



User Guide

Version 2.0

Copyright © 2004-2006 by A&B Software LLC

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

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

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

ActiveBF is a trademark of A&B Software LLC.

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

Third Edition

March 2006

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

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

Table of Contents

Part I Introduction	4
1 License Agreement	5
2 System Requirements	7
3 Installation	8
Part II Getting started	10
1 Visual Basic	11
2 Visual C++	13
3 VB.NET	17
4 Visual C#	20
Part III Reference	22
1 Properties	24
Acquire	26
Asynch	27
BackColor	28
Bayer	29
BayerLayout	31
Board	33
Brightness	35
Camera	36
Contrast	38
Display	39
Edge	41
Encoder	42
EncoderInput	43
ExtTrigger	44
Family	45
Format	46
Gamma	48
Input	50
LineScan	51
Overlay	53
OverlayColor	54
OverlayFont	55
Palette	56
ScrollBars	58
ScrollX	60
ScrollY	61
SizeX	62
SizeY	63
StartStop	64
Timeout	65
Trigger	66

TriggerInput	68
Zoom	69
2 Methods	71
Draw	73
GetBytesPerPixel	74
GetComponentData	75
GetComponentLine	77
GetDIB	79
GetImageData	81
GetImageWindow	83
GetInputBit	85
GetPicture	86
GetPixel	87
GetPointer	89
GetRGBPixel	91
GetLine	93
Grab	95
OverlayClear	96
OverlayEllipse	97
OverlayLine	98
OverlayPixel	99
OverlayRectangle	100
OverlayText	101
SaveImage	102
SerialClose	104
SerialConnect	105
SerialOpen	107
SerialRead	109
SerialWrite	110
SetImageWindow	111
SetOutputBit	113
ShowProperties	114
StartCapture	115
StopCapture	117
SwitchTrigger	118
3 Events	119
FormatChanged	120
FrameAcquired	121
FrameAcquiredX	123
LineAcquired	124
MouseDown	125
MouseDown	126
MouseMove	127
MouseUp	128
Overflow	129
Scroll	130
Timeout	131
4 PropertyPages	132
Color	133
VideoConnect	134
VideoFormat	136
VideoDisplay	138

Part IV Samples

141

Index

143

1 Introduction

ActiveBF is an ActiveX control built around BitFlow SDK and designed for rapid application development tools, such as Visual Basic, VB.NET, Visual C++, C#, Java, VBA, etc. With *ActiveBF* control your application immediately supports all of BitFlow's current camera interface products: the *Road Runner*, the *Raven*, the *R3*, and the *R64*.

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
- Adjust hardware brightness, contrast, and gamma.
- 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.
- Save images in BMP, TIF and JPEG formats with adjustable compression.
- Communicate with cameras via Camera Link serial port.
- Control digital input/output on the board.

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.

[License agreement](#)

[System requirements](#)

[Installation](#)

Distributing your application

1.1 License Agreement

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

1. Subject of agreement

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

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 Control* 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 Control*, 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 Control* based programs. Likewise, the end user must not be given the *ActiveBF Control* serial number.

4. Royalties

The *ActiveBF Control* 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 Control*.

5. Redistributable files

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

6. 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 Control*. 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:

- alter, translate, decompile, or to disassemble *ActiveBF Control*
- copy *ActiveBF Control's* accompanying written documentation
- lend, hire out or lease *ActiveBF Control*.

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

7. Exclusion of warranties

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

8. 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 Control*. 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.

9. 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 Control* are to be destroyed.

1.2 System Requirements

Hardware requirements:

- Pentium II 400 MHz or better CPU recommended.
- 64 Mb or more RAM recommended.
- A graphic card supporting 65535 colors or higher
- One or more BitFlow framegrabbers installed on the system.

Software requirements:

- Windows NT/2000/XP
 - BitFlow SDK 4.00 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 NT/2000/XP, you are recommended having administrator-level access.

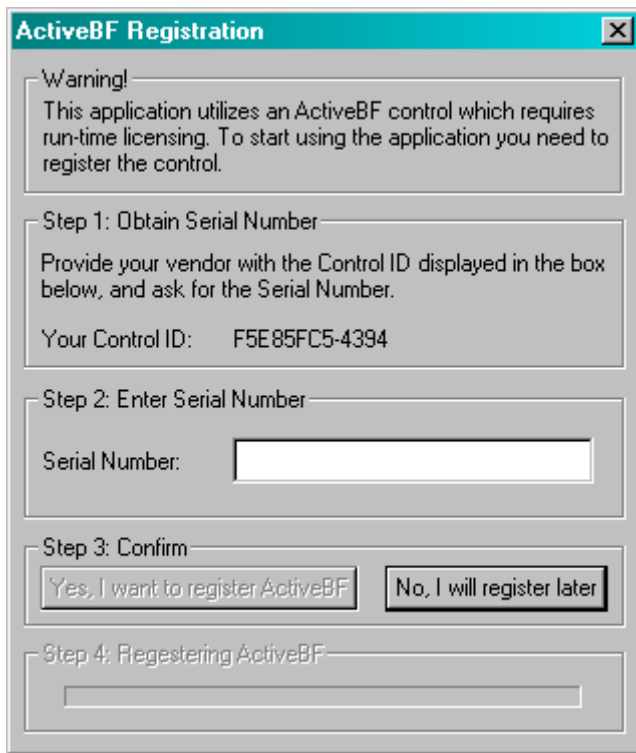
To install *ActiveBF*, perform the following steps:

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 the control, 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 VB development environment, you should perform a run-time registration. This is done through the following procedure.

