( 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: Intager

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 ActiveBF1_FrameAcquired()
    ActiveBF1.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]
```
objActiveBF.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: Intager

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 ActiveBF1_FrameAcquired()
    ActiveBF1.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]
```
objActiveBF.DrawRectangle X1, Y1, X2, Y2, Width, Red [,Green, Blue, Opacity
]
```

[C/C++]
```
HRESULT DrawRectangle( 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: Intager

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 ActiveBF1_FrameAcquired()
    ActiveBF1.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]
```
objActiveBF.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: Intager

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 ActiveBF1_FrameAcquired()
    ActiveBF1.DrawText  50,100, "ActiveBF SDK for BitFlow boards", 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 GetBoardHandle

Description

Returns the handle to the currently selected board as defined by BitFlow API.

Syntax

[VB]
```
Value=objActiveBF.GetBoardHandle()
```

[C/C++]
```
HRESULT GetBoardHandle( long* pHandle );
```

Data Types [VB]

Return value: Long

Parameters [C/C++]

pHandle
Pointer to the board handle

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example displays the board handle in a message box:

```
MsgBox ActiveBF1.GetBoardHandle
```

Remarks

This method is provided for advanced programming when BitFlow API functions are used in parallel with *ActiveBF*. For more information refer to *BitFlow SDK Reference Manual.*

3.2.17 GetBoardList

Description

Returns the array of strings containing the names of BitFlow boards (physical and virtual) installed on the system and belonging to the currently selected Family.

Syntax

[VB]
```
Value=objActiveBF.GetBoardList()
```

[C/C++]
```
HRESULT GetBoardList( 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 framegrabber names and uses it to switch between the cameras:

```
Private Sub Form_Load()
...
BrdLst = ActiveBF1.GetBoardList
For i = 0 To UBound(BrdLst)
Combo1.AddItem (BrdLst(i))
Next
Combo1.ListIndex = 0
ActiveBF1.Acquire = True
End Sub

Private Sub Combo1_Click()
ActiveBF1.Board = Combo1.ListIndex
End Sub
```

This MFC example fills out a combo box with framegrabber names:

```
VARIANT m_BrdArray=m_ActiveBF.GetBoardList();
```

```
SAFEARRAY *pArray=m_BrdArray.parray;
UINT nBoard=pArray->rgsabound[0].cElements;

CString strBoard;
CComboBox *pBoard=(CComboBox*)GetDlgItem(IDC_BOARD);
for(UINT i=0;i<nBoard;i++)
{
    CString str; str.Format("Board %d",i);
    pBoard->AddString(str);
}
int iBoard=m_ActiveBF.GetBoard();
pCamera->SetCurSel(iBoard);

SafeArrayDestroy(pArray);
```

Remarks

The index of an element in the camera list can be used as an argument of the Board property to select a specific camera.

Note that in C/C++ applications it is required to call SafeArrayDestroy() to delete the SAFEARRAY returned by **GetBoardList**.

3.2.18 GetBytesPerPixel

Description

Returns the number of bytes per pixel for the current video format.

Syntax

[VB]
```
Value=objActiveBF.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=ActiveBF1.GetBytesPerPixel()
MsgBox value
```

Remarks

The method returns the pixel depth of the internal image buffer of *ActiveBF* control which can be different from the pixel depth of the image ouputed by the camera. For instance, the 30-bit color images are always converted to the RGB 8:8:8 video, therefore the number of bytes per pixel reported for this format will be 3. Also, when the Bayer color conversion is activated for 8-bit and and 16-bit monochrome format, the resulting video will have 3 or 6 bytes per pixel accordingly.

3.2.19 GetAudioLevel

Description

Returns the recording level of the currently selected audio device.

Syntax

[VB]
```
Value=objActiveBF.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()
ActiveBF1.SetAudioSource 0
HScroll1.Min=0
HScroll1.Max=100
HScroll.Value=ActiveBF1.GetAudioLevel
End Sub

Private Sub Capture_Click
ActiveBF1.StartCapture "c:\\capture_with_sound.avi"
End Sub

Private Sub HScroll1_Scroll()
ActiveBF1.SetAudioLevel HScroll1.Value
End Sub
```

Remarks

In order for this method to work, the audio recording device must be selected with SetAudioSource.

3.2.20 GetAudioList

Description

Returns the array of strings containing the names of audio recording devices available in the system.

Syntax

[VB]
```
Value=objActiveBF.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 = ActiveBF1.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**.

3.2.21 GetAudioSource

Description

Returns the index of the currently selected audio recording device for the AVI capture.

Syntax

[VB]
```
Value = objActiveBF.GetAudioSource
```

[C/C++]
```
HRESULT GetAudioSource(short* Index);
```

Data Types [VB]

Return value: Integer

Parameters [C/C++]

None

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 = ActiveBF1.GetAudioList
index=ActiveBF1.GetAudioSource
MsgBox AudioLst(index)
```

Remarks

If the return value is -1, the audio recording is disabled.

3.2.22 GetCodec

Description

Gets the name of the currently selected codec (video compressor) for the AVI capture.

Syntax

[VB]
```
Value = objActiveBF.GetCodec
```

[C/C++]
```
HRESULT GetCodec(bstr* pName);
```

Data Types [VB]

Name: String
Return value: String

Parameters [C/C++]

pName [out, retval]
Pointer to the string specifying the name of the codec

Return Values

S_OK
Success
E_FAIL
Failure to set the codec

Example

The following VB example prints the name of the currently selected codec:

```
MsgBox ActiveBF1.GetCodec
```

3.2.23 GetCodecList

Description

Returns the array of strings containing the names of AVI codecs (compressors) available in the system.

Syntax

[VB]
```
Value=objActiveBF.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 = ActiveBF1.GetCodecList
For i = 0 To UBound(CodecLst)
Combo1.AddItem (CodecLst(i))
Next
Combo1.ListIndex = 0
```

Remarks

Note that in C/C++ applications it is required to call SafeArrayDestroy() to delete the SAFEARRAY returned by **GetCodecList**.

3.2.24 GetComponentData

Description

Returns the two-dimensional array of pixel values in the specified color component of the current frame.

Syntax

[VB]
```
Value=objActiveBF.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 (0 - Red, 1 - Green, 2 - Blue)
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
ActiveBF1.Grab
DataArray=ActiveBF1.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- and 32-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 scaled to the dynamic range defined by the Format property.

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.

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, subract them from 65535 (see the example above).

3.2.25 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=objActiveBF.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 (0 - Red, 1 - Green, 2 - Blue)
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.

```
ActiveBF1.Grab
Line=ActiveBF1.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- and 32-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 scaled to the dynamic range defined by the Format property.

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.

3.2.26 GetDIB

Description

Returns the handler to a Device Independent Bitmap.

Syntax

[VB]
```
Value=objActiveBF.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
*pD*ib
Pointer to the handler to *ActiveBF's* display DIB

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This C example demonstrates how to retrieve and display *ActiveBF's* DIB

```
BITMAPINFO *pDIB;
pActiveBF->GetDIB((ULONG*)&pDIB);
RECT rect; long sx,sy;
GetWindowRect(hWnd,&rect);
pActiveBF->GetWidth(&sx);
pActiveBF->GetHeight(&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 *ActiveBF's* live display. *ActiveBF* 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 GetImagePointer methods.

3.2.27 GetFPS

Description

Returns the acquired (displayed) frame rate in frames per second.

Syntax

[VB]
```
Value=objActiveBF.GetFPS
```

[C/C++]
```
HRESULT GetFPSAcquired(float* pValue);
```

Data Types [VB]

Return value: Single

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 acquired frame rate.

```
Private Sub ActiveBF1_FrameAcquired()
Label1.Caption = ActiveBF1.GetFPS
End Sub
```

Remarks

This method returns the frame rate based on the arrival of the video frames into the application buffer. It is equal to the frequency at which the FrameAcquired event is called in your application. 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. If the CPU spends more time on processing each frame than the original interval between the frames, the acquired frame rate will be lower than the camera fps.

3.2.28 GetHistogram

Description

Returns the histogram of the current image frame over a selected ROI .

Syntax

[VB]
```
Value=objActiveBF.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 ActiveBF1_FrameAcquired()
ActiveBF1.SetROI 100,100,500, 400
HistArray=ActiveBF1.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.

3.2.29 GetImageData

Description

Returns the two-dimensional array of pixel values in the currently acquired frame.

Syntax

[VB]
```
Value=objActiveBF.GetImageData
```

[C/C++]
```
HRESULT GetImageData( VARIANT* pArray );
```

Data Types [VB]

Y: Integer
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 grabs a frame, retrieves its content and displays the value of the specified element of the array.

```
ActiveBF1.Grab
DataArray=ActiveBF1.GetImageData
x=16
y=32
MsgBox DataArray(x,y)
```

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 *ActiveBF's* image frame in VB.NET and C# use GetImagePointer.

The type of data and dimensions of the array returned by **GetImageData** depends on the Format of the video, its horizontal width SizeX and the number of lines acquired, as specified in the following table:

Format	Data type	Dimensions
8-bit gray	Byte	0 to SizeX -1, 0 to Lines - 1
10-, 12-, 14-, 16-bit gray	Integer (word)	0 to SizeX -1, 0 to Lines - 1
24-bit RGB	Byte	0 to SizeX * 3 - 1, 0 to Lines - 1
32-bit RGB	Byte	0 to SizeX * 4 - 1, 0 to Lines - 1
30-, 36-, 42-, 48-bit RGB	Integer (word)	0 to SizeX * 3 - 1, 0 to Lines - 1

If a line-scan camera is used, the number of image lines acquired in the StartStop mode will differ from the vertical frame size specified by SizeY. Therefore, the vertical dimension of the data array returned by **GetImageData** can vary for each frame depending on the timing of the trigger signal.

If the video is of the high-bit depth, the original pixel values will be scaled to the dynamic range defined by Format.

3.2.30 GetImagePointer

Description

Returns the pointer to the pixel at the specified coordinates of the current frame.

Syntax

[VB]
```
Value=objActiveBF.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 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 CActiveBF is used to access the *ActiveBF* control:

```
BYTE line[4096];
int iWidth=m_ActiveBF.GetSizeX;
m_ActiveBF.Grab()
VARIANT v=m_ActiveBF.GetImagePointer(0,32);
BYTE* ptr=ActiveBF1.GetImagePointer(0,32);
memcpy(&line,v.pcVal,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 axActiveBF1_FrameAcquired(object sender,
AxACTIVEBFLib._IActiveBFEvents_FrameAcquiredEvent e)
{
int sx=axActiveBF1.SizeX;
object obj=axActiveBF1.GetImagePointer(0, 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:

Format	Line width in bytes
8-bit gray	SizeX
10-, 12-, 14-, 16-bit gray	SizeX * 2
24-bit RGB	SizeX * 3
32-bit RGB	SizeX * 4
30-, 36-, 42-, 48-bit RGB	SizeX * 6

The values of the x and y coordinates must not exceed the width and height of the video frame, or the error will occur.

3.2.31 GetImageStat

Description

Returns the array containing statistical data of the current image frame over a selected ROI.

Syntax

[VB]
```
Value=objActiveBF.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)

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 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 ActiveBF1_FrameAcquired()
ActiveBF1.SetROI 100,100,500, 400
StatArray=ActiveBF1.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.