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



The image shows a dialog box titled "ActiveBF Registration" with a close button (X) in the top right corner. The dialog is divided into four sections:

- Warning!**: A text box containing the message: "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**: A text box containing the message: "Provide your vendor with the Control ID displayed in the box below, and ask for the Serial Number." Below this is a label "Your Control ID:" followed by the value "F5E85FC5-4394".
- Step 2: Enter Serial Number**: A text box with the label "Serial Number:" followed by an empty input field.
- Step 3: Confirm**: Two buttons: "Yes, I want to register ActiveBF" and "No, I will register later".
- Step 4: Registering ActiveBF**: A text box that is currently empty.

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

-

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

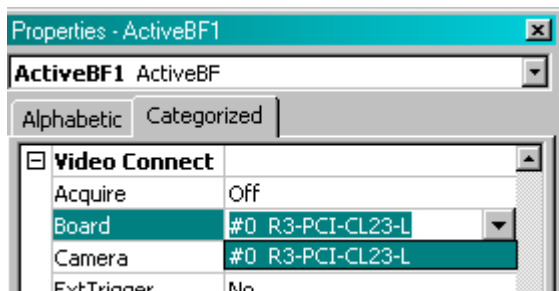
Assuming that you have already run the *ActiveBF* installation program and started Visual Basic, the next step is to create a project. Begin by selecting the *New Project* command from the file menu, and select *Standard EXE* as your project type. Then use the *Project / Components...* command and select *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 appear on the form, and the *Project* window on the right will display *ActiveBF*'s properties.

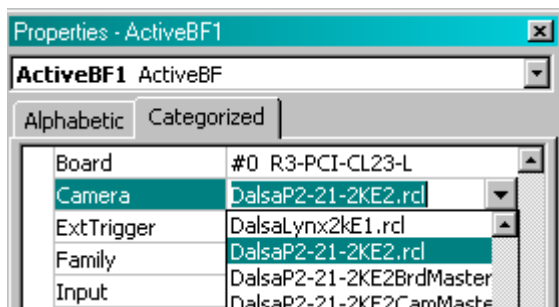
Selecting the Board

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:



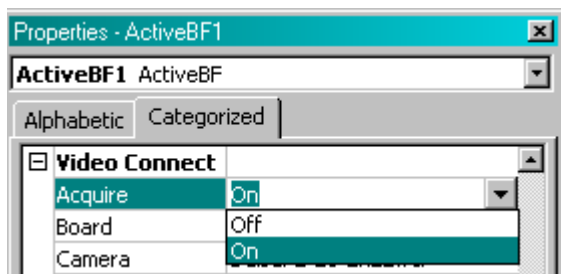
Selecting the Camera

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:



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++ 7.0 (.NET)

In the .NET development environment select *New -> Project*. The *New Project* Dialog box will appear. Select *Visual C++ projects* on the left and *MFC Application* on the right. In the *Name* field below type the name of your application, for instance *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

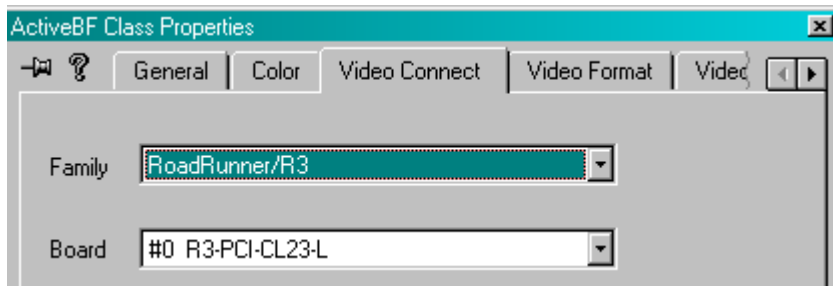
Right click in the dialog and select *Class Wizard* from the shortcut menu. When *MFC Class Wizard* appears, click the *Member Variables* tab. In the *Control ID* window click *IDC_ACTIVEBF1* and then click *Add Variable*. Confirm generating the new *CActiveBF* class. When the *Add Member Variable* dialog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

VC++ 7.0 (.NET)

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.

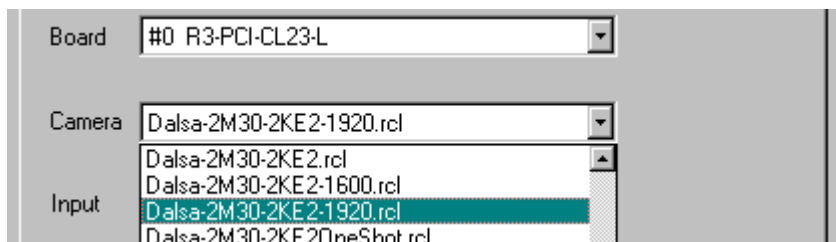
Selecting the Board

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.



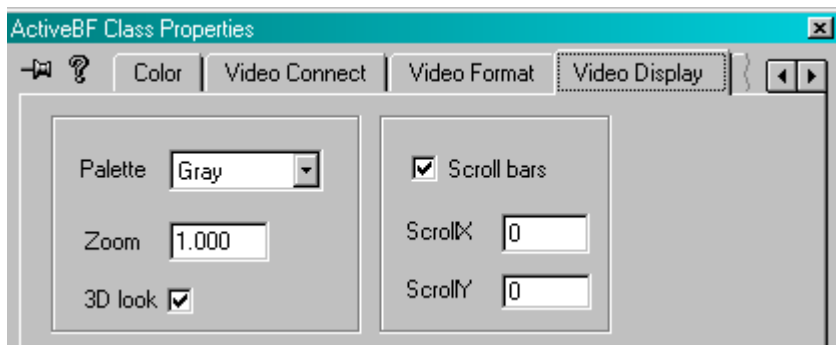
Selecting the Camera

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:



Modifying the Control's appearance

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



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);
}
```

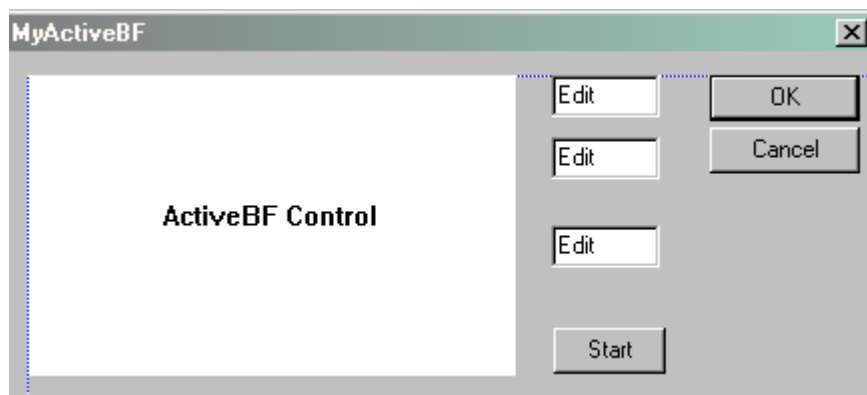
VC++ 7.0

```
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* field 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

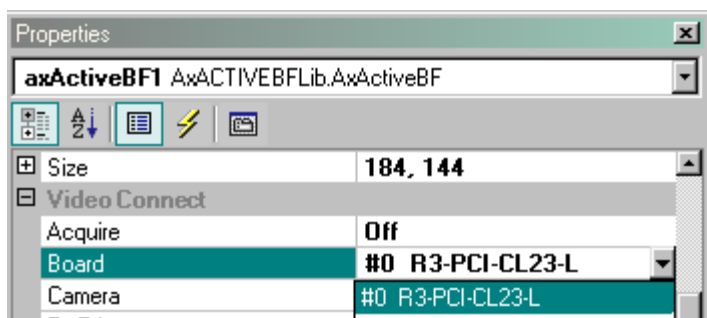
In the *Toolbox* select *Components*. From the *Tools* menu select *Customize Toolbox* and then select *ActiveBF Class* from the list. You will see *ActiveBF* icon appear at the bottom of the toolbox.



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.

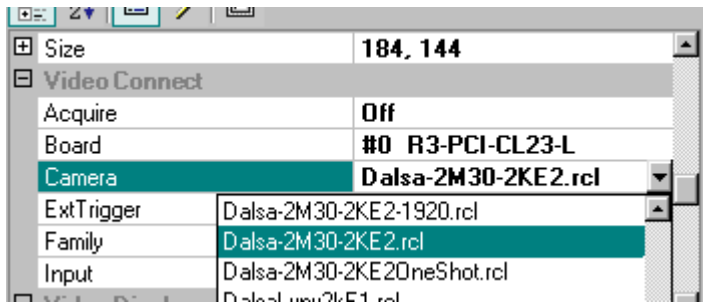
Selecting the Board

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:



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:



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
```

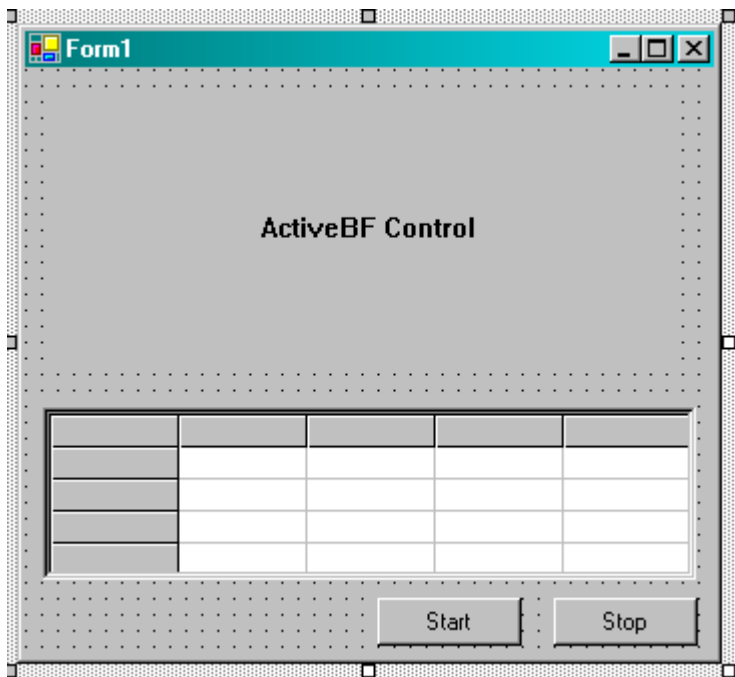
Repeat the same procedure for the *Stop* button adding the following line to the *Button2_Click* function:

```
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 rows and columns added to the grid on the main form.