3.2.32 GetImageWindow

Description

Returns the two-dimensional array of pixel values corresponding to the selected window in the currently acquired frame.

Syntax

[VB]
```
Value=objActiveBF.GetImageWindow (X, Y, Width, Height)
```

[C/C++]
```
HRESULT GetImageWindow( short X, short Y, short Width, short Height,
VARIANT* pArray );
```

Data Types [VB]

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()
ActiveBF1.Display = False
ActiveBF1.Acquire = True
End Sub

Private Sub ActiveBF1_FrameAcquired(ByVal Lines As Integer)
xc = ActiveBF1.SizeX / 2
yc = ActiveBF1.SizeY / 2
w = ActiveBF1.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
ActiveBF1.SetImageWindow xc - 70, yc - 50, w
ActiveBF1.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. 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:

Format	**Data type**	**Dimensions**
8-bit gray	Byte	0 to SizeX -1, 0 to Lines - 1
10-, 12-, 14-, 16-bit gray	Integer (word)	0 to SizeX -1, 0 to Lines - 1
24-bit RGB	Byte	0 to SizeX * 3 - 1, 0 to Lines - 1
32-bit RGB	Byte	0 to SizeX * 4 - 1, 0 to Lines - 1
30-, 36-, 42-, 48-bit RGB	Integer (word)	0 to SizeX * 3 - 1, 0 to Lines - 1

If the dimensions of the window are too large to accomodate the frame size, they will be clipped to the frame boundaries. If the video is of the high-bit depth, the original pixel values will be scaled to the dynamic range defined by Format.

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, subract 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 *ActiveBF*.

3.2.33 GetInputBit

Description

Reads the state of the specified general purpose input bit (GPIN) on the board.

Syntax

[VB]
```
Value=objActiveBF.GetInputBit( Bit )
```

[C/C++]
```
HRESULT GetInputBit( short Bit, bool *pValue );
```

Data Types [VB]

Bit: Integer
Return value: Boolean

Parameters [C/C++]

Bit [in]
The zero-based index of the input bit
pValue [out,retval]
Pointer to the bit's value

Return Values

S_OK
Success
E_INVALIDARG
Invalid input argument.

Example

This VB example displays the state of the specified input bit:

```
MsgBox ActiveBF1.GetInputBit(1)
```

Remarks

The valid values of the input bit index are 0 for RoadRunner and R3 Diff boards, 0-2 for R3 CL, 0-5 for Raven and 0-6 for R64.

3.2.34 GetLevels

Description

Returns the array containing the current levels for the Window/Level operation.

Syntax

[VB]
```
value = objActiveBF.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=ActiveBF1.GetLevels
MsgBox "Min: ", Levels[0], "Max: ", Levels[1]
```

Remarks

For more information on the Window/Level parameters see SetLevels.

3.2.35 GetLine

Description

Returns the array of pixel values at the specified horizontal line of the currently acquired frame.

Syntax

[VB]
```
Value=objActiveBF.GetLine( Y )
```

[C/C++]
```
HRESULT GetLine( 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 32th row of pixels and displays the value of 10th pixel in the row.

```
ActiveBF1.Grab
Line=ActiveBF1.GetLine(32)
MsgBox Line(10)
```

Remarks

The type of data and dimension of the array returned by **GetLine** depends on the Format property as specified in the following table:

Format	Data type	Dimension
8-bit gray	Byte	0 to SizeX - 1
10-, 12-, 14-, 16-bit gray	Integer (word)	0 to SizeX - 1
24-bit RGB	Byte	0 to SizeX * 3 - 1
32-bit RGB	Byte	0 to SizeX * 4 - 1
30-, 36-, 42-, 48-bit RGB	Integer (word)	0 to SizeX * 3 - 1

If the video is of the high-bit depth, the original pixel values will be scaled to the dynamic range defined by Format.

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, subract them from 65535 (see example above).

3.2.36 GetLUT

Description

Returns the array containing the current lookup table.

Syntax

[VB]
```
value = objActiveBF.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=ActiveBF1.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

3.2.37 GetPicture

Description

Returns a Picture object created from the currently acquired frame.

Syntax

[VB]
```
Value=objActiveBF.GetPicture()
```

[C/C++]
```
HRESULT GetPicture(IPictureDisp* *pValue);
```

Data Types [VB]

Return value: Picture

Parameters [C/C++]

pValue
Pointer to the IPictureDisp interface object

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example uses the FrameAcquired event to display live video in a PictureBox:

```
Private Sub ActiveBF1_FrameAcquired(ByVal Lines As Integer)
  Picture1.Picture=ActiveBF1.GetPicture
End Sub
```

This VB.NET example shows how to display a live video in a PictureBox:

```
Private Sub AxActiveBF1_FrameAcquired(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles AxActiveBF1.FrameAcquired
 If Not (PictureBox1.Image Is Nothing) Then PictureBox1.Image.Dispose()
 PictureBox1.Image = Bitmap.FromHbitmap(AxActiveBF1.GetPicture().Handle)
End Sub
```

This C# statement shows how to display an image in a PictureBox:

```
 pictureBox1.Image =
Bitmap.FromHbitmap((System.IntPtr)axActiveBF1.GetPicture().Handle);
```

Remarks

The **GetPicture** method provides the most convinient graphic interface to *ActiveBF* 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 GetImagePointer 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.

3.2.38 GetPixel

Description

Returns the pixel value at the specified coordinates.

Syntax

[VB]
```
Value=objActiveBF.GetPixel( X, Y )
```

[C/C++]
```
HRESULT GetPixel( short X, short Y, short* pValue );
```

Data Types [VB]

X: Integer
Y: Integer
Return value: Integer

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.

```
ActiveBF1.Grab
value=ActiveBF1.GetPixel(64,32)
MsgBox value
```

Remarks

The value returned by **GetPixel** depends on the format of the video acquired. For 8-bit grayscale video the method will retrieve the data from the internal image memory. If the video is of the high-bit depth, the pixel value will be scaled to the dynamic range defined by the Format property. 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.

3.2.39 GetRawData

Description

Returns the two-dimensional array of values in the currently acquired data buffer or pointer to this buffer.

Syntax

[VB]
```
Value=objActiveGige.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
ActiveGige1.Grab
RawData=ActiveGige1.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). **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 poitner 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).

3.2.40 GetRGBPixel

Description

Returns the array of RGB values at the specified coordinates.

Syntax

[VB]
```
Value=objActiveBF.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 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.

```
ActiveBF1.Grab
RGB=ActiveBF1.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. If the video is of the high-bit depth, the pixel values will be scaled to the dynamic range defined by the Format property. 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.

3.2.41 GetROI

Description

Returns the array containing the current ROI settings.

Syntax

[VB]
```
value = objActiveBF.GetROI
```

[C/C++]
```
HRESULT SetROI( 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=ActiveBF1.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.

3.2.42 GetSequenceFrameCount

Description

Returns the current number of frames in the memory sequence.

Syntax

[VB]
```
Value=objActiveBF.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=ActiveBF1.GetSequenceFrameCount()
MsgBox value
```

Remarks

For more information on the sequence capture see StartSequenceCapture.

3.2.43 GetSequencePicture

Description

Returns a Picture object representing a selected frame in the memory sequence.

Syntax

[VB]
```
Value=objActiveBF.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
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=ActiveBF1.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.

3.2.44 GetSequencePixel

Description

Returns the pixel value at the specified coordinates.

Syntax

[VB]
```
Value=objActiveBF.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.

```
ActiveBF1.Grab
value=ActiveBF1.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.

3.2.45 GetSequencePointer

Description

Returns the pointer to the pixel at the specified coordinates of the selected frame in the memory sequence.

Syntax

[VB]
```
Value=objActiveBF.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 CActiveBF is used to access the *ActiveBF* control:

```
BYTE line[4096];
int iWidth=m_ActiveBF.GetSizeX;
int iHeight=m_ActiveBF.GetSizeY;
```

```
VARIANT v=m_ActiveBF.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 *ActiveBF* 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.

3.2.46 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=objActiveBF.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=ActiveBF1.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).

3.2.47 GetSequenceTimestamp

Description

Returns the timestamp of the selected frame in the memory sequence, in seconds.

Syntax

[VB]
```
Value=objActiveBF.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 ActiveBF1_FrameLoaded()
Label1.Caption = ActiveBF1.GetSequenceTimestamp
End Sub
```

Remarks

If your GigE 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.

3.2.48 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=objActiveBF.GetImageWindow (Frame, X, Y, Width, Height)
```

[C/C++]
```
HRESULT GetImageWindow( 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 ActiveBF1.GetSequenceFrameCount-1
sx=ActiveBF1.SizeX
sy=ActiveBF1.SizeY
a = ActiveBF1.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**.

3.2.49 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=objActiveBF.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.

```
ActiveBF1.OpenVideo "c:\\video1.avi"
MsgBox ActiveBF1.GetVideoFPS
```

3.2.50 GetVideoFrameCount

Description

Returns the number of frames in the currently open video file.

Syntax

[VB]
```
Value=objActiveBF.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.

```
ActiveBF1.OpenVideo "c:\\sequence.tif"
value=ActiveBF1.GetVideoFrameCount()
MsgBox value
```

3.2.51 GetVideoPosition

Description

Returns the zero-based position of the current frame in the open video file or sequence.

Syntax

[VB]
```
Value=objActiveBF.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 ActiveBF1_FrameLoaded()
frameindex = ActiveBF1.GetVideoPosition + 1
LabelCount.Caption = frameindex
```

Remarks

When the video file or sequnce has just been opened, the video position is reset to zero.

3.2.52 GetVideoVolume

Description

Returns the volume level of the currently open AVI file.

Syntax

[VB]
```
Value=objActiveBF.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.

```
ActiveBF.OpenVideo "C:\\video1.avi", True
MsgBox ActiveBF1.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.

3.2.53 Grab

Description

Grabs a single frame into the internal memory.