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



Adding the FrameAcquired event

Double-click in the 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* field 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

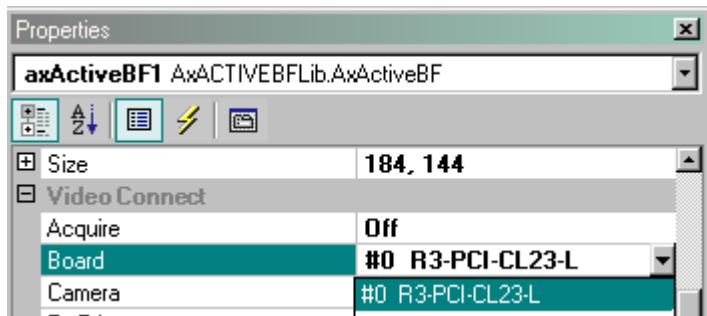
In the *Toolbox* select *Components*. From the *Tools* menu select *Customize Toolbox* and then select *ActiveBF Class* from the list. You will see *ActiveBF* icon appear at the bottom of the toolbox.



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.

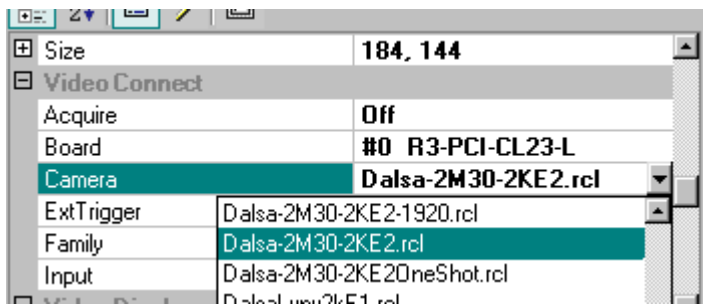
Selecting the Board

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:



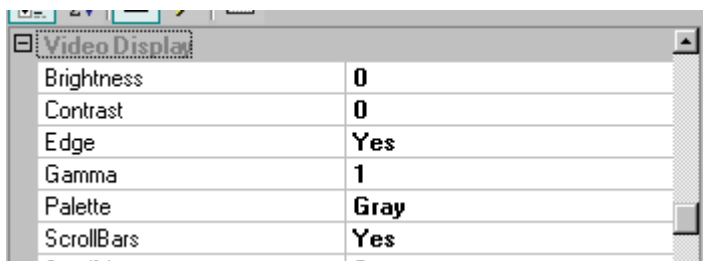
Selecting the Camera

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:



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"



Adding the Start button

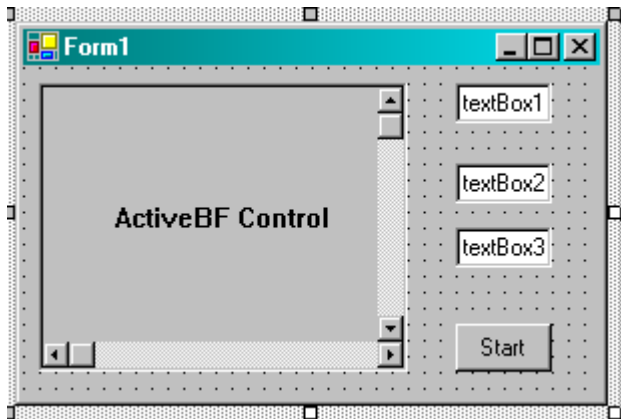
In the *Toolbox* select *Windows Form*, click on the *Button* icon and then draw a rectangular area on the program dialog. A button "Button1" will appear. Go to the *Property* window and change the *Text* property to "Start". Double click on the button. A new member function *button1_Click* will be added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

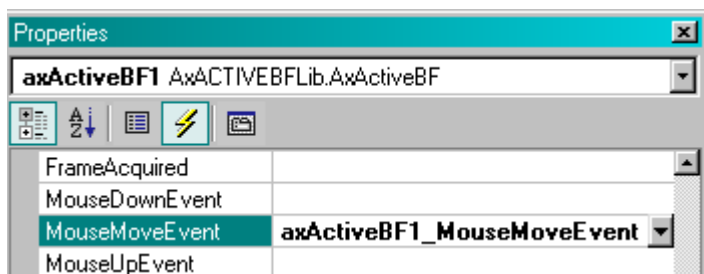
Adding text boxes

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*.



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!

3 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

<u>BackColor</u>	Returns or sets the background color of the control window
<u>Display</u>	Enables/disables automatic live display in the control window.
<u>Overlay</u>	Enables/disables the overlay
<u>OverlayColor</u>	Returns or sets the color of the overlay graphics
<u>OverlayFont</u>	Returns or sets the font for drawing text in the overlay

Video Connect Category

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

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 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.

3.1.3 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*.

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

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

Return Values

S_OK
Success
E_FAIL
Failure.