Syntax

[VB]
```
objActiveBF.Grab
```

[C/C++]
```
HRESULT Grab();
```

Parameters

None

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example grabs a frame and saves it as a tiff file

```
ActiveBF1.Grab
ActiveBF1.SaveImage("image1.tif",0)
```

Remarks

If the Asynch property is set to FALSE (synchronous acquisition), the **Grab** method will wait for the current frame to be acquired before returning. If the asynchronous mode is selected, the method will initiate the acquisition and return immediately. This will allow your application to perform other tasks while the frame is being acquired. When the acquisition of the current frame is complete, the FrameAcquired event will be raised.

If the continuous acquisition mode is selected (the Acquire property is set to TRUE), the **Grab** method will not affect the acquisition process, however you can use it to delay your application until the current frame is completed.

3.2.54 LoadImage

Description

Loads the image from the specified file into the internal frame buffer and displays it in *ActiveBF* window. The method supports BMP, TIFF and JPEG formats.

Syntax

[VB]
```
objActiveBF.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.

```
ActiveBF1.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.

3.2.55 LoadSequence

Description

Loads a fragment of the specified AVI or TIFF file into the memory sequence.

Syntax

[VB]
```
objActiveBF.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.

```
ActiveBF1.LoadSequence "video1.tif"
ActiveBF1.OpenVideo "ram"
ActiveBF1.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.

3.2.56 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]
```
objActiveBF.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()
ActiveBF1.OpenVideo "ram"
ActiveBF1.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 *ActiveBF* object. To play back several video sources, use several *ActiveBF* objects.

3.2.57 OverlayClear

Description

Clears graphics and text from the overlay.

Syntax

[VB]
```
objActiveBF.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
ActiveBF1.Acquire = True
ActiveBF1.OverlayColor = RGB(255, 0, 0)
ActiveBF1.Overlay = True
End Sub


Dim x As Integer
Private Sub ActiveBF1_FrameAcquired(ByVal Lines As Integer)
ActiveBF1.OverlayClear
ActiveBF1.OverlayRectangle x, 10, x + 50, 80, 3
x = x + 2
If x = ActiveBF1.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.

3.2.58 OverlayEllipse

Description

Draws an empty or filled ellipse in the overlay.

Syntax

[VB]
```
objActiveBF.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, relative to the image origin.
EndX [in], *EndY* [in]
Pixel coordinates of the bottom right corner of the ellipse, 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:

```
ActiveBF1.Acquire = True
ActiveBF1.OverlayEllipse 100,100,200,200,0
ActiveBF1.OverlayColor=RGB(255,0,0)
ActiveBF1.Overlay=  True
```

Remarks

To draw a filled ellipse, use *Width=0.* Also see OverlayColor, OverlayClear.

3.2.59 OverlayLine

Description

Draws a line in the overlay.

Syntax

[VB]
```
objActiveBF.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:

```
ActiveBF1.Acquire = True
ActiveBF1.OverlayLine 100,100,200,200,3
ActiveBF1.OverlayColor=RGB(255,0,0)
ActiveBF1.Overlay=  True
```

Remarks

To draw multiple lines, call this method several times. Also see OverlayColor, OverlayClear.

3.2.60 OverlayPixel

Description

Draws a pixel in the overlay.

Syntax

[VB]
```
objActiveBF.OverlayPixel X, Y
```

[C/C++]
```
HRESULT OverlayPixel( short X, short Y, bstr Text );
```

Data Types [VB]

X, Y : Integer
Text : String

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:

```
ActiveBF1.Acquire = True
ActiveBF1.OverlayPixel 100,100
ActiveBF1.OverlayPixel 100,200
ActiveBF1.OverlayPixel 200,100
ActiveBF1.OverlayPixel 200,200
ActiveBF1.OverlayColor=RGB(255,0,0)
ActiveBF1.Overlay=  True
```

Remarks

To draw custom shapes, call this method multiple times. Also see OverlayColor, OverlayClear.

3.2.61 OverlayRectangle

Description

Draws an empty or filled rectangle in the overlay.

Syntax

[VB]
```
objActiveBF.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:

```
ActiveBF1.Acquire = True
ActiveBF1.OverlayRectangle 100,100,200,200,0
ActiveBF1.OverlayColor=RGB(255,0,0)
ActiveBF1.Overlay=  True
```

Remarks

To draw a filled rectangle, use *Width=0.* To draw multiple rectangles, call this method several times.
Also see OverlayColor, OverlayClear.

3.2.62 OverlayText

Description

Draws a string of text in the overlay.

Syntax

[VB]
```
objActiveBF.OverlayText X, Y, Text
```

[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.
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
ActiveBF1.OverlayFont = Font
ActiveBF1.OverlayText 10, 100, "ActiveBF rules!"
ActiveBF1.OverlayColor = RGB(255, 0, 0)
ActiveBF1.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.

3.2.63 PlayVideo

Description

Plays back the currently open video file (AVI or TIF) or memory sequence in the *ActiveBF* window.

Syntax

[VB]
```
objActiveBF.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 ommitted, 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()
ActiveBF1.OpenVideo "C:\\video1.avi"
ActiveBF1.PlayVideo 0, 50
End Sub

Dim frameplayed As Integer

Private Sub ActiveBF1_FrameLoaded()
frameplayed = frameplayed + 1
LabelCount.Caption = frameplayed
```

```
End Sub

Private Sub ActiveBF1_PlayCompleted (ByVal Frames As Long)
ActiveBF1.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 *ActiveBF* buffer. The pixels of the currently loaded frame can be accessed via *ActiveBF* 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.

3.2.64 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]
```
objActiveBF.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
ActiveBF1.SaveBkg (1)
End Sub

Private Sub SaveBright_Click
ActiveBF1.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 filed 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 ActiveBF 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.

3.2.65 SaveImage

Description

Saves the current frame buffer in the specified image file. The method supports BMP, TIFF and JPEG formats with a selectable compression.

Syntax

[VB]
```
objActiveBF.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 grabs a frame and saves it in a JPEG file with the specified compression quality.

```
ActiveBF1.Grab
ActiveBF1.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 "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.

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.

If the *Compression* argument is omitted, BMP and TIFF files will be saved with no compression while JPEG files will be saved with the quality of 75.

3.2.66 SaveSequence

Description

Saves the current memory sequence or its fragment in the specified AVI or TIFF file.

Syntax

[VB]
```
objActiveBF.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()
ActiveBF1.StartSequenceCapture  1000
End Sub
```

```
Private Sub ActiveBF1_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.

3.2.67 SerialClose

Description

Closes the currently open Camera Link serial device.

Syntax

[VB]
```
Value=objActiveBF.SerialClose
```

[C/C++]
```
HRESULT SerialClose();
```

Parameters

None

Return Values

S_OK
  Success
E_FAIL
  Failure.

Example

The following VB example opens the serial port on the board, writes a command to it and closes the port.

```
ActiveBF1.SerialOpen 1,"9600,N,8,1"
ActiveBF1.SerialWrite "ssf 10000\n",1000
ActiveBF1.SerialClose
```

Remarks

To open and configure a serial port, use SerialOpen

3.2.68 SerialConnect

Description

Selects the number of the serial device on the Camera Link board

Syntax

[VB]
```
Value=objActiveBF.SerialConnect( Mode )
```

[C/C++]
```
HRESULT SerialRead( long Mode );
```

Data Types [VB]

Mode: Integer

Parameters [C/C++]

Mode [in]
The connection mode.

Return Values

S_OK
Success
E_FAIL
Failure.
E_INVALIDARG
Invalid input argument.

Example

The following VB example switches the serial interface to the external serial device and communicates with a camera through the PC's serial port:

```
ActiveBF1.SerialConnect(2);
ActiveBF1.SerialOpen 0,"9600,N,8,1"
```

Remarks

Use this function to control the connection of serial ports on Camera Link connectors of the currently selected framegrabber. For dual interface Camera Link boards (e.g. BitFlow R64-1-3) this provides switching the on-board serial device (UART) between two cameras. You can also configure the board to bypass the on-board UART and reroute the serial port signals on a camera-link connector to an external serial device (usually the host PC's communication port).

The connection mode must be one of the following values:
0 – the on-board UART connected to the serial port of the main Camera Link connector.
1 – the on-board UART connected to the serial port of the auxiliary Camera Link connector.
2 – serial port the main Camera Link connector connected to the external port through the I/O connector.

3 –serial port the auxiliary Camera Link connector connected to the external port through the I/O connector.

3.2.69 SerialOpen

Description

Opens the serial device on the Camera Link board and configures its settings.

Syntax

[VB]
```
objActiveBF.SerialOpen Com, Settings
```

[C/C++]
```
HRESULT SerialOpen( long Com, bstr* Setting );
```

Data Types [VB]

Com: Integer
Return value: Integer

Parameters [C/C++]

Com [in]
Zero-based index of the serial device. Must be not higher than n-1, where n is a number of Camera Link serial devices on the board.
Settings [in]
String specifying the serial device's baud rate, parity, data bit, and stop bit attributes. It is composed of four settings and has the following format: "BBBB,P,D,S", where BBBB is the baud rate, P is the parity, D is the number of data bits, and S is the number of stop bits. If this parameter is an empty string or omitted, the following settings will be used: "9600, N, 8, 1".

Return Values

S_OK
Success
E_FAIL
Failure.
E_INVALIDARG
Invalid input arguments.

Example

The following VB example opens the serial port on the board, writes a command to it and closes the port.

```
ActiveBF1.SerialOpen 0,"9600,N,8,1"
ActiveBF1.SerialWrite "ssf 10000\n",1000
ActiveBF1.SerialClose
```

Remarks

The valid baud rates are: 110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 56000,

57600, 115200, 128000, 256000.
The valid parity values are: E (Even), M (Mark), N (None), O (Odd), S (Space).
The valid data bit values are: 4, 5, 6, 7, 8
The valid stop bit values are: 1, 1.5, 2

3.2.70 SerialRead

Description

Reads the string of characters from the currently open Camera Link serial device. The function will wait for the specified number of milliseconds for data in the receive buffer. In the case that data is in the receive buffer, the buffer will be read until empty.

Syntax

[VB]
```
Value=objActiveBF.SerialRead [ Timeout ]
```

[C/C++]
```
HRESULT SerialRead( long Timeout, bstr* pInput );
```

Data Types [VB]

Timeout: Integer (optional)
Return value: String

Parameters [C/C++]

Timeout [in]
The maximum time to wait for data in the receive buffer, in milliseconds. If this parameter is zero or omitted, the timeout will never occur
pInput [out,retval]
String containing the data from the receive buffer

Return Values

S_OK
Success
E_FAIL
Failure.
E_INVALIDARG
Invalid input argument.