Example

This VB example activates the bilinear Bayer filter:

```
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 (demaosaicing). 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.

3.1.5 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]
objActiveDcam.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.

3.1.6 Board

Description

Returns or sets the ordinal number of the currently selected BitFlow board. 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.Board = 1  
MsgBox ActiveBF1.Board
```

Remarks

The valid **Board** property values are 0-3. 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.

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.

3.1.7 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.

3.1.8 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.

3.1.9 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.

3.1.10 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.

3.1.11 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.

3.1.12 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.

3.1.13 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.

3.1.14 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.

3.1.15 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 product lines are Raven, Road Runner/R3, and R64.

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 are: "Raven", "RoadRunner/R3", "Road Runner", "R2", "R3", "R64", non case-sensitive. If you try to set an invalid value, an error will occur.

3.1.16 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.

3.1.17 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.

3.1.18 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.

3.1.19 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.

3.1.20 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](#).

3.1.21 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](#).

3.1.22 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](#).

3.1.23 Palette

Description

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

0 - Gray

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

1 - Inverse

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

2 - Saturated

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

3 - Rainbow

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

4 - Spectra

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

5 - Isodense

Applies the 256-level grayscale palette, each 8-th entry of which is 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.

3.1.24 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.

3.1.25 ScrollX

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.

3.1.26 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.

3.1.27 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.

3.1.28 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.

3.1.29 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.

3.1.30 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.

3.1.31 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 resettable 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.

3.1.32 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.

3.1.33 Zoom

Description

Returns or sets the zoom factor for the live video display. The zero value will fit the image to the size of the control window.

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]
The zoom factor to be set.

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 live 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. 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.

3.2 Methods

ActiveBF control provides the following methods to a container application:

<u>Grab</u>	Grabs a single frame into the internal memory
<u>Draw</u>	Displays the current frame in <i>ActiveBF</i> window.
<u>SwitchTrigger</u>	Asserts/deasserts the software trigger signal
<u>GetBytesPerPixel</u>	Returns the number of bytes per pixel in the video
<u>GetPixel</u>	Returns the pixel value at the specified coordinates
<u>GetRGBPixel</u>	Returns the array of RGB values at the specified coordinates
<u>GetLine</u>	Returns the array of pixel values in the specified horizontal line
<u>GetComponentLine</u>	Returns the array of pixel values of the color component in the specified horizontal line
<u>GetImageData</u>	Returns the two dimensional array of pixel values in the currently acquired frame
<u>GetComponentData</u>	Returns the two dimensional array of pixel values in the specified color component
<u>GetImageWindow</u>	Returns the 2D array of pixel values in the specified rectangular area of the current frame
<u>SetImageWindow</u>	Copies the 2D array of pixel values to the selected window of the current frame
<u>GetPointer</u>	Returns the memory pointer to the specified pixel
<u>GetDIB</u>	Returns the handler to a Device Independent Bitmap
<u>GetPicture</u>	Returns the Picture object corresponding to the currently acquired frame
<u>SaveImage</u>	Saves the current frame buffer in the specified image file
<u>StartCapture</u>	Starts video capture to the specified AVI file or series of images
<u>StopCapture</u>	Stops video capture
<u>SerialOpen</u>	Opens the serial device on the Camera Link board
<u>SerialConnect</u>	Selects the number of the serial device on the Camera Link board
<u>SerialRead</u>	Reads a string of characters from the currently open Camera Link serial device
<u>SerialWrite</u>	Writes the string of characters to the currently open Camera Link serial device
<u>SerialClose</u>	Closes the currently open Camera Link serial device

<u>GetInputBit</u>	Reads the state of the digital input bit
<u>SetOutputBit</u>	Sets the state of the digital output bit
<u>OverlayClear</u>	Clears graphics and text from the overlay
<u>OverlayEllipse</u>	Draws an empty or filled ellipse in the overlay
<u>OverlayLine</u>	Draws a line in the overlay
<u>OverlayPixel</u>	Draws a pixel in the overlay
<u>OverlayRectangle</u>	Draws an empty or filled rectangle in the overlay
<u>OverlayText</u>	Draws a string of text in the overlay
<u>ShowProperties</u>	Displays property pages in run-time mode

3.2.1 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.

3.2.2 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 outputted 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 16-bit monochrome format, the resulting video will have 3 or 6 bytes per pixel accordingly.

3.2.3 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. This is opposite to the order of rows in the array returned by [GetImageData](#).

If the video has a grayscale format, the *Component* parameter will have no effect and the output array will contain the luminance data.

For integer (word) type of data you can receive negative numbers if pixel values exceed 32767. In C and C# you can convert signed integers to unsigned ones by using data casting. To get rid of negative values in VB and Delphi, subtract them from 65535 (see the example above).

3.2.4 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.

3.2.5 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

pDib
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 [GetPointer](#) methods.

3.2.6 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 [GetPointer](#).

Image in *ActiveBF* are stored bottom up, therefore the first element of the array is first pixel of the

bottom line of the image. The type of data and dimensions of the array returned by **GetImageData** depends on the [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](#).

3.2.7 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. This is opposite to the order of rows in the arrays returned by [GetImageData](#). The type of data and dimensions of the array returned by **GetImageWindow** depends on the output format of the video, the width and height of the window and the number of lines acquired, as specified in the following table:

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 accommodate 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, subtract them from 65535.

Note that modifying elements of the array will not change actual pixel values in the frame buffer. Use [SetImageWindow](#) to transfer pixel values into *ActiveBF*.

3.2.8 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.

3.2.9 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
```

Remarks

The **GetPicture** method provides the most convenient 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 [GetPointer](#) methods.

3.2.10 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.

3.2.11 GetPointer

Description

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

Syntax

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

```
[C/C++]  
HRESULT GetPointer( 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.GetPointer(0,32);  
BYTE* ptr=ActiveBF1.GetPointer(0,32);  
memcpy(&line,v.pVal,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.GetPointer(0, 32);
int a = (int)obj;
byte* ptr= (byte*)a;
for (int x=0; x<sx; x++, ptr++)
*ptr=0;
}
```

Remarks

The **GetPointer** 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.

3.2.12 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.

3.2.13 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, subtract them from 65535 (see example above).

3.2.14 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.

3.2.15 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](#).

3.2.16 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](#).

3.2.17 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](#).

3.2.18 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](#).

3.2.19 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](#).

3.2.20 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](#).

3.2.21 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.

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.

3.2.22 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](#)

3.2.23 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.

3.2.24 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

3.2.25 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](#)

3.2.26 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](#)

3.2.27 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.

3.2.28 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.

3.2.29 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
```

3.2.30 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

ActiveBF records 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.

3.2.31 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](#).

3.2.32 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.

3.3 Events

The following events are generated by *ActiveBF* control:

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

3.3.1 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.

3.3.2 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)
    Labell.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.

3.3.3 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)  
    Labell1.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.

3.3.4 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)
    Labell.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.

3.3.5 MouseDbIcIck

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_MouseDbIcIck( ByVal X As Integer, ByVal Y As Integer )
```

[C/C++]

```
HRESULT Fire_MouseDbIcIck( 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 **MouseDbIcIck** event to open the *Properties* dialog:

```
Private Sub ActiveBF1_MouseDbIcIck(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.

3.3.6 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.

3.3.7 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.

3.3.8 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.

3.3.9 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.

3.3.10 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.

3.3.11 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_Timeout();
```

Parameters [C/C++]

None

Return Values

S_OK
Success
E_FAIL
Failure.