Example

The following VB example opens the serial port on the board, writes a command to it and displays a response from the camera:

```
Dim Str As String
ActiveBF1.SerialOpen 0,"9600,N,8,1"
ActiveBF1.SerialWrite "ssf 10000\n",1000
Str=ActiveBF1.SerialRead(200)
MsgBox Str
```

Remarks

To open and configure a serial port, use SerialOpen

3.2.71 SerialWrite

Description

Writes the string of characters to the currently open Camera Link serial device.

Syntax

[VB]
```
objActiveBF.SerialWrite Output [, Timeout ]
```

[C/C++]
```
HRESULT SerialWrite( bstr strOutput, long Timeout );
```

Data Types [VB]

Output: String
Timeout: Integer (optional)

Parameters [C/C++]

pOutput [in]
The output string
Timeout [in]
The timeout in millisecond. The function will try to write the data to the serial device for the specified period of time. If the data could not be written within that time, the timeout error flag will be raised. If this parameter is zero or omitted, the timeout will never occur

Return Values

S_OK
Success
E_FAIL
Failure.
E_INVALIDARG
Invalid input argument.

Example

The following VB example opens the serial port on the board, writes a command to it and closes the port.

```
ActiveBF1.SerialOpen 0,"9600,N,8,1"
ActiveBF1.SerialWrite "ssf 10000\n",1000
ActiveBF1.SerialClose
```

Remarks

To open and configure a serial port, use SerialOpen

3.2.72 SetAudioLevel

Description

Sets the recording level of the currently selected audio device.

Syntax

[VB]
```
objActiveBF.SetAudioLevel Value
```

[C/C++]
```
HRESULT SetAudioLevel(short Value);
```

Data Types [VB]

Value: Short

Parameters [C/C++]

Value [out,retval]
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()
ActiveBF1.SetAudioSource 0
HScroll1.Min=0
HScroll1.Max=100
HScroll.Value=ActiveBF1.GetAudioLevel
End Sub

Private Sub Capture_Click
ActiveBF1.StartCapture "c:\\capture_with_sound.avi"
End Sub

Private Sub HScroll1_Scroll()
ActiveBF1.SetAudioLevel HScroll1.Value
End Sub
```

Remarks

In order for this method to work, the audio recording device must be selected with SetAudioSource.

3.2.73 SetAudioSource

Description

Sets the index of the audio recording device to be used during the AVI capture.

Syntax

[VB]
```
objActiveBF.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 = ActiveBF1.GetAudioList
For i = 0 To UBound(AudioLst)
Combo1.AddItem (AudioLst(i))
Next
Combo1.ListIndex = 0
ActiveBF1.SetAudioSource 0

Private Sub Combo1_Click()
ActiveBF1.SetAudioSource Combo1.ListIndex
End Sub

Private Sub Capture_Click
ActiveBF1.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 ActiveBF.

To initiate the AVI recording, use StartCapture.

3.2.74 SetCodec

Description

Sets the name of the codec (video compressor) to be used during the AVI capture.

Syntax

[VB]
```
objActiveBF.SetCodec Name [, Quality, Datarate, Keyframe ])
```

[C/C++]
```
HRESULT SetCodec(bstr Name [, long Quality = 0, long Datarate = 0, long
Keyframe = 0 ]);
```

Data Types [VB]

Name: String
Quality: Integer
Datarate: Integer
Keyframe: Integer

Parameters [C/C++]

Name [in]
String specifying the name of the feature
Quality [out, retval]
Numerical value of the quality parameter used by the codec, if supported.
Datarate [out, retval]
Datarate in kb/sec used by the codec, if supported.
Keyframe [out, retval]
The amount of frames after which a key frame is recorded, if supported.

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()
ActiveBF1.SetCodec "DivX 5.0.2 Codec"
ActiveBF1.Acquire=True
End Sub

Private Sub StartButton_Click()
ActiveBF1.StartCapture "c:\mycapture.avi"
End Sub
```

```
Private Sub StopButton_Click()
ActiveBF1.StopCapture
End Sub
```

The following VB example demonstrates how to set up a codec by its index in the system:

```
CodecList=ActiveBF1.GetCodecList
CodecName=CodecList(9)
ActiveBF1.SetCodec CodecName
```

Remarks

This method allows you to set only the basic compression parameters which may not be supported by certain codecs. To adjust the internal parameters of a codec, use ShowCodecDialog.

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.

3.2.75 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]
```
objActiveBF.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.
fR [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()
ActiveBF1.Acquire=True
HScroll1.Min = 1
HScroll1.Max = 10
HScroll2.Min = 1
HScroll2.Max = 10
HScroll3.Min = 1
HScroll3.Max = 10
ActiveBF1.LUTMode=True
```

```
End Sub

Private Sub HScroll1_Scroll()
L1 = HScroll1.Value
L2 = HScroll2.Value
L3 = HScroll3.Value
ActiveBF1.SetGains L1, L2, L3
End Sub

Private Sub HScroll2_Scroll()
L1 = HScroll1.Value
L2 = HScroll2.Value
L3 = HScroll3.Value
ActiveBF1.SetGains L1, L2, L3
End Sub
```

Remarks

The gain adjustment will be applied to the video only when LUTMode is enabled. Otherwise the original pixel values will remain intact.

3.2.76 SetImageWindow

Description

Copies pixel values from the two-dimensional array into the selected window of the current frame.

Syntax

[VB]
```
objActiveBF.SetImageWindow X, Y
```

[C/C++]
```
HRESULT SetImageWindow( short X, short Y, VARIANT* pArray );
```

Data Types [VB]

Return value: 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.
pArray [out,retval]
Pointer to the SAFEARRAY 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()
ActiveBF1.Display = False
ActiveBF1.Acquire = True
End Sub

Private Sub ActiveBF1_FrameAcquired(ByVal Lines As Integer)
xc = ActiveBF1.SizeX / 2
yc = ActiveBF1.SizeY / 2
w = ActiveBF1.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
ActiveBF1.SetImageWindow xc - 70, yc - 50, w
ActiveBF1.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:

Format	**Data type**	**Dimensions**
8-bit gray	Byte	0 to SizeX -1, 0 to Lines - 1
10-, 12-, 14-, 16-bit gray	Integer (word)	0 to SizeX -1, 0 to Lines - 1
24-bit RGB	Byte	0 to SizeX * 3 - 1, 0 to Lines - 1
32-bit RGB	Byte	0 to SizeX * 4 - 1, 0 to Lines - 1
30-, 36-, 42-, 48-bit RGB	Integer (word)	0 to SizeX * 3 - 1, 0 to Lines - 1

where Width is the intended horizontal size of the window in in pixels. If the dimensions of the window are too large to accomodate 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.

3.2.77 SetLevels

Description

Sets the levels for the Window/Level operation. If the LUTMode 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]
```
objActiveBF.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.

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()
ActiveBF1.Acquire=True
HScroll1.Min = 0
HScroll1.Max = 100
HScroll2.Min = 0
HScroll2.Max = 100
ActiveBF1.LUTMode=True
End Sub

Private Sub HScroll1_Scroll()
L1 = HScroll1.Value
L2 = HScroll2.Value
ActiveBF1.SetLevels L1, L2
End Sub

Private Sub HScroll2_Scroll()
L1 = HScroll1.Value
L2 = HScroll2.Value
ActiveBF1.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 GigE Vision formats

Pixel Format	**Maximum pixel value**
Mono8	255
Mono10, Mono10Packed	1023
Mono12, Mono12Packed	4095
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.

3.2.78 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]
```
objActiveBF.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
ActiveBF1.SetLUT A, 3
```

Remarks

The LUT processing in ActiveBF 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.

3.2.79 SetOutputBit

Description

Sets the state of the specified general purpose output bit (GPOUT) on the board.

Syntax

[VB]
```
objActiveBF.SetOutputBit  Bit, Value
```

[C/C++]
```
HRESULT SetOutputBit( short Bit, bool Value );
```

Data Types [VB]

Bit: Integer
Value: Boolean

Parameters [C/C++]

Bit [in]
The zero-based index of the output bit
Value [in]
Bit value to be set

Return Values

S_OK
Success
E_INVALIDARG
Invalid input argument.

Example

This VB example sets the output bit #2 to 1 (TRUE):

```
ActiveBF1.SetOutputBit 2, True
```

Remarks

The valid values of the output bit index are 0-2 for RoadRunner and R3 Diff boards, 0-4 for R3 CL, 0-5 for Raven and 0-6 for R64.

3.2.80 SetROI

Description

Sets the rectangular region of interest and luminance range for the histogram and image statistics calculations.

Syntax

[VB]
```
objActiveBF.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()
ActiveBF1.Palette = 8
HScroll1.Min = 0
HScroll1.Max = 255
X1 = 0
Y1 = 0
X2 = ActiveBF1.SizeX
```

```
Y2 = ActiveBF1.SizeY
L1 = 0
L2 = 255
End Sub


Private Sub HScrollThreshold1_Scroll()
L1 = HScrollThreshold1.Value
LabelThreshold1.Caption = L1
ActiveBF1.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.

3.2.81 SetVideoFPS

Description

Sets the playback frame rate of the currently open video file or memory sequence.

Syntax

[VB]
```
objActiveBF.SetVideoFPS Value
```

[C/C++]
```
HRESULT SetVideoFPS(float Value);
```

Data Types [VB]

Value: Single

Parameters [C/C++]

Value [out,retval]
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.

```
ActiveBF1.OpenVideo "c:\\video1.avi"
fps=ActiveBF1.GetVideoFPS * 2
ActiveBF1.SetVideoFPS fps
ActiveBF1.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.

3.2.82 SetVideoPosition

Description

Seeks the specified frame in the currently open video file or memory sequence, extracts it and displays in the *ActiveBF* window.

Syntax

[VB]
```
objActiveBF.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()
ActiveBF1.OpenVideo "C:\\video1.avi"
HScroll1.Min=0
HScroll1.Max=ActiveBF1.GetVideoFrameCount - 1
End Sub

Private Sub HScroll1_Scroll()
frameposition = HScroll1.Value
ActiveBF1.SetVideoPosition frameposition
End Sub
```

Remarks

This method extracts the specified frame from the video file or memory sequence and loads it into the internal *ActiveBF* buffer. The FrameLoaded event will be fired, when **SetVideoPosition** is called successfully. The pixels of the loaded frame can be accessed via *ActiveBF* 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.

3.2.83 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]
```
objActiveBF.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 *ActiveBF* 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 *ActiveBF* objects.

```
Private Sub Form_Load()
ActiveBF1.OpenVideo "C:\\video1.avi"
ActiveBF2.OpenVideo "C:\\video2.avi"
ActiveBF1.SetVideoSync 1
ActiveBF2.SetVideoSync 1
ActiveBF1.PlayVideo
ActiveBF2.PlayVideo
fps=ActiveBF1.GetVideoFPS
Timer1.Interval=1000/fps
End Sub

Private Sub Timer1_Timer()
ActiveBF1.TriggerVideo
ActiveBF2.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.

3.2.84 SetVideoVolume

Description

Sets the volume level of the currently open AVI file.

Syntax

[VB]
```
objActiveBF.SetVideoVolume Value
```

[C/C++]
```
HRESULT SetVideoVolume(short Value);
```

Data Types [VB]

Value: Short

Parameters [C/C++]

Value [out,retval]
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
ActiveBF1.OpenVideo "C:\\video1.avi", True
HScroll.Value=ActiveBF1.GetVideoVolume
ActiveBF1.PlayVideo
End Sub

Private Sub HScroll1_Scroll()
ActiveBF1.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.

3.2.85 ShowAudioDlg

Description

Displays the audio Input dialog for adjusting the mixer properties of the audio source.

Syntax

[VB]
```
objActiveBF.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:

```
ActiveBF1.SetAudioSource 0
ActiveBF1.ShowAudioDlg
```

Remarks

The options available in the audio input dialog will depend on the configuration of your audio devices.

3.2.86 ShowCodecDlg

Description

Displays the configuration dialog for the currently selected codec. The dialog is provided by the codec's manufacturer.

Syntax

[VB]
```
objActiveBF.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:

```
ActiveBF1.SetCodec "DivX 5.0.2 Codec"
ActiveBF1.ShowCodecDlg
```

Remarks

The settings selected in the codec configuration dialog will be remembered as long as the corresponding ActiveBF control remains in the memory.

3.2.87 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]
```
objActiveBF.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()
ActiveBF1.Acquire=True
End Sub

Private Sub CompressionButton_Click()
ActiveBF1.ShowCompressionDlg
End Sub

Private Sub StartButton_Click()
ActiveBF1.StartCapture "c:\mycapture.avi"
End Sub

Private Sub StopButton_Click()
ActiveBF1.StopCapture
End Sub
```

Remarks

The settings selected in the compression dialog are remembered as long as the corresponding *ActiveBF* control remains in the memory.

3.2.88 ShowProperties

Description

Displays *ActiveBF* property pages in runtime mode.

Syntax

[VB]
```
objActiveBF.ShowProperties [ EnableCamList ]
```

[C/C++]
```
HRESULT ShowProperties( bool bEnableCamList );
```

Parameters

None

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()
ActiveBF1.ShowProperties
End Sub
```

3.2.89 StartCapture

Description

Starts time-lapse video capture to the specified AVI file or series of image files (bmp, tif or jpg). Use StopCapture to end video capture.

Syntax

[VB]
objActiveBF.StartCapture File [, Timelapse = 0]

[C/C++]
HRESULT StopCapture(bstr File, float Timelapse);

Data Types [VB]

File : String
Timelapse : Single (optional)

Parameters [C/C++]

File [in]
The string containing the path to the avi file or image file (bmp, tif or jpg).
Timelapse [in]
The interval between consecutive frames in seconds

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()
ActiveBF1.StartCapture "c:\\mycapture.avi"
End Sub

Private Sub StopButton_Click()
ActiveBF1.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)
{
        axActiveBF1.StartCapture("c:\\images\\myframe.tif", 0.5);
}

private void stopbutton_Click(object sender, System.EventArgs e)
{
        axActiveBF1.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, *ActiveBF* 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 and system throughput.

If *Playrate* is not specified, the video will be played back at the same rate it was captured.

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.

3.2.90 StopCapture

Description

Stops video capture to an AVI file or series of images.

Syntax

[VB]
```
objActiveBF.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()
ActiveBF1.StartCapture "c:\mycapture.avi", 0.5
End Sub

Private Sub StopButton_Click()
ActiveBF1.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.

3.2.91 StopVideo

Description

Stops the playback of a video file or memory sequence that has been initiated by PlayVideo.

Syntax

[VB]
```
objActiveBF.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()
ActiveBF1.OpenVideo "C:\\video1.avi"
End Sub

Private Sub Play_Click()
ActiveBF1.PlayVideo -1
End Sub

Private Sub StopButton_Click()
ActiveBF1.StopVideo
End Sub
```

Remarks

When this method is called, the PlayCompleted event will be fired.

3.2.92 StartSequenceCapture

Description

Starts acquiring a sequence of frames into the memory. Use StopSequenceCapture to end the sequence capture process.

Syntax

[VB]
```
objActiveBF.StartSequenceCapture  Frames [, Timelapse = 0 ]
```

[C/C++]
```
HRESULT StartSequenceCapture( bstr File, float Timelapse);
```

Data Types [VB]

Frames : String
Timelapse : Single (optional)

Parameters [C/C++]

Frames [in]
The maximum 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()
ActiveBF1.StartSequenceCapture  -1000
End Sub

Private Sub StopButton_Click()
ActiveBF1.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, ActiveBF 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, *ActiveBF* 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.

3.2.93 StopSequenceCapture

Description

Stops acquiring a sequence of frames into the memory.

Syntax

[VB]
```
objActiveBF.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()
ActiveBF1.StopSequenceCapture
End Sub
```

Remarks

Use this method to explicitely end the sequence capture initiated by StartSequenceCapture.

When the sequence capture stops, the CaptureComplete event will be fired.

3.2.94 StartVideoCapture

Description

Starts time-lapse video capture to the AVI file created by CreateVideo. Use StopVideoCapture to end video capture.

Syntax

[VB]
```
objActiveBF.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 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()
ActiveBF1.CreateVideo  "c:\\mycapture.avi"
ActiveBF1.Acquire = True
End Sub

Private Sub StartButton_Click()
ActiveBF1.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 immediatelly 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, *ActiveBF* 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.

3.2.95 StopVideoCapture

Description

Stops video capture to the AVI file.

Syntax

[VB]
```
objActiveBF.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()
ActiveBF1.CreateVideo  "c:\\mycapture.avi"
ActiveBF1.Acquire = True
End Sub

Private Sub StartButton_Click()
ActiveBF1.StartVideoCapture 0, 0.5
End Sub

Private Sub StopButton_Click()
ActiveBF1.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.

3.2.96 SwitchTrigger

Description

Asserts/deasserts the software trigger signal.

Syntax

[VB]
```
objActiveBF.SwitchTrigger
```

[C/C++]
```
HRESULT SwitchTrigger();
```

Parameters

None

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example sets the continuous acquisition mode and then activates the software trigger to initiate the frame capture:

```
ActiveBF1.Acquire = TRUE
ActiveBF1.SwitchTrigger
```

Remarks

Use **SwitchTrigger** to generate the software trigger signal. Note that this method will only have effect if the Trigger mode is enable. The trigger signal will typically initiate an acquisition of a single frame. If a line-scan Camera is used with the StartStop mode enabled, the **SwitchTrigger** method should be used twice: first time to initiate the frame acquisition and second time to finish it. In this case the size of the frame will depend on the time between two calls.

3.2.97 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]
```
objActiveBF.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 *ActiveBF* objects.

```
Private Sub Form_Load()
ActiveBF1.OpenVideo "C:\\video1.avi"
ActiveBF2.OpenVideo "C:\\video2.avi"
ActiveBF1.SetVideoSync 1
ActiveBF2.SetVideoSync 1
ActiveBF1.PlayVideo
ActiveBF2.PlayVideo
fps=ActiveBF1.GetVideoFPS
Timer1.Interval=1000/fps
End Sub

Private Sub Timer1_Timer()
ActiveBF1.TriggerVideo
ActiveBF2.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.

3.3 Events

The following events are generated by *ActiveBF* control:

FrameAcquired	Called after a frame has been acquired into the internal memory
FrameAcquiredX	Called after a frame has been acquired into the internal memory (multithreaded version)
LineAcquired	Called after a horizontal line has been acquired into the internal memory
Overflow	Called if an overflow occurred during the acquisition
Timeout	Called if the acquisition timeout has expired
MouseDown	Called when the mouse button is pressed inside the control window
MouseUp	Called when the mouse button is released inside the control window
MouseDblClick	Called when the mouse button is double-clicked inside the control window
MouseMove	Called when the mouse button has moved inside the control window
Scroll	Called when the live video display has been scrolled
FormatChanged	Called if the video size or pixel format has changed

3.3.1 CaptureCompleted

Description

This event is fired when the capture into the memory sequence or video file is stopped.

Syntax

[VB]
```
Private Sub objActiveBF_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()
ActiveBF1.StartSequenceCapture 1000
End Sub

Private Sub ActiveBF1_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.

3.3.2 FormatChanged

Description

This event is fired each time the frame size or pixel format of the video changes.

Syntax

[VB]
```
Private Sub objActiveBF_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 ActiveBF1_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 *ActiveBF*. See ShowProperties for more details.

3.3.3 FrameAcquired

Description

This event is fired each time a frame has been acquired into the internal memory. Returns the actual number of lines acquired.

Syntax

[VB]
```
Private Sub objActiveBF_FrameAcquired( ByVal Lines As Integer )
```

[C/C++]
```
HRESULT Fire_FrameAcquired( SHORT Lines );
```

Data Types [VB]

Lines: Integer

Parameters [C/C++]

Lines [in]
The number of lines acquired.

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 ActiveBF1_FrameAcquired(ByVal Lines As Integer)
Label1.Caption = ActiveBF1.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 *Line* parameter is especially useful when the video is being acquired from a line-scan camera in the StartStop mode. In this case the actual number of lines acquired depends on the timing of a trigger signal and differs from the vertical frame size specified by SizeY.

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.

3.3.4 FrameAcquiredX

Description

This event is fired each time a frame has been acquired into the internal memory. Unlike FrameAcquired event it is fired from a processing thread providing a higher efficiency. Might not work in certain containers.

Syntax

[VB]
```
Private Sub objActiveBF_FrameAcquiredX()
```

[C/C++]
```
HRESULT Fire_FrameAcquiredX();
```

Parameters

The *Line* parameter is especially useful when the video is being acquired from a line-scan camera in the StartStop mode. In this case the actual number of lines acquired depends on the timing of a trigger signal and differs from the vertical frame size specified by SizeY.

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:

```
Private Sub AxActiveBF1_FrameAcquired(ByVal sender As System.Object, ByVal e As
AxACTIVEBFLib._IActiveBFEvents_FrameAcquiredEvent)
 Label1.Text = AxActiveBF1.GetPixel(16, 32)
End Sub
```

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 and Delphi use FrameAcquired event instead.