Example

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

```
Private Sub 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.

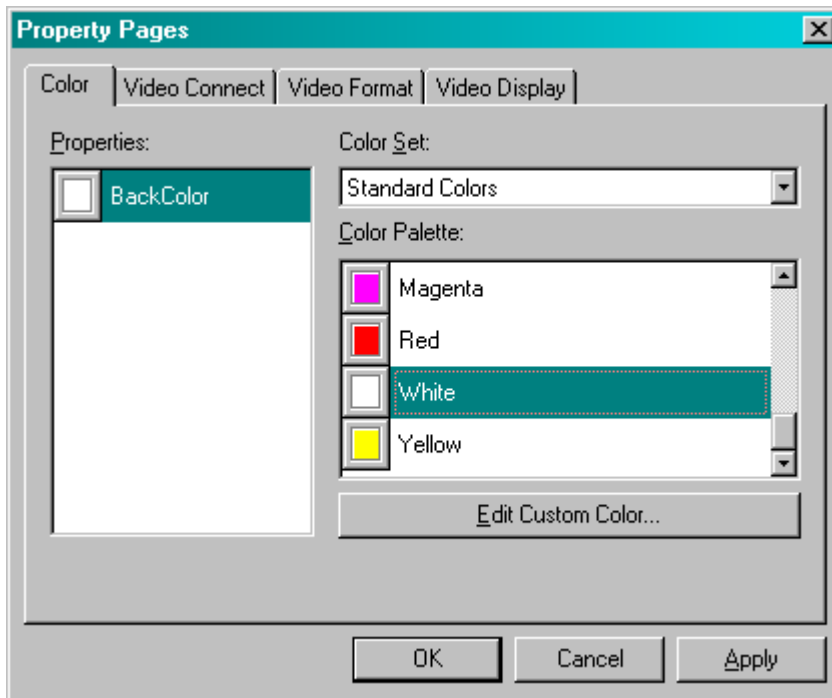
3.4 PropertyPages

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

Color	Used to select the background color of the control window and the overlay color
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 Color

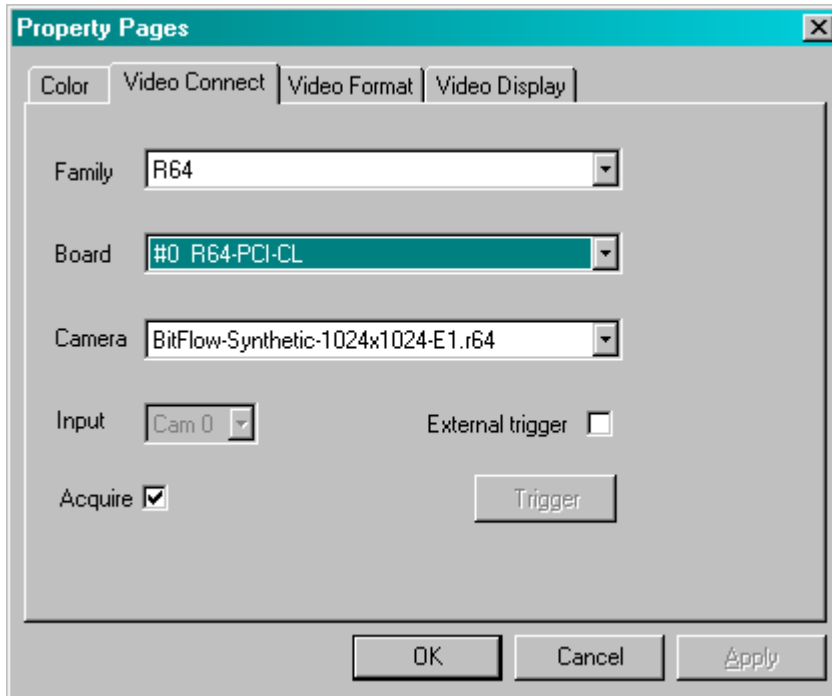
This property page is used to select the background color of the control window.



Select the desired color from the color palette on the right. See [BackColor](#) for more details.

3.4.2 VideoConnect

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



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.

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.

External Trigger

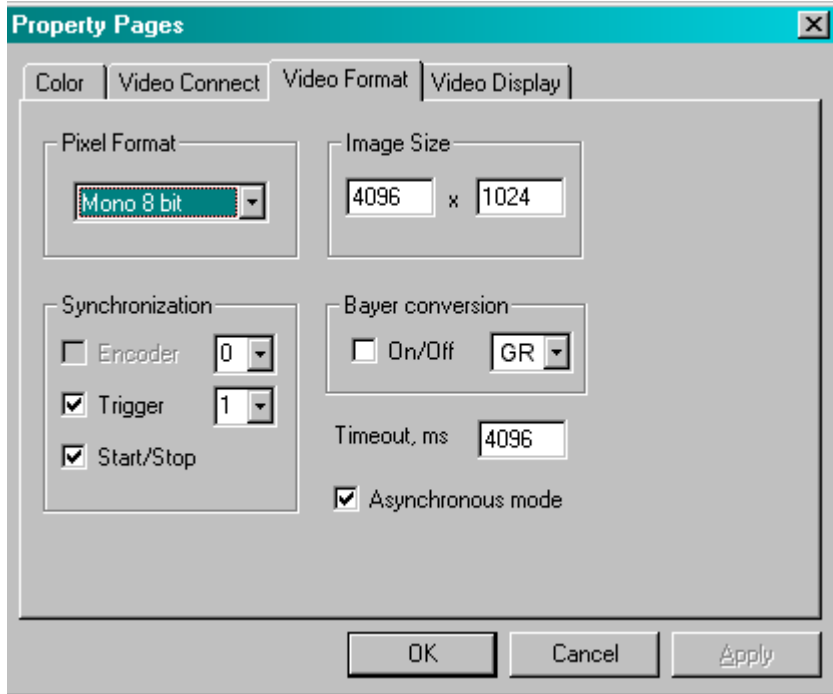
Lets you connect the external hardware trigger to the acquisition circuitry. If you do not have a hardware trigger and want to use the software trigger, clear this option to disable the trigger circuitry. If this option is checked and no trigger is connected, the board may randomly self-trigger from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

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.

3.4.3 VideoFormat

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



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 resettable 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.

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.

Timeout

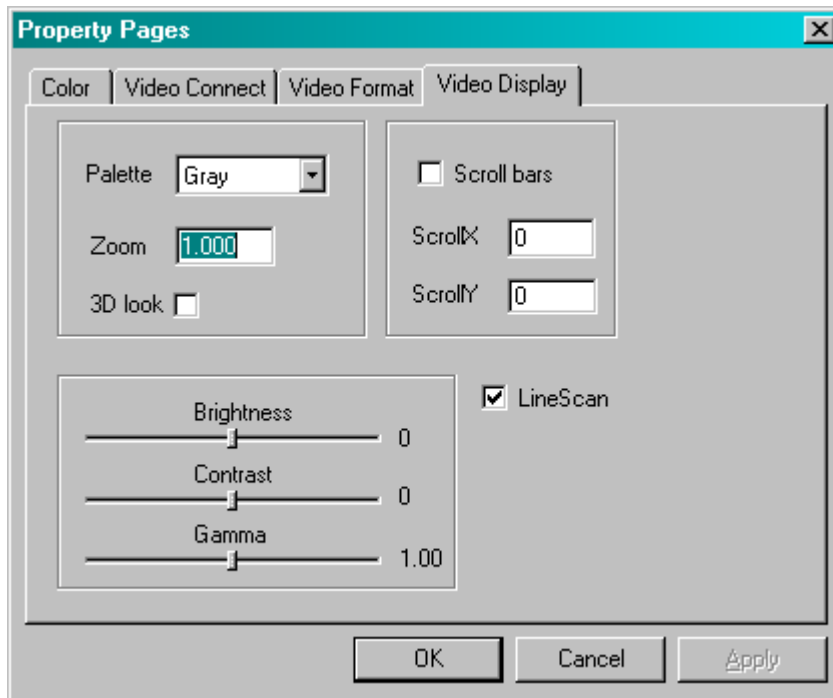
Lets you select the number of milliseconds to wait for a frame to be acquired. If you acquire images with a slow frame rate, set this option to a higher value. If you set this value to zero, the timeout will never occur. Note that disabling the timeout may cause the [Grab](#) function to wait indefinitely 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.

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.

3.4.4 VideoDisplay

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



Select from the following options:

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.

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.

3D look

Lets you enable or disable 3D-appearance of the control window. If this box is checked, the sunken edge will appear around the border of the control window. Equivalent to the [Edge](#) property.

Scroll bars

Lets you enable or disable the scroll bars in the control window. If this box is checked and the video width or/and height exceed the size of the control window, the scroll bar(s) will be displayed on the border of the control window allowing you to pan the live video. Equivalent to the [ScrollBars](#) property.

Gray

ScrollX, ScrollY

Lets you select the horizontal and vertical scroll positions for the live video display. To modify the scroll position, enter the desired value in the corresponding box. Note that the values must be given in the image coordinate system, not in the screen coordinates. Equivalent to the [ScrollX](#) and [ScrollY](#) properties.

Brightness

Move the slider to adjust the brightness (average intensity level) of the video by modifying the hardware look-up table. If the currently selected board does not have a LUT, this option will be unavailable. Equivalent to the [Brightness](#) property.

Contrast

Move the slider to adjust the contrast ((the difference in intensity between the dark and light areas) of the video by modifying the hardware look-up table. If the currently selected board does not have a LUT, this option will be unavailable. Equivalent to the [Contrast](#) property.