The *Line* parameter is especially useful when the video is being acquired from a line-scan camera in the StartStop mode. In this case the actual number of lines acquired depends on the timing of a trigger signal and differs from the vertical frame size specified by SizeY.

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.

3.3.5 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 objActiveBF_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 ActiveBF1_FrameLoaded((ByVal Frame As Long))
Label1.Caption = Frame
End Sub
```

Remarks

3.3.6 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 objActiveBF_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 ActiveBF1_FrameRecorded(ByVal Frame As Long)
Label1.Caption = Frame
End Sub
```

Remarks

3.3.7 FrameReady

Description

This event is fired each time a decoded frame is submitted for display.

Syntax

[VB]
```
Private Sub objActiveBF_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 CGcamProDlg::OnFrameReadyActiveBF1()
{
   BYTE *ptr=m_ActiveBF.GetImagePointer(0, m_ActiveBF.GetSizeY()-1);
   int nSize=m_ActiveBF.GetSizeX()*m_ActiveBF.GetSizeY();
   int max=255;
   for(int y=0;y<nSize;y++,ptr++)
      *ptr=(*ptr+16)*2;
}
```

Remarks

The **FrameReady** event is fired from *ActiveBF'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

3.3.8 LineAcquired

Description

This event is fired each time a horizontal line of pixels has been acquired into the internal memory. Returns the line number in the frame.

Syntax

[VB]
```
Private Sub objActiveBF_LineAcquired( ByVal Line As Integer )
```

[C/C++]
```
HRESULT Fire_LineAcquired( SHORT Line );
```

Data Types [VB]

Line: Integer

Parameters [C/C++]

Lines [in]
The number of the line acquired, from top to bottom.

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example uses the **LineAcquired** event to display a line counter in real time:

```
Private Sub ActiveBF1_LineAcquired(ByVal Line As Integer)
Label1.Caption = Line
End Sub
```

Remarks

One of the following conditions must be met, before the **LineAcquired** event will be fired:
The LineScan property has been set to TRUE
The Acquire property has been set to TRUE or the Grab method has been called.

Note that using this event is only recommended for line scan cameras set into a slow line rate. Firing and handling this event for fast-rate cameras (>1000 lines/sec) can significantly reduce the performance of your application. Also, when used with regular cameras this event might not be generated per each line, since the DMA process moves data extremely fast.

3.3.9 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 objActiveBF_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 ActiveBF1_MouseDblClick(ByVal X As Integer, ByVal Y As
Integer)
Dim Value As Integer
Value = ActiveBF1.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.

3.3.10 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 objActiveBF_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 ActiveBF1_MouseDown(ByVal X As Integer, ByVal Y As Integer)
Dim Value As Integer
Value = ActiveBF1.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.

3.3.11 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 objActiveBF_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 ActiveBF1_MouseMove(ByVal X As Integer, ByVal Y As Integer)
Dim Value As Integer
Value = ActiveBF1.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.

3.3.12 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 objActiveBF_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 ActiveBF1_MouseUp(ByVal X As Integer, ByVal Y As Integer)
Dim Value As Integer
Value = ActiveBF1.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.

3.3.13 PlayCompleted

Description

This event is fired when the playback of the memory sequence or video file is stopped.

Syntax

[VB]
```
Private Sub objActiveBF_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()
ActiveBF1.OpenVideo "ram"
ActiveBF1.PlayVideo
End Sub
```

```
Private Sub ActiveBF1_PlayCompleted(ByVal Frames As Long)
ActiveBF1.CloseVideo
ActiveBF1.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.

3.3.14 RawFrameAcquired

Description

This event is fired each time a raw frame has arrived from the camera, before it is submitted to *ActiveBF* decoding and processing chain. Can be used in combination with GetRawData to perform a preprocessing on raw images.

Syntax

[VB]
```
Private Sub objActiveBF_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 CGcamProDlg::OnRawFrameAcquiredActiveBF1()
{
   VARIANT v=m_ActiveBF.GetRawData(true);
   int nSize=m_ActiveBF.GetSizeX()*m_ActiveBF.GetSizeY();
   BYTE* ptr=(BYTE*)v.pcVal;
   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 ActiveBF 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.

3.3.15 Overflow

Description

This event is fired each time an overflow occurs during the acquisition of a frame.

Syntax

[VB]
```
Private Sub objActiveBF_Overflow()
```

[C/C++]
```
HRESULT Fire_Overflow();
```

Parameters [C/C++]

None

Return Values

S_OK
  Success
E_FAIL
  Failure.

Example

This VB example uses the **Overflow** event to generate a sound signal:

```
Private Sub ActiveBF1_Overflow()
Beep
End Sub
```

Remarks

The **Overflow** event is raised if a video FIFO overflow occurs during the acquisition of a frame resulting in a frame drop. The most common reasons for this are low PCI bandwidth or CPU overload.

3.3.16 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 objActiveBF_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 ActiveBF1_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.

3.3.17 Timeout

Description

This event is fired each time a timeout occurs during the acquisition of a frame.

Syntax

[VB]
```
Private Sub objActiveBF_Timeout()
```

[C/C++]
```
HRESULT Fire_Timout();
```

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 ActiveBF1_Timeout()
Beep
End Sub
```

Remarks

The **Timeout** event is raised if a time period set by the Timeout property expires 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). Common reasons for a timeout are the absence of a video signal on the board or a missing trigger signal in the Trigger mode.

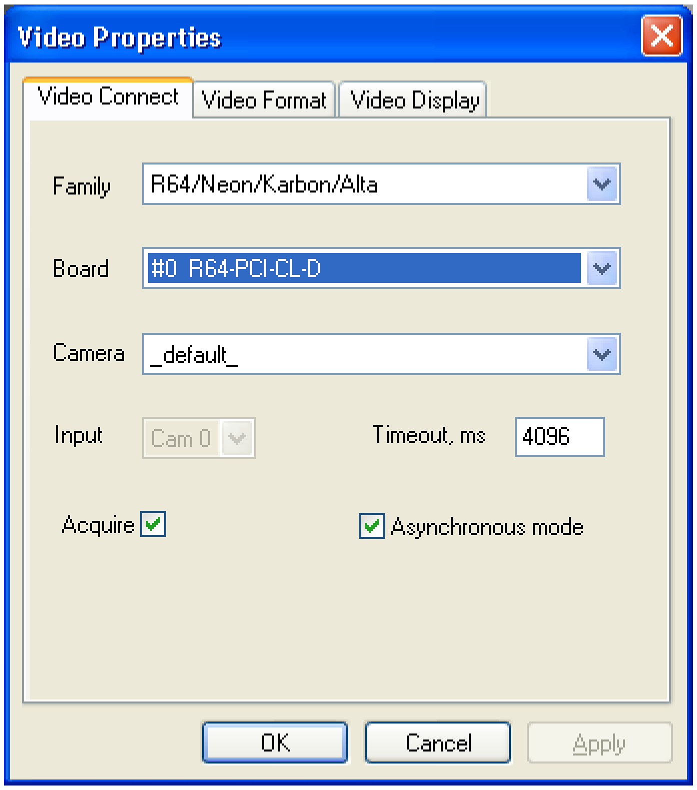
3.4 PropertyPages

The following property pages are available in *ActiveBF* control:

Video Connect	Used to select the properties specifying the source for the video input
Video Format	Used to select the properties specifying the format and synchronization of the video
Video Display	Used to select the properties specifying the display features of the video

3.4.1 VideoConnect

This property page is used to select the properties specifying the source for the video input.



Family: R64/Neon/Karbon/Alta
Board: #0 R64-PCI-CL-D
Camera: _default_
Input: Cam 0
Timeout, ms: 4096
Acquire ✓
✓ Asynchronous mode

Select from the following options:

Family

Displays the family (product line) of BitFlow boards selected for the video capture. If boards of different families are installed on your system, use this option to select the active family. Currently supported product lines are *Raven*, *Road Runner / R3*, and *R64*. Equivalent to the Family property.

Board

Displays the model of the currently selected BitFlow board and the board's number. Boards of the selected **Family** are numbered sequentially as they are found when the system boots. A given board will be the same number every time the system boots, as long as the quantity of boards remains the same and they remain in the same PCI slots. A typical selection in the **Board** field will read as follows:

#1 R64-PCI-CL

which indicates that the first *R64* board is currently selected for the video capture. If you have more than one board installed on your system, you can switch to another board of the same family (in the above example, another *R64*) by choosing the corresponding board in the list. Equivalent to the Board property.

Camera

Lets you choose a camera connected to the current **Board**. By default, *ActiveBF* will use the first camera in the list maintained by the SysReg application from *BitFlow SDK*. The name of the corresponding camera configuration file will appear in the **Camera** box. You can change the current camera by selecting the desired camera configuration file from the list, which will display all camera files available for the current **Board**. The camera file list comes from folder defined in SysReg application as the Camera configuration file path. In addition, you can use the default camera file by selecting the string "_default_" as a property value. In this case *ActiveBF* control will use the first camera from the list maintained by the SysReg application. Equivalent to the Camera property.

Input

Lets you choose one of the four analog video inputs when you use multiple cameras connected to the *Raven* board. If the currently selected **Board** is of a different type, this option will be unavailable. Equivalent to the Input property.

Timeout

Lets you select the number of milliseconds to wait for a frame to be acquired. If you acquire images with a slow frame rate, set this option to a higher value. If you set this value to zero, the timeout will never occur. Note that disabling the timeout may cause the Grab function to wait indefinitely for completing a frame. This usually happens when the signal from a camera or trigger is not coming to the board. If the board is set to the **Trigger** mode, this option will be unavailable, as the timeout will be set to infinite. Equivalent to the Timeout property.

Acquire

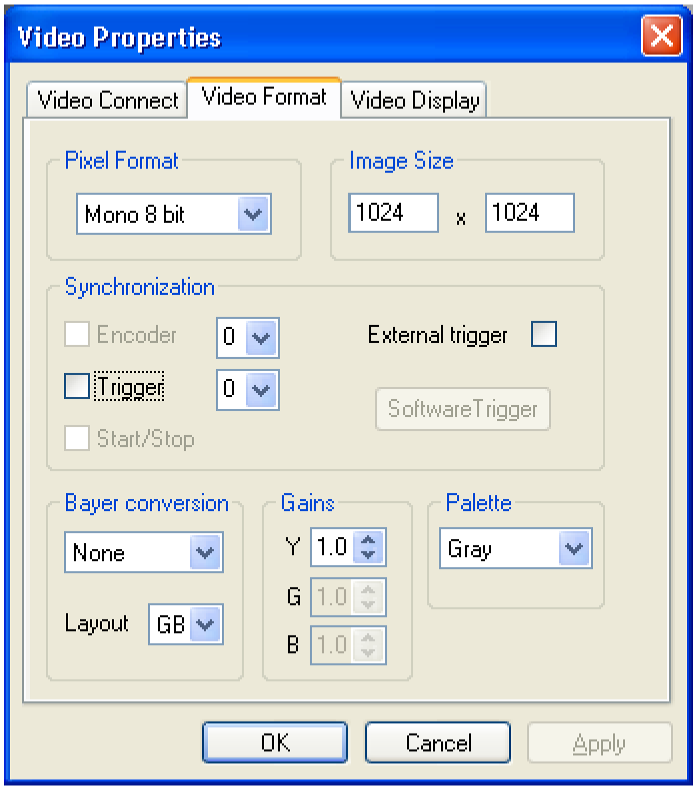
Lets you enable the continuous acquisition mode. If this box is checked, the board will continuously acquire the video into the internal image memory. If the control is visible, the live video will be displayed in the control window. Equivalent to the Acquire property.

Asynchronous mode

Lets you select an acquisition mode. Check this box to set the board to the asynchronous mode. In this mode the board will continuously transfer pixels into the host memory, allowing your application to process a captured frame while the acquisition of the next frame occurs. The asynchronous mode provides the fastest and most efficient setup for real-time image processing. However, if processing occurs at a slower rate than pixels are acquired, using this mode may result in the decomposition of images and loss of data. Clear this check box to set the board to the synchronous mode. In this mode the acquisition of the next frame will be initiated upon the capture command. The synchronous mode is slower, but it guarantees the wholeness of images during real-time processing. Equivalent to the Asynch property.

3.4.2 VideoFormat

This property page is used to select the properties specifying the format and synchronization options of the video.



Pixel Format: Mono 8 bit
Image Size: 1024 x 1024
Encoder: 0
Trigger: 0
Bayer conversion: None
Layout: GB
Gains: Y 1.0, G 1.0, B 1.0
Palette: Gray

Select from the following options:

Pixel Format
Use this list to select the pixel format of the captured video. The choices in the list will depend on the pixel depth provided by the current camera configuration. If the camera generates 8 bits per channel (i.e. 8 bpp Mono or 24/32 bpp), the **Pixel Format** box will display the current pixel depth with no other choice available. If the camera delivers high bit depth video (such as 12 bpp Mono or 30 bpp RGB), you will be able to choose among the following formats: 8-bit, 10-bit, 12-bit, 14-bit, 16-bit for the monochrome video and 24-bit, 30-bit, 36-bit, 42-bit, 48 bit for the RGB video. By default **Pixel Format** will be set to the bit depth of the current camera configuration file. Equivalent to the Format.

Image Size
Lets you change the image width and height defined by the current camera configuration. To modify the size of the video frames, enter the desired values for the image width and height. Note that the valid range of the image sizes depends on the camera associated with the board. For example, if you exceed the sensor size of the camera, you will end up with a scrambled or unusable image. Also, this option may not work with complex and/or multi-tap cameras. Equivalent to the SizeX and SizeY properties.

Encoder
Select this check box to set the board to the external horizontal synchronization mode. This mode is only used with a line scan camera, the horizontal synchronization for which is provided by a motion encoder. The encoder generates a trigger signal at regular spatial intervals, so that the lines captured are synchronous with movement of an object in the field of view. To set the board to

the internal horizontal synchronization mode, clear the **Encoder** check box. Note that this option is available only for camera configuration files that have an encoder support. Equivalent to the Encoder property.

Encoder Input
Lets you select one of three encoder inputs for an *R64* board. Choose the desired input number from the list located next to the **Encoder** check box. Note that this option will be unavailable for any board other than *R64*. Equivalent to the EncoderInput property.

Trigger
Select this check box to set the board to the trigger mode. Depending on the camera configuration file, the board will be set to either one shot or trigger acquire mode. The one shot mode is typically used with an asynchronously resetable camera. The acquisition of a frame will occur upon receiving a signal from an external hardware trigger. The trigger acquire mode will be used for free-run camera configuration files. In this mode the trigger event will cause the board to acquire an image at the start of the next frame. To set the board to the free-run mode, clear the **Trigger** check box. Note that not all camera files support switching between the trigger and free-run modes. It is recommended to use a camera configuration file specifically designed for the desired mode. Equivalent to the Trigger property.

Trigger Input
Lets you select one of three hardware trigger inputs for an *R64* board. Choose the desired input number from the list located next to the **Trigger** check box. This option is available only for *R64* board set to the **External Trigger** mode. Equivalent to the TriggerInput property.

Start/Stop
Select this check box to set the board to the start/stop mode. This mode is typically used with a line scan camera in order to initiate and finish a capture of a variable size frame. The acquisition of a frame will start when the external trigger signal asserts and end when it de-asserts. You can simulate trigger events by using the **Trigger** command twice. The size of an image captured in the start/stop mode will depend on the time interval between two trigger events. To set the board to the trigger acquire mode, clear the **Start/Stop** check box. Note that this option is available only for camera configuration files designed to support the **Start/Stop** mode. Equivalent to the StartStop property.

External Trigger
Lets you connect the external hardware trigger to the acquisition circuitry. If you do not have a hardware trigger and want to use the software trigger, clear this option to disable the trigger circuitry. If this option is checked and no trigger is connected, the board may randomly self-trigger from the electric noise on the unconnected input. Equivalent to the ExtTrigger property.

Software Trigger
Click this button to generate a software trigger signal. This option is available only if the Trigger property is set to TRUE and ExtTrigger to FALSE.   Equivalent to calling the SwitchTrigger method.

Bayer conversion
Select this option to activate the real-time color conversion of a grayscale video generated by a Bayer camera. The CCD layout of the camera can be selected from the list on the right. Both options are available only for monochrome video modes. See Bayer and BayerLayout for more information.

Gains (R, G , B , Y)
Let 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 apply 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 colorized 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 colorized. The isodense palette allows you to clearly see transitions between different levels of intensities as isolines on a topographic map.

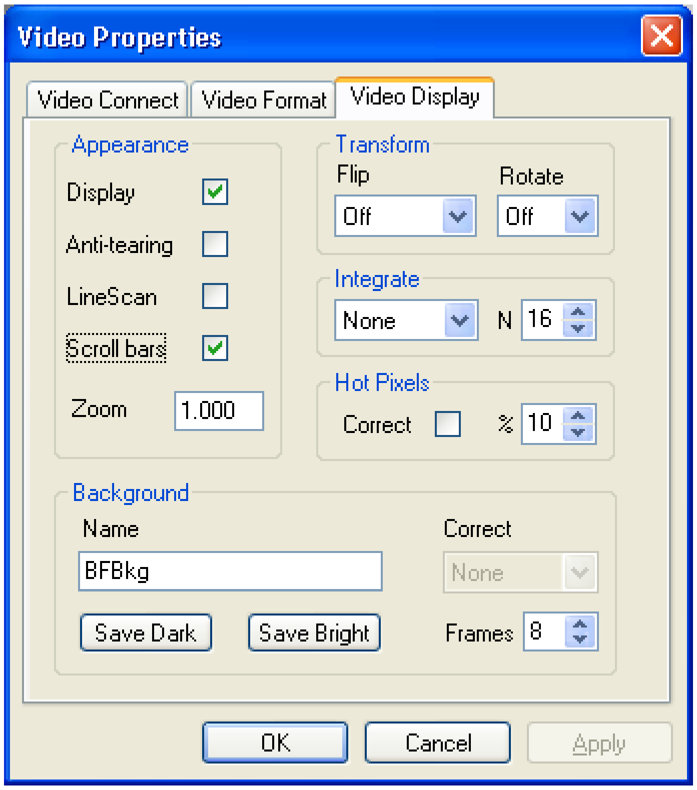
Multiphase
: Applies the multiphase palette. Entries in the multiphase palette are at opposite ends of the color model so even small changes in gray levels are highlighted.

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 VideoDisplay

This property page is used to select the properties specifying the display features of the video.


Video Properties

Video Connect | Video Format | Video Display

Appearance
- Display ✓
- Anti-tearing ☐
- LineScan ☐
- Scroll bars ✓
- Zoom 1.000

Transform
- Flip: Off
- Rotate: Off

Integrate
- None, N 16

Hot Pixels
- Correct ☐, % 10

Background
- Name: BFBkg
- Correct: None
- Save Dark, Save Bright
- Frames 8

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. 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.

LineScan
Select this check box to set ActiveBF in the LineScan mode. In this mode the control if visible will update its window per each captured horizontal line, thus allowing you to display live video captured by line scan cameras with a slow line rate. When working with regular cameras, disable this option as it can significantly reduct the performance of your application. Equivalent to the LineScan property.

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 Zoom property. *Note - if the Display option is unchecked and Zoom is set to zero, ActiveBF will enter a raw image transfer mode with the minimal CPU usage.*

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.

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 assosiated 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 avaraging 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 filed 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 avaraging the number of consecutive frames specified by the **Frames** option.

Correct background
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

ActiveBF SDK includes a *DirectShow Video Capture Filter* which allows Windows users to view/capture images from BitFlow boards and control their parameters in DirectShow-compliant applications such as *AmCap, Video Capturix, Blaze Media Pro* and others. The filter is automatically installed on your system during *ActiveBF* 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 BitFlow interfaces, with high-depth pixel modes automatically converted to 8-bit monochrome or 24-bit RGB format.

ActiveBF 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). *ActiveBF Video Capture Filter* provides a DirectShow interface to BitFlow framegrabbers. It has one output Video Capture pin which can be connected to other filters. The pin supports all video modes provided by BitFlow framegrabbers, 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 video format, *ActiveBF Video Capture Filter* delivers the following media types:

If *Display* property of *IActiveBF* 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

If *Display* property of *IActiveBF* 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 *ActiveBF Video Source Filter*, you will need to install DirectX SDK and have a general understanding of COM client programming.

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 *ActiveBF Video Source Capture Filter* does not enumerates BitFlow boards. Instead, the selection of a specific board is available ivia the Connect property page or through the IActiveBF interface. Multiple BitFlow interfaces can be accessed at the same time using several instances of the filter. See Working with multiple boards for 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 Retrieving the Filter

To instantiate *ActiveBF Video Capture Filter*, you need to enumerate all video capture filters using the system device enumerator and match the filter name with "BitFlow Camera Interface":

```
IBaseFilter* pFilter;
HRESULT hr;