Gamma

Move the slider to adjust 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 an image and increase the contrast in its darker areas, increase the gamma by moving the slider to the right or indicating a value greater than 1. To darken the image and emphasize the contrast in the lighter areas, decrease in the gamma by moving the slider to the left or indicating a value less than 1. If the currently selected board does not have a LUT, this option will be unavailable. Equivalent to the [Gamma](#) 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 reduce the performance of your application. Equivalent to the

[LineScan](#) property.

4 Samples

The ActiveBF distribution package includes the following sample applications:

Programming language	Project name	Description
Visual Basic 6.0	BFProfile VBProcess BFByRef	Live image preview, real time pixel values, real-time line profile Live image processing with direct pixel manipulation in the frame buffer Live image in PictureBox object, pixel values at cursor coordinates, fps display, using <i>ActoveBF</i> by reference
Visual C++ 6.0	ActiveDemo	Live image preview, real time pixel values, real-time line profile, saving image into file, continuous capture into multiple frames
VB.NET	VBOverlay VBCapture	Live image preview, real time pixel values, overlay animation Time-lapse video capture to AVI files or series of images, overlaid frame counter
Visual C#	FilterSharp TwoBoardsDemo	Direct access to the frame buffer using pointers, real-time Emboss filter, non-destructive color overlay, interactive drawing Simultaneous capture from two framegrabbers, real time access to both image arrays.

Index

- 3 -

3D look 41

- A -

Acquire 26
Asynch 27
Asynchronous acquisition 27
AVI 115, 117

- B -

Background Color 28
Bayer filter 29
Bayer layout 31
bmp 102
Board 33
Brightness 35

- C -

C# 20
C++ 13
Camera 36
Color 133
Continuous acquisition 26
Contrast 38

- D -

Display 39
Draw 73

- E -

Edge 41
Encoder 42
EncoderInput 43
Events 119
External Trigger 44
ExtTrigger 44

- F -

Family 45
Format 46
FormatChanged event 120
FrameAcquired 121
FrameAcquiredX event 123

- G -

Gamma 48
GetComponentData 75
GetComponentLine 77
GetDIB 79
GetImageData 81, 111
GetImageWindow 83
GetInputBit 85, 113
GetLine 93
GetPicture 86
GetPixel 87
GetPointer 89
GetRGBPixel 91
Grab 95
Guide 10

- H -

Height 63

- I -

Input 50
Installation 8
Introduction 4

- J -

jpeg 102

- L -

License 4, 5
LineAcquired 124
LineScan 51
Look-up table 35, 38, 48

- M -

Methods 71
MouseDown event 125
MouseDown 126
MouseMove 127
MouseUp 128

- O -

Overflow 129
Overlay 53
OverlayClear 96
OverlayColor 54
OverlayEllipse 97
OverlayFont 55
OverlayLine 98
OverlayPixel 99
OverlayRectangle 100
OverlayText 101
Overview 4

- P -

Palette 56
Pixel depth 74
Properties 24
Property pages 114, 132
Pseudo colors 56

- R -

R3 45
R64 45
Raven 45
Reference 22
Requirements 7
Road Runner 45

- S -

Samples 141
SaveImage 102
Scroll 130
ScrollBars 58

ScrollX 60
ScrollY 61
SerialClose 104
SerialConnect 105
SerialOpen 107
SerialRead 109
SerialWrite 110
ShowProperties 114
SizeX 62
SizeY 63
Software trigger 118
StartCapture 115
StartStop 64
StopCapture 117
SwitchTrigger 118
Synchronous acquisition 27

- T -

tiff 102
Timeout 65, 131
Trigger 66
TriggerInput 68

- V -

VB 11
VB.NET 17
Video Connect 134
Video Display 138
Video Format 136
Visual Basic 11
Visual C# 20
Visual C++ 13

- W -

Width 62

- Z -

Zoom 69