//locate the 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 ActiveBF filter?
if( !memcmp( W2CA(var.bstrVal), "BitFlow Camera Interface", 24 ) )
{
hr = pM->BindToObject( 0, 0, IID_IBaseFilter, (void**)pFilter );
}
noCapDevice++;
pM.Release();
pBag.Release();
SysFreeString( var.bstrVal );
}
pEm.Release();
```

4.1.2 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 ActiveBF filter to the graph
hr = pGraph->AddFilter(pFilter, L"BitFlow board" );
if( FAILED(hr) )
{
Error( "Failed to add board to graph." );
return FALSE;
}

//specify the filter graph to the graph builder
hr = pBuilder->SetFiltergraph (pGraph);
```

4.1.3 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_IMediaContrlo> control = pGraph;

hr = control->Run();        // start the graph
////
//// ........
hr=control->Stop();         //stop the graph
```

4.1.4 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.

4.1.5 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 and RGB24. If your work with high-bit depth video formats and want to access original pixel values, use the GetImageData or GetImagePointer methods of IActiveBF interface.

4.1.6 Displaying Property Pages

BitFlow Video Capture Filter provides a pass-through access to ActiveBF 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 IUnknown 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 properties that can be modified while the graph is running, while the pin's property pages control the settings that cannot be accepted dynamically such as format change. Therefore you must stop the graph before displaying the pin's property pages and rebuild it afterwards.

4.2 Interfaces

The *ActiveBF Video Capture Filter* provides a programmatic access to video acquisition 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, contrast and gamma.
IAMVideoControl	Provides programmatic control over image flipping and camera triggering
IActiveBF	Provides a pass-through interface to *ActiveBF* COM object
ISpecifyPropertyPages	Returns a list of *ActiveBF* property pages supported by the Filter

Video Capture Pin

IAMStreamConfig	Provides an ability to retrieve and set video modes
ISpecifyPropertyPages	Provides a list of *ActiveBF* property pages supported by the Pin
IAMVideoCompression	Provides the name of a board
IAMDroppedFrames	Provides information about dropped and non-dropped frames

4.2.1 IAMVideoProcAmp

Description

Provides programmatic control over the following properties: brightness, contrast, gamma.

Methods

HRESULT Get (long *Property*, long **pValue*, long **Flags*)
Retrieves the value of a specific property.

HRESULT Set (long *Property*, long *lValue*, long *Flags*);
Sets the value of a specific property.

HRESULT GetRange (long *Property,* long **pMin,* long **pMax,* long **pSteppingDelta,* long **pDefault,* long **pFlags*);
Retrieves values associated with the range of a specified property.

Parameters

Property
[in] Value that specifies the property. Use one of the following values:

VideoProcAmp_Brightness - to set and retrieve the brightness setting.
VideoProcAmp_Gamma - to control the gamma setting.
VideoProcAmp_Contrast - to control the contrast setting.

pValue / lValue
[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] Not used.

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 sets the brightness value to 64:

```
IAMVideoProcAmp *pVideoProcAmp;
HRESULT hr;
hr = pFilter->QueryInterface(IID_IAMVideoProcAmp, (void **)&pVideoProcAmp);
if(hr==S_OK)
{
   hr=pVideoProcAmp->Set(VideoProcAmp_Brightess,64,0);
}
```

Remarks

IAMVideoProcAmp is a standard DirectShow interface. Refer to *Microsoft DirectX SDK* documentation for more details.

4.2.2 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 board 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 board 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_Brightess,64,VideoProcAmp_Flags_Manual);
}
```

Remarks

IAMVideoControl is a standard DirectShow interface. Refer to *Microsoft DirectX SDK* documentation for more details.

4.2.3 IActiveBF

Description

Provides a pass-through access to an *ActiveBF* 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 **IActiveBF** interface from the video capture filter and sets up a bilinear Bayer interpolation with GB layout:

```
IActiveBF *pActiveBF;
HRESULT hr;
hr = pFilter->QueryInterface(IID_IActiveBF, (void **)&pActiveBF);
if(hr==S_OK)
{
   hr=pActiveBF->put_Bayer(2);
   hr=pActiveBF->put_BayerLayout(0);
}
```

Remarks

Use **IActiveBF** interface to access properties and methods not available via standard DirectShow interfaces, such as Bayer conversion, trigger mode, trigger polarity, image rotation, timeout, and others. Since DirectShow does not support 16 bits per channel media types, you might want to use *IActiveBF's* image access methods for retrieving high-depth pixel data.

Refer to ActiveX Reference chapter for more details.

4.2.4 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).

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.

Remarks

Use the **GetNumberOfCapabilities** and **GetStreamCaps** methods to collect the information about available video modes. 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.

4.2.5 IAMVideoCompression

Description

Provides the name and serial number of the currently selected board.

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, such as "Camera #1 Teli CSB4000F S/N 0000204".
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 board:

```
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))
            wsprintf(camName, TEXT("%s - %s\0"), W2T(wachDesc), W2T(wachVer));
}
```

Remarks

IAMVideoCompression is a standard DirectShow interface. Refer to *Microsoft DirectX SDK* documentation for more details.

4.2.6 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 **plDropped*)
Retrieves the total number of frames that the pin dropped since it last started streaming.

HRESULT GetNumNotDropped(long **plNotDropped*)
Retrieves the total number of frames that the pin delivered downstream (did not drop).

HRESULT GetAverageFrameSize(long **plAverageSize*)
Retrieves the average size of frames that the pin dropped.

Parameters

plDropped
[out] Pointer to the total number of dropped frames.
plNotDropped
[out] Pointer to the total number of frames that were not dropped
plAverageSize
[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(&nNotDropped);
}
```

Remarks

IAMDroppedFrames is a standard DirectShow interface. Refer to *Microsoft DirectX SDK* documentation for more details.

4.2.7 ISpecifyPropertyPages

Description

Provides a list of *ActiveBF* 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 *ActiveBF* 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 adress 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 and Video Capture Pin support the Connect, Format and Display property pages. Refer to ActiveX Reference chapter for more information.

5 Troubleshooting

Below is the list of the most frequently encountered issues and remedies for their resolutions:

Problem description	**Cause**	**Resolution**
The board is not recognized by *ActiveBF*. "No BitFlow board found" message appears in the control window.	BitFlow SDK/driver is not install on your system.	Install the latest version of BitFlow SDK from www.bitflow.com
Live video is not appearing in the control window	Wrong camera file is selected. The camera is not powered up or is not properly connected to the board.	Select a correct camera file that correspond to your camera model. Once a proper camera file is found, use BitFlow's SysReg application to make it a default camera file for your board. Check the connections, replace the cable or try to run the camera/software on a different system.
When I run my application, "Board already open by another process" error pops up.	Two instances of ActiveBF are trying to access the same board. A hidden instance of an ActiveBF-based application is running on the background	If your application uses several ActiveX controls, make sure that the Board property of each control is set to a different board index. Go to the Task Manager and kill a rogue process.
I am getting "Internal SDK Error" messages	BitFlow board is not properly configured.	Run BitFlow's CiView application. If you get the same error messages, contact BitFlow technical support

Problem description	**Cause**	**Resolution**
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 try to compile your VB.NET and C# samples in Visual Studio under Vista or Windows 7, I am getting error messages.	The DEP memory access protection is enabled on your system which prevents Visual Studio from interfacing with ActiveBF.	Run the Command Prompt (as Administrator) and execute the following command: *bcdedit.exe /set {current} nx AlwaysOff* Alternatively, run the depfix.bat from the DRIVER folder (as Administrator). Reboot your computer. The problem should be fixed.
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 write permissions due to User Account Control (UAC).	Copy the Samples folder from C:\Program Files\ActiveBF\ 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 ActiveBF folder.
When I run a live video application from within Visual Studio, I get an error message "Board already opened by another process"	Live video is active in the Design mode. Visual Studio doesn't close the designer while starting the application.	*ActiveBF* cannot run two instances of live video acquired from the same board. Make sure to close the design view before running your application. The best way to avoid this problem is not to use live display in the design mode except for testing purposes. You can initiate acquisition in your code by setting the *Acquire* property to true.
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 *ActiveBF* 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 and/or CPU.

Problem description	**Cause**	**Resolution**
ActiveBF installation fails with the following error: "ActiveBF.dll failed to register, HRESULT -2147024770"	Runtimes components of Visual C++ 2005 are not installed on your system	Install Microsoft's VC++ 2005 Redistributable Package, then run *ActiveBF* installation again. For the 64-bit OS you may also need to install VC++ 2005 Redistributable Package 64-bit
ActiveBF installation fails with the following error: "ActiveBF.dll failed to register, HRESULT -2147220473"	This error is usually caused by a corrupted registration of atl.dll system file.	1. Locate atl.dll, generally found in "C:\WINNT\system32" or "C:\WINDOWS\system32". 2. Open the command-line prompt • 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: • o regsvr32 "C:\WINNT\system32\atl.dll" • o press "ENTER" 4. You should see a regsvr32 window, confirming success: Close it. 5. Repeat *ActiveBF* installation.

6 Samples

The ActiveBF distribution package includes the following sample applications:

Programming language	Project name	Description
Visual Basic 6.0	BFProfile	Live image preview, real-time line profile, drawing text, rectangles and ellipses
	VBProcess	Live image processing with direct pixel manipulation in the frame buffer
	BFStat	Real-time image statistics over an ajdustable luminance range
	BFEnhance	Real time frame averaging and integration, software gain control, hot pixel correction
	BFAlpha	Animation effects in the alpha-plane over the live video, full screen mode
	BFByRef	Live image in PictureBox object, pixel values at cursor coordinates, fps display, using *ActoveBF* by reference
	BFSerial	Serial read and write operations over the CameraLink serial port
	DcamAVI (DVR-version)	Video-capture and playback to/from AVI files with the sound recording
	DcamSequence (DVR-version)	Video capture and playback to/from memory sequence
	DcamReplay (DVR-version)	Live video with delayed replaying using memory loop recording
Visual C++, MFC	ActiveDemo	Live image preview, real-time line profile, histogram and image statistics over ROI, saving image into file, continuous capture into multiple frames.
Visual C++, Win32 API	BFConsole	Console application showing how to use ActiveBF API to retrieve the list of connected cameras and video formats, capture a series of frames and modify camera properties.
	BFWin	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	BFCap	Live image preview, built in and custom camera controls. Based on ActiveBF Video Capture Filter and Microsoft DirectX SDK.
VB.NET	BFOverlay	Live image preview, real time pixel values, overlay animation
	BFCapture	Time-lapse video capture to AVI files or

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 BFOverlay and BFSharp samples assume the presense of MSCOMM32.ocx on the system.

All the samples include complete source code and executables which can be run with any BitFlow board out of the box. Simply connect one or more cameras to your BitFlow boards, start an application of your choice and see how *ActiveBF* performs in real world!

Index